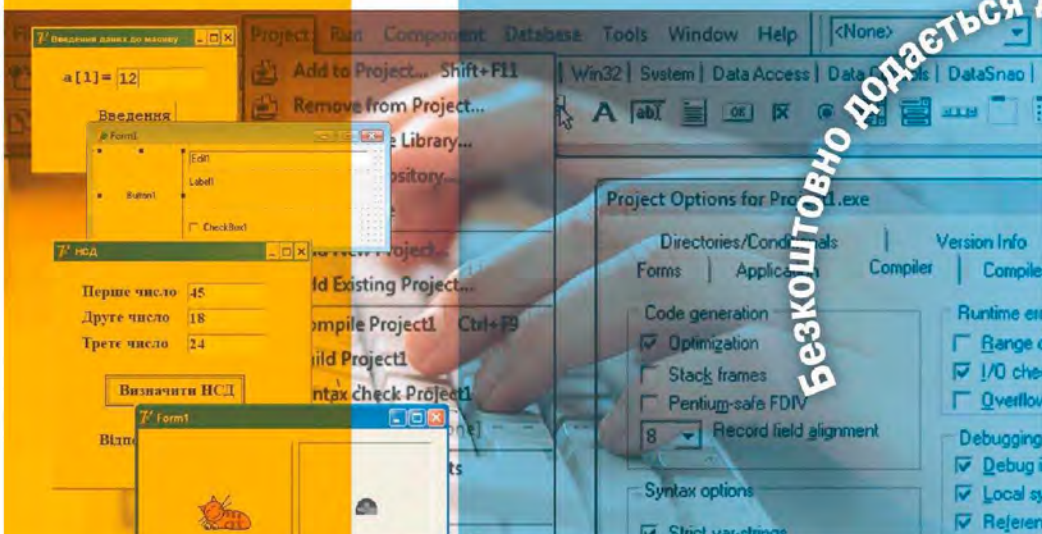


КУРС ЗА ВИБОРОМ

Л. Б. Кащєєв, С. В. Коваленко, С. М. Коваленко

Основи візуального програмування

Навчальний посібник



ВИДАВНИЦТВО
РАНОК

Рекомендовано для використання
в навчально-виховному процесі

КУРС ЗА ВИБОРОМ

ЦИФРОВА
ІНФОРМАТИКА

ВИДАВНИЦТВО
РАНОК

Л. Б. Кащеев, С. В. Коваленко, С. М. Коваленко

Основи візуального програмування

Навчальний посібник

Рекомендовано для використання
в навчально-виховному процесі

УДК 004.42(076)
ББК 32.973я721
К31

Відповідає програмі курсу за вибором
«Основи візуального програмування», рекомендованій
МІНІСТЕРСТВОМ ОСВІТИ І НАУКИ УКРАЇНИ
(лист від 10.08.2006 р. № 1/11-4927)

Видано за ліцензією ТОВ Видавництво «Ранок»

Рецензенти:

О. С. Куценко, зав. кафедри системного аналізу і керування
Національного технічного університету «ХПІ», д-р техн. наук, професор;
М. М. Корнієнко, учитель інформатики вищої категорії
Харківської гімназії № 55, учитель-методист, заслужений учитель України

Кащєєв Л. Б.

К31 Інформатика. Основи візуального програмування: Навч. посібник / Л. Б. Кащєєв,
С. В. Коваленко, С. М. Коваленко. — Х.: Веста, 2011. — 192 с. — (Курс за вибором).

ISBN 978-611-540-765-1

Пропонований навчальний посібник є складовою частиною навчально-методичного комплексу з основ візуального програмування, до складу якого також входять зошит для практичних робіт і методичні рекомендації для вчителів.

Метою посібника є навчання основ сучасного програмування. Видання орієнтоване на старшокласників, які роблять перші кроки у візуальному програмуванні мовою Delphi.

УДК 004.42(076)
ББК 32.973я721

ISBN 978-611-540-765-1

© Л. Б. Кащєєв, С. В. Коваленко, С. М. Коваленко, 2010
© ТОВ Видавництво «Ранок», 2011

ВСТУП

Сучасне програмування нерозривно пов'язане з візуальними середовищами. Абсолютна більшість прикладних програм — редакторів, баз даних, електронних таблиць, тестерів, веб-додатків і ігор, працюють у вікнах візуального середовища. Із цієї причини і їх програмування має вестися відповідними мовами. Це видання пропонує старшокласникам опанувати візуальне програмування в середовищі Delphi.

Чому саме Delphi? По-перше, через відносну простоту цієї мови. Нагадаємо, що мова Object Pascal, яка є основою Delphi, спочатку створювалася як навчальна, а тому багато алгоритмічних структур писати нею значно простіше, ніж іншими мовами. Дехто може сказати, що Visual Basic ще простіша. Безумовно, але розробка серйозних програм мовою з неявним оголошенням типів набагато складніша. По-друге, у результаті компіляції в Delphi утворюються досить ефективні виконувані модулі (exe-файли), які за швидкістю виконання не порівнянні з жодною мовою-інтерпретатором. Це дозволить тим, хто опанував Delphi, брати участь в олімпіадах, змаганнях зі спортивного програмування й у конкурсах Малої академії наук. По-третє, знаючи принципи візуального програмування, надалі програміст легко зможе перейти на інші візуальні мови, наприклад C++ Builder або C#.


Цей посібник має на меті навчити візуального програмування від самих азів. Учні послідовно вивчають усі етапи готування програми, починаючи з питань алгоритмізації та правил задавання алгоритмів і закінчуючи доволі складними проблемами зіставлення різних методів пошуку й сортування даних у масиві. При цьому той, хто опановує візуальне програмування, не має необхідності попередньо вивчати яку-небудь мову програмування: у книжці докладно описано типи даних, операції, правила побудови виразів, принципи формування умов (логічних виразів).

Посібник має 11 розділів. Навчальний матеріал кожного розділу об'єднано в теми відповідно до програми курсу за вибором «Основи візуального програмування», рекомендованої Міністерством освіти і науки України.

Кожен розділ посібника супроводжується численними прикладами, що ілюструють матеріал, викладений у ньому. Багато місця у виданні відведено опису інструментального середовища. Не секрет, що, запустивши перший раз візуальне середовище розробки програм, програміст-початківець губиться через велику кількість меню, вікон,

кнопок і закладок. Тому автори посібника визнали за необхідне при розгляді більшості прикладів зазначати, до якого пункту меню варто звернутися, з якої закладки мають бути перетягнуті на форму відеокомпоненти, де розташований той або інший налагоджуваний параметр компонента.

У посібник включено 13 практичних робіт. У кожній із них містяться докладно розглянутий приклад, у якому крок за кроком описано дії з проектування форми й наповнення методів відеокомпонентів необхідним програмним кодом, і задачі для самостійного розв'язування. Усі завдання дібрано так, щоб їх можна було виконати протягом однієї академічної години.

Посібник має додаток у вигляді компакт-диску. На ньому є програми більшості розв'язаних у книжці прикладів і завдань практичних робіт — у посібнику вони позначені значком . Приклади, подані на компакт-диску, мають наскрізну нумерацію. Таким чином, учень, розв'язуючи приклад із чергового розділу або завдання практичної роботи, може скористатися наявним у додатку проектом, що містить повністю готову форму з відеокомпонентами, програму, записану в методи (процедури) відповідних елементів форми, а якщо необхідно, то й базові зображення, з яких будувався розроблювальний інтерфейс.

Обравши серед спецкурсів саме «Основи візуального програмування», у цій книжці ви одержите можливість опанувати цей предмет на відносно простих і зрозумілих проектах. Ми намагалися зробити так, щоб кожен урок і в цілому курс були цікавими й дали поштовх до майбутніх професійних успіхів учнів.



У цьому розділі ми звернемося до історії обчислювальної техніки, розглянемо хронологію появи перших програм для комп'ютерів і перших мов програмування. Тут буде введено важливі поняття: «програма», «середовище розробки програм», «дані», «алгоритм» тощо. Поговоримо також про професію програміста.

1.1

Історія мов програмування. Поняття програми, мови програмування, компілятора та інтерпретатора. Сучасні мови програмування

⇒ Історія мов програмування

Офіційно вважається, що першою мовою програмування стала мова опису алгоритмів для машини Чарльза Беббіджа. У 1843 р. цією мовою Ада Авґуста Лавлейс склала програму — «План операцій для аналітичної машини, за допомогою яких можна розв'язати рівняння Бернуллі». Однак Беббідж не довів свою машину до робочого стану, тому мова її програмування становить лише історичний інтерес (рис. 1.1).

14 лютого 1946 р. почала працювати в штатному режимі перша електронна обчислювальна машина ENIAC (*Electronical Numerical Integrator and Calculator* — електронний числовий інтегратор і обчислювач), створена на замовлення американської Лабораторії балістичних досліджень для розрахунків артилерійських таблиць (рис. 1.2). Слідом за цією машиною з'явилося чимало близьких до неї параметрами: Mark I, EDSAC, EDVAC, МЭСМ. Кожна з них мала свою мову програмування з числовими позначеннями команд і змінних (як змінні

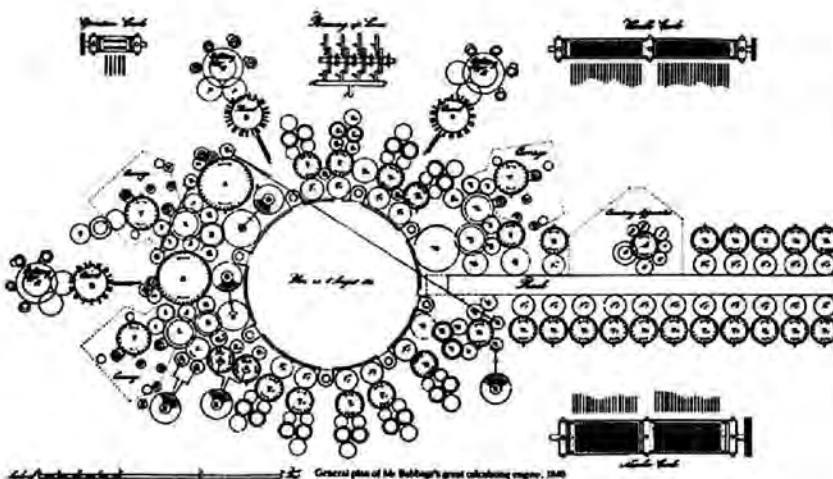


Рис. 1.1. Креслення з патенту на аналітичну машину Беббіджа і автор першої програми для цієї машини Ада Авґуста Лавлейс

застосовували номери «комірок пам'яті», де вони зберігалися). Це були мови низького рівня.

У період від 1954 до 1957 р. група програмістів під керуванням Джона Бекуса в американській корпорації IBM створила першу мову програмування високого рівня — Fortran. Європейською альтернативою цій мові став Algol, розроблений у 1958–1960 рр. комітетом IFIP. Algol був краще пристосований для реалізації складних структур даних і алгоритмів. Наприкінці 1960-х рр. було здійснено кроки з подальшого вдосконалення мови. У конкурсі зі створення нової версії мови Algol-68 з'явилася нова, схожа на Algol мова, яка згодом стала мовою програмування Pascal, а потім — Delphi.

Основи мови Pascal розробив у 1967–1968 рр. професор Ніклаус Вірт (Niklaus Wirth). Стандарт мови він зареєстрував у 1974 р. разом із Кетлін Єнсен (Kathleen Jensen). Мова Pascal досить швидко перетворилася із засобу, призначеного для навчання програмування, в інструмент для творення великих програмних проектів. Зауважимо, що жодна з наступних мов, розроблених Н. Віртом (Modula і Oberon), не набула такого поширення. Популярність мови Pascal, можливо, пояснюється тим, що в той час у розпорядженні програмістів було небагато мов високого рівня.

Мова Fortran, що її один із піонерів програмування Е. Дейкстра (Edsger Dijkstra) назвав «дезорганізатором», призначалася тільки для математичних розрахунків. Вона прекрасно працювала з математичними формулами й алгоритмами, але не витримувала жодної критики



Рис. 1.2. Перша електронна обчислювальна машина ENIAC. Її програму було написано в машинних кодах — мови програмування ще не існувало

в плані роботи зі складними структурами даних (графіка, бази даних, електронні таблиці), до того ж вона не мала засобів для роботи із зовнішніми пристроями.

➔ Поняття програми, мови програмування, компілятора та інтерпретатора

Програма — це послідовність команд, за допомогою яких записують алгоритм розв'язування задачі. Програміст пише програму якоюсь мовою програмування. *Мова програмування* — це формальна мова, яка має всі необхідні засоби для того, щоб перетворити правильно описаний алгоритм на послідовність команд, що забезпечують розв'язання задачі. Очевидно, що мова програмування орієнтована на зручність сприйняття програмістом задачі, тож для опису дій у ній доречні слова природної мови (зазвичай англійської).

Формалізація мови — це внесення у неї набір правил про те, які слова можна використовувати як операторів, а яких не можна, і в якій послідовності записуються оператори та їхні атрибути. Її мета полягає, звичайно, не в тому, щоб заплутати програміста й зробити мовні засоби недоступними для рядових користувачів. Коректно написана програма, в принципі, так само незрозуміла обчислювальній машині, як, наприклад, і наше вільне словесне спілкування: машині потрібні конкретні машинні коди операцій і адреси в оперативній пам'яті, де містяться ті чи інші дані. Це завдання розв'язують дві групи програм — компілятори й інтерпретатори.

Компілятор — це спеціальна програма, що переводить весь текст вихідної програми в зрозумілий для обчислювальної машини об'єктний код, у якому чітко розписано, як розташовуються змінні в пам'яті, у якій послідовності будуть виконуватися команди, а якщо операції складні — які спеціальні бібліотеки буде підключено для їх виконання. Компілятор обов'язково знайде помилку в описах програми незалежно від того, чи вона в 1-му, чи в 1001-му рядку. Помилкова команда буде або незрозуміла компілятору, або хибно інтерпретована.

По-іншому працює програма-інтерпретатор. *Інтерпретатор* — це спеціальна програма, яка зчитує текст вихідної програми рядок за рядком і кожен прочитаний рядок перетворює на машинні коди й запускає на виконання. Для коротких програм інтерпретатор працює швидше за компілятор — доки той «збере» увесь виконуваний модуль, доки встановить зв'язок між командами і змінними, інтерпретатор уже запустив на виконання перші рядки. Однак було б помилкою вважати, що така «швидка» інтерпретація дозволить одержати такий же «швидкий» розрахунок. Інтерпретатор, виконуючи перші рядки програми, навіть не намагається виділити пам'ять під змінні, описи яких зустрінуться наприкінці програмного тексту. Уявіть-но таку ситуацію. Вам потрібно розмістити в готелі велику кількість приїжджих, при цьому їх слід оптимально розподілити в номери (у ближчих номерах — неспокійних жителів, які постійно вимагають до себе уваги персоналу; у міру

виїзду гостей потрібно, щоб у послідовності номерів не залишалося незайнятих кімнат). А ви навіть не знаєте, коли приїдуть гості і скільки їх буде! З такою проблемою зустрічається інтерпретатор. Друга важлива проблема — час виконання програми. Виконати кілька програмних рядків інтерпретатор може дуже швидко, але як тільки текст програми більшає, пооператорна інтерпретація тексту стає тривалішою, ніж компіляція всієї програми в цілому. Та й виконуються інтерпретовані програми довше за компільовані, оскільки тут нереальними є спроби оптимізувати програму (викинути повторювані описи, об'єднати команди, що стосуються тих самих аргументів, і т. д.).

Згадані вище мови Fortran, Algol, Pascal мали компілятори.

У 1963 р. виникла спрощена мова програмування Basic. Інтерпретатор Basic (тоді ще без префікса Visual) не підходив для розв'язування серйозних задач. Цією мовою важко було програмувати, а використання інтерпретатора замість компілятора робило надзвичайно повільним процес виконання програм. Життєздатність мови підтримувалася штучно: спочатку — шляхом прошивання її в ОЗП перших персональних комп'ютерів. Деяким користувачам подобалося, що Basic можна було завантажити на ввімкненому комп'ютері без установлення його на машині. «Друге дихання» Basic одержав завдяки продуктам фірми Microsoft — довгий час, якщо ви хотіли додати можливостей і сервісу в Microsoft Office, писати додаткові програми належало на Visual Basic. Тепер же зі зміною орієнтації фірми Microsoft на мову C# майбутнє Basic виглядає досить туманним.

Мову Cobol створювали для розв'язування економічних задач; її основною перевагою була можливість іменувати змінні довгими, з багатьма літерами ідентифікаторами. Наприклад, там, де мовою Fortran змінну називали PROCENT, у Cobol можна було написати змінну ВІДСОТОК_НАРАХУВАННЯ_НА_ДОДАТКОВУ_ВАРТІСТЬ. У решті Cobol повторював ідеї мови Algol, і з розвитком Delphi його популярність різко впала.

Мова PL/1 довгий час була найпотужнішою алгоритмічною мовою, однак у 1976 р., коли розпочали випуск серійних персональних комп'ютерів, ще не було транслятора PL/1 для «персоналок». Коли ж нарешті такий транслятор з'явився, конкурувати з новими мовами Turbo Pascal і C стара PL/1 була не в змозі.

➡ Сучасні мови програмування

Після докладного розгляду мов програмування — тих, що існували й існують, — мимоволі виникає питання: які ж мови варто вчити нині?

Питання це дуже непросте. У програмуванні мета розробки часто диктує, яку мову програмування необхідно використати. Якщо метою є програмування обчислювальних задач або керування зовнішніми пристроями й організація складного інтерфейсу, то краще використовувати Delphi або C++ (ідеться саме про мови — кожна з них має чимало реалізацій і версій, як вдалих, так і не дуже).

Програмування в комп'ютерних мережах, залежно від завдання, потребуватиме знань мов PHP, Java, Pearle і використання багатьох спеціальних бібліотек, написаних для них.

Робота з базами даних вимагає знання мови SQL, що входить до складу поширеного пакета систем керування базами даних (MySQL, Oracle, MS SQL Server).

У програмуванні простих іграшок і анімованих веб-сторінок визначним лідером є мова скриптів flash (пакет Flash Macromedia тощо).

Іншими словами, єдиної мови програмування «на всі випадки життя» не існує, та й, напевно, не буде створено ніколи. З другого боку, багато які з перелічених мовних засобів мають настільки подібні алгоритмічні конструкції, що, володіючи кількома з цих мов, можна в разі потреби легко перейти на іншу. Щоб проілюструвати останнє твердження, наведемо як приклад програму простої задачі кількома мовами.

Задача. Підсумувати непарні числа з таблиці (масиву), що містить 100 цілих чисел.

Нижче наведено розв'язання задачі кількома мовами, при цьому описи типів і операції введення-виведення не подані.

Мова Pascal (Delphi):

```
sum:=0;
for i:=1 to 100 do
  if odd(a[i]) then
    sum:=sum+a[i];
```

Мова Basic:

```
10 LET SUM=0
20 FOR I=1 TO 100
30 IF NOT ODD(A(I)) THEN GOTO 50
40 LET SUM=SUM+A(i)
50 NEXT I
```

Мова Cobol:

```
СУМ_НЕПАР
MOVE 0 TO СУМ
PERFORM АНАЛІЗ_ОДНОГО_ЧИСЛА VARYING I FROM 1 UNTIL I > 100
АНАЛІЗ_ОДНОГО_ЧИСЛА
DIVIDE 2 INTO A(I) GIVING ПІВЧИСЛА RMAIND ОСТАЧА
IF ОСТАЧА IS EQUAL TO 1 ADD A(I) TO СУМ
```

Мова Lisp:

```
(DEFUN СУМНЕПАР
(LAMBDA (ЧИСЛА)
(COND
((NULL ЧИСЛА)0))
((ODD(CAR ЧИСЛА)(PLUS(CAR ЧИСЛА)(СУМПАР(CDR ЧИСЛА))))
(T(СУМНЕПАР (CDR ЧИСЛА))))))
```

Порівнюючи наведені тексти, бачимо, що текст мовою Delphi (Pascal) має трохи простіший вигляд, що цілком зрозуміло, бо цю мову розроблено в тому числі і як навчальну алгоритмічну.

⇒ Історія Delphi (Pascal)

Розглянемо хронологію розвитку мов Pascal і Delphi.

- 1967 р. — Н. Вірт запропонував Algol-W (версія Algol-60), мову не прийнято;
- 1968 р. — поява першої версії Pascal в ETH на основі Algol-W;
- 1971 р. — Єнсен і Вірт опублікували теоретичний опис мови;
- 1975 р. — перша програмна реалізація Pascal на машині CDC-6000;
- 1979 р. — поява версії Apple-Pascal;
- 1982 р. — прийнято міжнародний стандарт ISO (британський аналог — BS 6192);
- 1983 р. — у Делавері (США) Філіпп Кан зареєстрував фірму Borland International і мову Turbo Pascal; випущено декілька версій;
- 1993 р. — створено орієнтований на Windows Borland Pascal 7.01;
- 1994 р. — поява першої версії Delphi 1;
- 1997 р. — створено Delphi 3 (найнадійніша з трьох перших версій);
- 1998 р. — створено Delphi 4 (відмінності від базового Object Pascal настільки значні, що Delphi починають називати не візуальним середовищем, а мовою програмування);
- 2002 р. — розроблено Delphi 7;
- 2006 р. — розроблено Delphi 8;
- 2007 р. — розроблено CodeGear RAD Studio.

У березні 2008 р. було оголошено про припинення дальшого розвитку Delphi. Однак у серпні 2008 р. компанія Embarcadero, новий господар CodeGear, оприлюднила прес-реліз мови Delphi 2009 for Win32. Можливості нового програмного продукту були на порядок вищими, ніж у попередніх пакетів. Разом з тим стався відхід від колишнього «демократичного» принципу, коли мовне середовище могло успішно

працювати на персональних машинах із досить середніми характеристиками. Новий CodeGear відносно швидко завантажується й ефективно працює тільки на двоядерних комп'ютерах частотою 2 ГГц і більше з обсягом пам'яті не менш ніж 2 Гбайт. Із цієї причини на багатьох шкільних олімпіадах і змаганнях зі спортивного програмування, як і раніше, широко використовується Delphi 7.

Завершуючи з історією Delphi, нагадаємо, що саме на цій мові базуються візуальне середовище Borland C++ Builder, мультимедійний плеєр AIMP, програма для голосового зв'язку по Інтернету Skype, програми для мережевого обміну інформацією QIP і The Bat!, веб-редактор Quick Page 2008, бухгалтерська програма «Парус» і комп'ютерні ігри «Age of wonders», «Космічні рейнджери», «Space Empires», «Битва героїв» та багато інших відомих продуктів.

? ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Чим уславилися Чарльз Беббідж і Ада Августі Лавлейс?
2. Як називалася перша електронна обчислювальна машина? Для чого вона була призначена?
3. Дайте визначення поняття «мова програмування».
4. Що собою являють компілятор і інтерпретатор? У чому їхні відмінності?
5. Назвіть відомі вам мови програмування.
6. Укажіть основні етапи розвитку мов Pascal і Delphi. Які відомі програмні продукти були розроблені мовою Delphi?

1.2

Алгоритм, основні властивості алгоритмів

Алгоритм — це базове поняття інформатики, тому єдиного визначення цього поняття немає. Проте спроби сформулювати таке визначення здійснювалися не раз і тривають донині.

Слово «алгоритм» походить від імені великого середньоазіатського вченого Мухаммеда аль-Хорезмі, який жив у першій половині IX ст. (точні роки його життя невідомі, але вважається, що він народився близько 780 р., а помер близько 850 р.). «Аль-Хорезмі» означає «з Хорезму» (історична область у нинішньому Узбекистані, центром якої було місто Хіва).

Наведемо лише два визначення алгоритму.

«Алгоритм — це набір правил, що визначає послідовність операцій для розв'язування конкретної множини задач і має п'ять важливих рис: скінченність, визначеність, введення, виведення, ефективність» (Д. Е. Кнут, автор відомого багатотомника «Мистецтво програмування»).

«Алгоритм — це точний припис, що визначає обчислювальний процес, який іде від варійованих вихідних даних до шуканого результату» (А. Марков).

Існують також інші визначення алгоритму, і ці визначення явно чи неявно задають загальні риси, властиві алгоритмам:

- *дискретність* — процес розв'язування задачі подається як послідовне виконання простих кроків;
- *визначеність* (детермінованість) — кожен наступний крок однозначно обумовлений станом програми (системи) в теперішній момент;
- *зрозумілість* — усі команди алгоритму однозначно «зрозумілі» обчислювальній машині (виконавцеві);
- *масовість* — алгоритм може бути виконаний для різних наборів допустимих вхідних даних;
- *результативність* — виконання алгоритму має привести до очікуваного результату;
- *скінченність* — результат може бути отриманий за обмежене число кроків.

Розглянемо визначення кожної властивості алгоритму.

Дискретність. Процес розбивається на ряд послідовних приписів:

$$1 \rightarrow 2 \rightarrow 3.$$

Визначеність. Кожен виконавець, якому зрозумілий алгоритм, одержить один і той самий результат.

Зрозумілість. Алгоритм подається у вигляді тексту, зрозумілого для виконавця.

Масовість. Алгоритм справджується для різних наборів даних.

Результативність. Після точного виконання всіх запропонованих дій неодмінно має бути отримано результат.

Скінченність. Кількість послідовних дій не є нескінченною, їх можна порахувати:

$$1. \dots \quad 2. \dots \quad 3. \dots \quad N_{\text{останньої}}$$

Розглянемо ці властивості на прикладах алгоритмів у словесній формі.

Дискретність

Вдалий опис алгоритму: «*Махнула красуня правою рукою — виникло озеро пречудове, потім лівою рукою махнула — попливли озером лебеді...*» (Опис подано крок за кроком: спершу озеро, потім лебеді, потім буде ще палац.)

Невдалий опис алгоритму: «*Цап меле, цап меле, коза насипає, а маленьке козенятко на скрипочку грає*». (З опису не ясно, коли починається черговий крок і що відбувається по його завершенні.)

Зрозумілість

Вдалий опис алгоритму: *«Інструкція українською мовою з картинками»*.

Невдалий опис алгоритму: *«Інструкція на дощечках, написаних мовою ронгоронго»* (до речі, досі не розшифрованих).

Визначеність

Вдалий опис алгоритму: *«На дубі — скринька, у скриньці — качка, у качці — яйце, у яйці — голка, а в голці — Коцієва смерть»*.

Невдалий опис алгоритму: *«Піди туди, не знаю куди, принеси те, не знаю що»*.

Масовість

Вдалий опис алгоритму: *«Кожній дочці батько привіз по дорогому подарунку»*.

Невдалий опис алгоритму: *«Принц міг одружитися тільки зі справжньою принцесою»*. (У казці не визначено правило, до яких вихідних даних можна застосувати зазначену дію, лише наприкінці виробляється критерій — з периною й горошиною, але він підходить до єдиної вхідної змінної.)

Результативність

Вдалий опис алгоритму: *«Мишка бігла, хвостиком махнула — яйце впало й розбилось»*.

Невдалий опис алгоритму: *«Баба яйце біла-біла — не розбила, дід бив-бив — не розбив...»*

Скінченність

Вдалий опис алгоритму: *«Мама зварила смачну кашу в горщику»*.

Невдалий опис алгоритму: *«Каша вже заповнила всі вулиці, а горщик усе варив і варив»*.

➡ Основні відомості про професію програміста

Програміст — одна з найпопулярніших професій останніх десятиліть. Кількість обчислювальних машин невинно зростає, у зв'язку з чим виникає потреба в численних фахівцях, які мають обслуговувати й програмувати ці машини, оформлювати одержувані результати. Ця сфера діяльності належить до інформаційних технологій (*information technology*). Цим терміном називають технології керування та оброблення інформації з використанням обчислювальної техніки.

То чим займається програміст? У 1950–60-х рр., коли обчислювальних машин було мало і вони сильно відрізнялись одна від одної, люди, які працювали в обчислювальних центрах, займалися всім — від контролю працездатності техніки та її ремонту до програмування й налагодження завдань, а також оформлення результатів. Відтоді минуло всього 50–60 років, але для інформатики вони стали цілою епохою. Зараз у світі безліч обчислювальних машин. Абсолютна більшість із них — персональні комп'ютери й ноутбуки. Машини схожі одна на одну як зовнішніми пристроями (наприклад клавіатурою), так

і встановленими програмами. Щоб не перелічувати, які програми встановлено на конкретному комп'ютері, всі разом ці програми називають програмним забезпеченням.

Загалом створення нової програми складається з кількох етапів:

- визначення завдання (на цьому етапі вирішується, що робитиме розроблювана програма, за яких вхідних даних і як можна буде нею керувати);
- розробки алгоритму (на цьому етапі програма розбивається на окремі кроки, які будуть виконуватись один за одним; одні з них дуже прості, інші ж вимагають серйозних знань у математиці, графіці, у предметній галузі розроблюваної програми);
- вибір програмних засобів (тут розробник програми вирішує, на якому програмному забезпеченні належить вести програмування завдання, при цьому часто доводиться використовувати декілька програм: мови програмування, спеціальні бібліотеки, графічні, анімаційні й музичні редактори);
- великі проекти рідко робить одна людина, тому важливим етапом є поділ програми на окремі незалежні модулі, що їх можуть писати й випробовувати декілька людей незалежно один від одного, а потім збирати в загальну програму;
- окремим питанням у розробленні є інтерфейс програми — зовнішній вигляд запущеної програми на екрані монітора з кнопками, картинками, елементами введення-виведення, що дозволяють користувачеві (який не завжди є програмістом) застосовувати написану програму;
- важливими частинами сучасних програм є звуковий супровід, графіка, елементи мультиплікації, система допомоги (Help) і супровідна документація;
- коли всі складники програми готові, настає один із найважчих етапів — її остаточне збирання з тих складників і тестування одержаної програми в цілому; іноді тестування буває настільки складним, що програмістські фірми випускають програми для пробного використання майбутніми користувачами (так звані beta-версії програм);
- коли програма готова, з метою її захисту від копіювання програмісти застосовують різні засоби, щоб обмежити доступ до програми (паролі для встановлення на машину), час використання чи кількість запусків програми (trial-версії) або її можливості в разі безкоштовного встановлення (shareware). Іноді в Інтернеті виставляють не саму програму, а її опис із миттєвими знімками екрана (screen-shot) або анімаційними роликами, що показують можливості розробленої програми (trailer).

Для великої програми всі перелічені операції є дуже складним завданням як на одну людину. Тому в наш час програміст спеціалізується тільки на деяких із цих етапів. Традиційно

програмістом називають людину, котра пише програми, програмний код. Однак із появою візуального програмування пов'язувати поняття «робота програміста» з тим, скільки він написав рядків діючих програм, буде не зовсім правильно: великі фрагменти програмного коду вже вписано в компоненти, з яких будується розроблюваний проект. У цьому посібнику як приклади буде наведено хороші, ефективні програми, що, проте, містять мінімальне число рядків програмного коду. І діятимуть вони анітрохи не гірше, ніж якби ви всі дії, що відбуваються на екрані, програмували вручну у вигляді операторів (рядків розробленої програми).



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Наведіть приклади визначення алгоритму.
2. Перелічіть властивості, які має алгоритм. Поясніть їх.
3. У чому полягають особливості професії програміста?
4. Назвіть етапи створення програми.



Під час вивчення цього розділу ми ознайомимося з візуальним середовищем програмування Delphi та зробимо перші кроки у візуальному проектуванні додатків для Windows: створимо власну форму, дізнаємося про події, на які вона реагує, розробимо просту процедуру для оброблення події. У результаті в нас вийде перший власний проект у візуальному середовищі, який ми збережемо. Наприкінці розділу буде розглянуто, які файли з'являються на диску після збереження проекту, описано їхнє призначення.

2.1

***Знайомство з візуальним середовищем програмування.
Елементи вікна середовища програмування.
Програмна розробка й файли, які входять до її складу.
Створення найпростішого проекту, його компіляція,
збереження, виконання***

➤ Елементи вікна середовища програмування

Запускаючи програму сучасною алгоритмічною мовою, ми потрапляємо в середовище візуального програмування. Інтерфейс такого середовища може бути різним, що залежить від мови програмування й версії програми. Наприклад, на рис. 2.1 це середовище пакета Delphi 7, однак якщо запустити версію Delphi CodeGear 2009 або 2010, вигляд екрана буде іншим, хоча всі ці середовища використовують одні й ті самі компоненти.

Розглянемо елементи інтерфейсу. У головному меню зібрано основні пункти керування (кнопки) для редагування й запуску програми. Тут є як уже знайомі з інших додатків групи кнопок «Завантажити» (Open), «Зберегти» (Save), «Редагувати» (Edit), так і специфічні, властиві тільки мовним середовищам пункти «Проект» (Project), «Запуск» (Run), які визначають режим компіляції й виконання програми.

Нижче від головного меню, ліворуч, розташована інструментальна панель швидких кнопок, на якій зібрано кнопки швидкого запуску для виконання найчастіше повторюваних операцій. Кнопки панелі не мають підписів, але їхні іконки позначають ту саму операцію в різних мовних середовищах.

Нижче від головного меню, праворуч, розташована палітра компонентів — це багатосторінковий діалоговий елемент, на кожній закладці якого є кілька відеокомпонентів, які можна переносити на форму. Назви закладок подано на палітрі (рис. 2.2).

Інтерфейс додатка визначається вибором тих чи інших компонентів. У його проектуванні діє принцип WYSIWYG (What You See Is What You Get) — «що бачите, те й одержите»).

Для того щоб помістити компонент на форму, необхідно перейти на потрібну сторінку, клацнути мишею на обраному компоненті в палітрі, а потім — у потрібному місці форми. Наприклад, нам необхідно розмістити на формі три кнопки керування. Для цього із закладки Standard ми, переміщуючи іконку, переносимо на форму три компоненти, при цьому візуальне середовище саме пронумерує їх: Button1, Button2 і Button3.

Після перенесення компонента на форму можна змінити місце його розташування й розміри. За замовчуванням компоненти вирівнюються на формі по лініях сітки, крок сітки дорівнює восьми пікселям, і сітку видно на етапі проектування. Можливе припасування місця розташування компонента з точністю до пікселя за допомогою клавіш зі стрілками: якщо натиснуто клавішу Ctrl, відбувається зсув компонента, а якщо натиснуто клавішу Shift — змінюються його розміри. Для деяких компонентів передбачене задавання місця їх розташування на передньому

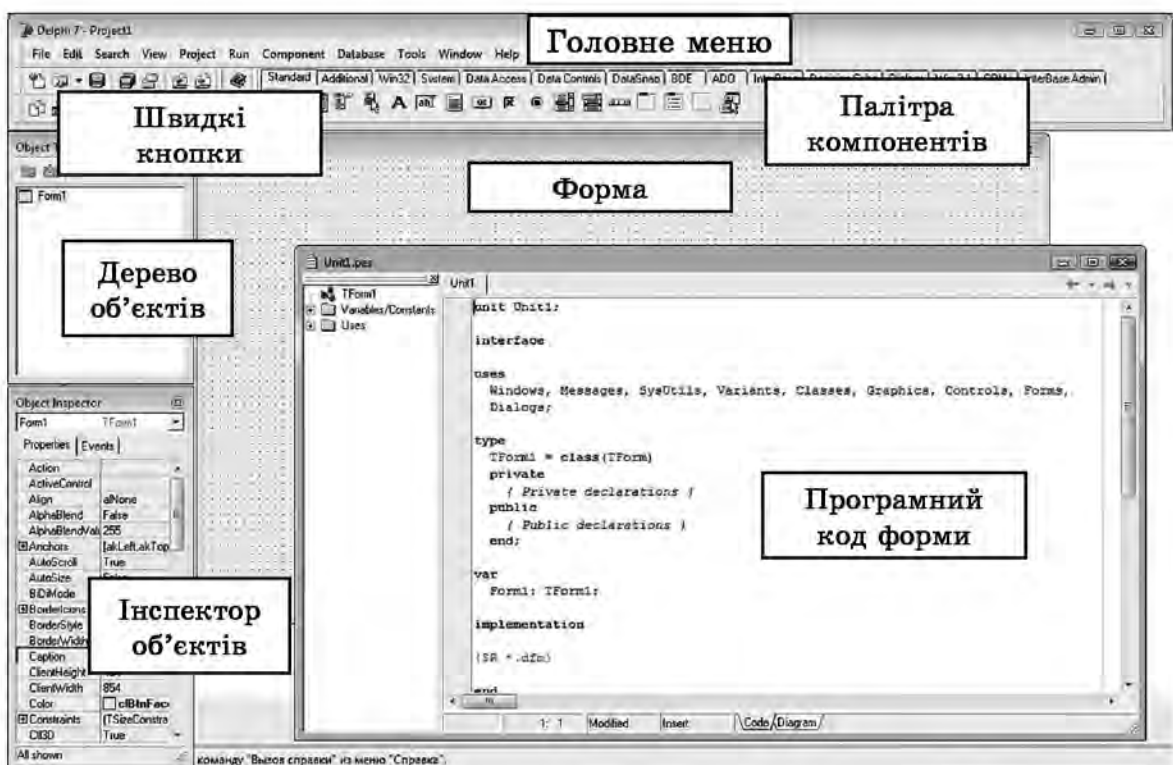


Рис. 2.1. Візуальне середовище програмування Delphi 7



Рис. 2.2. Палітра компонентів Delphi

або задньому плані. Вирівнювати розміщені компоненти можна або вручну, або виділивши групу компонентів і скориставшись командою Edit → Align або командою View → Alignment Palette (рис. 2.3).

Нижче від панелі швидких кнопок відкривається дерево об'єктів (Object TreeView) — вікно з умовним зображенням усіх компонентів, поміщених програмістом у проект. Це не просто перелік — якщо на формі є так звані об'єкти-контейнери (тобто компоненти, які містять у собі інші компоненти: панелі, сторінки, закладки тощо), то в цьому вікні лініями зображуються рівні їх підпорядкування (рис. 2.4).



Рис. 2.3. Способи вирівнювання компонентів

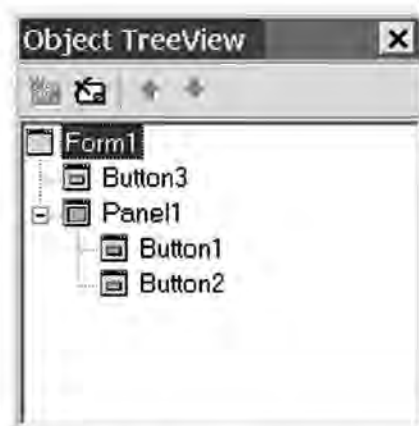


Рис. 2.4. Дерево об'єктів Object TreeView

Це важливо, наприклад, у тому випадку, якщо ви робите невидимим який-небудь об'єкт-контейнер — тоді всі підпорядковані компоненти також стають невидимими. Отже, ви винесли на форму кнопку. Подивившись у вікно дерева об'єктів, легко визначити, чи належить вона формі або якомусь об'єкту-контейнеру. Крім того, використовуючи дерево об'єктів, дуже зручно переносити компоненти з одного контейнера в інший за допомогою миші.

Основне місце на екрані займає вікно форми — це редаговане зображення майбутнього інтерфейсу програми. Як ви в процесі проектування розташуєте відеокомпоненти на формі, так у більшості випадків вони й будуть розташовані в запущеній на виконання програмі.

Під вікном форми є вікно редактора коду, у якому міститься програмний код виконуваного завдання. Програміст пише цей код самостійно. Візуальне проектування лише додає рядки опису в програму в міру виставлення компонентів на форму. Але середовищу Delphi, безперечно, невідомо, які дії ви хотіли виконати натисканням тієї чи іншої кнопки, винесеної на форму, чого ви хотіли від інших компонентів.

Кожен перенесений на форму компонент має багато важливих і корисних властивостей, значення яких установлені за замовчуванням,

але можуть бути настроєні програмістом для конкретної програми. Список доступних властивостей компонентів наведено у вікні інспектора об'єктів (Object Inspector), що складається з двох закладок — Properties і Events. У першій можна настроювати властивості компонентів, у другій — писати процедури обробників подій (рис. 2.5).



Рис. 2.5. Вікно Object Inspector

Слід пам'ятати, що деякі властивості не можна змінити на етапі дизайну. Вони доступні тільки під час виконання програми, і їх можна змінювати програмним шляхом. Властивості ж, перелічені в інспекторі об'єктів, дозволено змінювати як на етапі проектування, так і під час роботи додатка.

➡ Програмна розробка й файли, які входять до її складу

У середовищі Delphi розроблюваний проект являє собою набір файлів, з яких створюється додаток. У будь-який проект входять щонайменше шість файлів:

- project1.dpr — головний файл проекту, його формує система під час створення нового додатка;
- unit1.pas — перший модуль (unit) програми, що автоматично з'являється на початку роботи;
- unit1.dfm — файл опису форми, він використовується для збереження інформації про перелік відеокомпонентів і їх розташування на формі;

- `project1.res` — файл ресурсів, у ньому зберігаються іконки, растрові зображення, курсори;
- `project1.dof` — текстовий файл опцій для збереження установок, пов'язаних із цим проектом (наприклад директив компілятора);
- `project1.cfg` — файл конфігурації, він містить інформацію про стан середовища.

Крім перелічених, до проекту можуть належати файли з картинками, відеофрагментами, звуками, файли довідкової системи тощо. Якщо зберегти проект під іншим ім'ям, то, крім файла проекту, змінять назву й файли з розширенням `res`, `dof` і `cfg`. Якщо змінити ім'я файла модуля (`.pas`), то зміниться й ім'я файла опису форми (`.dfm`). Хорошим стилем програмування вважається використання імен, що мають змістове навантаження.

Після компіляції програми в робочій папці додатково з'являться файли з розширеннями:

- `dcu` — скомпільовані модулі;
- `exe` — виконуваний файл.

Головний файл проекту є текстовим файлом, що містить програмний код, записаний мовою Object Pascal. Текстовий файл підключає всі використовувані програмні модулі й містить оператори для запуску додатка. Під час створення нового додатка Delphi автоматично генерує файл проекту. Код файла проекту, що містить одну форму, наведено нижче.

```
-----  
program Project1;  
uses  
  Forms,  
  Unit1 in 'Unit1.pas' {Form1};  
{$R *.res}  
begin  
  Application.Initialize;  
  Application.CreateForm(TForm1, Form1);  
  Application.Run;  
end.  
-----
```

У розділі **uses** підключаються системний модуль `Forms` і модуль форми `Unit1`. Назва форми наводиться у фігурних дужках. Директива компілятора `{$R *.res}` підключає ресурси до результуючого `exe`-файла.

Тіло програми містить оператори, які готують додаток до роботи (ініціалізують), створюють форму й починають виконувати додаток.

У міру створення нових форм вміст `exe`-файла змінюється автоматично. Вручну цей файл коригується тільки в особливих випадках.

Під час створення додатка Delphi генерує порожню форму, текст модуля якої наведено нижче.

Приклад 1.

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs;
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
end.
```



Модуль починається із зарезервованого слова **unit**, після якого записується ім'я модуля. Воно збігається з ім'ям файлу, у якому збережений модуль. У загальному випадку структура модуля має такий вигляд:

- заголовок;
- секція інтерфейсних оголошень **interface**;
- секція реалізації **implementation**;
- секція ініціалізації **initialization**;
- секція завершення **finalization**;
- ознака закінчення тексту програми **end**.

➡ **Створення найпростішого проекту, його компіляція, збереження, виконання**

У момент старту Delphi автоматично створює новий проект. У разі завантаження іншого проекту (нового або наявного на диску) відкритий раніше проект закривається. Для створення нового проекту слід виконати команду **File → New → Application**, а для завантаження наявного проекту — команду **File → Open Project**. Рекомендується, створивши новий проект, відразу зберегти його в окремій папці.

Менеджер проекту Project Manager (рис. 2.6) призначений для керування проектами й складниками розроблюваного додатка. Його можна викликати командою View → Project Manager.

Менеджер проекту дозволяє працювати з групою проектів: можна переглядати, додавати й видаляти проекти і їхні складники. Вікно менеджера проекту розділене на дві частини. У верхній частині розташовані елементи керування (список, який розкривається, і кнопки), у нижній — перелік проектів і модулів із зазначенням шляхів до них. У разі додавання й видалення проектів та їхніх складників автоматично вносяться зміни у відповідні файли.

Установити параметри проекту можна у вікні Project Options (рис. 2.7), що відкривається за допомогою команди Project → Options.

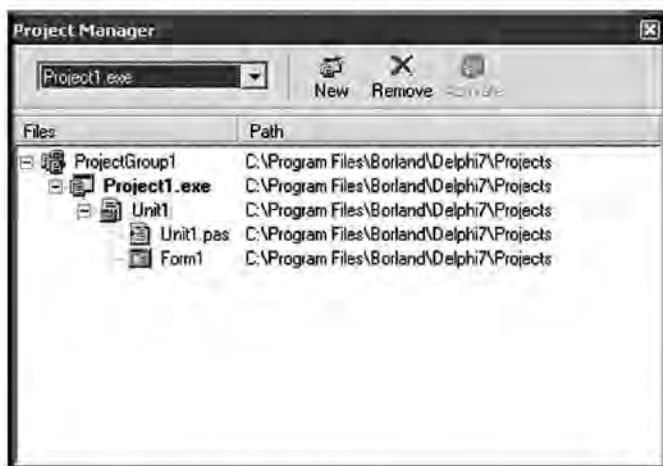


Рис. 2.6. Вікно Project Manager

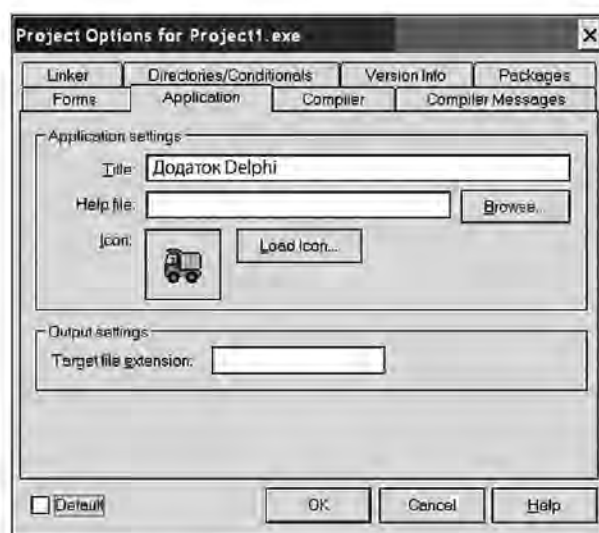


Рис. 2.7. Вікно налаштування параметрів проекту

На сторінці Forms можна призначити головну форму додатка й вибрати в списку Autocreate forms ті форми, які будуть створюватись одночасно з головною. Головна форма відкривається першою, і її закриття спричиняє закриття всього додатка. На сторінці Application задають назву й іконку, які будуть відображатись в середовищі Windows на панелі завдань, а також файл довідки, підключений до проекту. На сторінках Compiler і Linker встановлюють директиви компілятора й компоувальника (редактора зв'язків).

Компіляцію та збирання проекту можна виконати на будь-якій стадії його розроблення. Компіляцією називають одержання об'єктних модулів (dsu-файлів) з вихідних текстів програмних модулів (pas-файлів). Збирання передбачає одержання виконуваного файлу з об'єктних модулів. У середовищі Delphi компіляція і збирання проекту поєднані. Для виконання компіляції використовується команда меню Project → Compile <Ім'я проекту> або комбінація клавіш Ctrl+F9. При цьому компілюються всі вихідні модулі, вміст яких змінювався після останньої компіляції. У результаті для кожного програмного модуля створюється

файл із розширенням *dcu* (*Delphi Compiled Unit*). Далі середовище Delphi компілює головний файл проекту й компонує з *dcu*-модулів виконуваний файл (exe-файл), ім'я якого збігається з ім'ям проекту.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть основні файли, які входять до складу проекту.
2. Що означає хрестик (плюс), який стоїть у рамочці ліворуч від назви властивості об'єкта в Object Inspector?
3. Які файли (з якими розширеннями) з'являються в робочій папці після компіляції проекту?
4. Перелічіть основні розділи програмного коду модуля.
5. Яка інформація відображується у вікні Project Manager?
6. У якому вікні можна змінити іконку (картинку), що з'являється в лівому верхньому кутку вікна запущеного на виконання проекту?
7. Які обмеження існують на ім'я модуля, записане після ключового слова *unit*?
8. Що буде, якщо в ході перенесення з машини на машину загубиться файл із розширенням *dcu*?
9. Як виділити на формі прихований або візуально важко виокремлюваний відеокomпонент за допомогою вікна Tree View?
10. Як за допомогою вікна Tree View визначити, чи належить певна кнопка до контейнера-панелі?
11. Для чого призначене вікно Alignment?

2.2

Додавання кількох рядків коду до обробника події, їх аналіз. Поняття форми, елемента керування, події, обробника події. Редагування коду обробника події

➔ **Додавання кількох рядків коду до обробника події, їх аналіз**

Найпоширеніший спосіб поінформувати додаток про виникнення події — це клацнути мишею під час роботи на якомусь компоненті. Унаслідок цього в програмі виникає подія *OnClick*. Якщо ж подія не обробляється програмою, це ні до чого не приведе. Щоб змусити програму реагувати на клацання, необхідно написати мовою Object Pascal фрагмент програми, який називається обробником події. При цьому створений фрагмент має являти собою послідовність команд, у яких програміст указує, що саме повинна зробити програма у відповідь на клацання мишею.

Щоб змусити Delphi самостійно зробити заготовку для процедури обробника події *OnClick*, необхідно в момент проектування додатка двічі клацнути на вставленому компоненті. Візьмемо для прикладу кнопку

Button1 (компонент класу TButton). У відповідь на подвійне клацання на цьому компоненті Delphi активізує вікно коду, у якому можна побачити такий фрагмент:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
-----
```

Слово **procedure** сповіщає компілятор про початок методу класу, реалізованого у вигляді процедури. За ним іде складене ім'я, що містить у собі ім'я класу TForm1 і власне ім'я методу Button1Click. При цьому у верхній частині модуля, всередині опису класу (розділ **interface**), з'явиться рядок:

```
procedure Button1Click(Sender: TObject);
```

Виконані вищезазначені дії привели до виклику обробника OnClick — цей обробник найчастіше використовується за замовчуванням (він викликається для більшості компонентів). Існують також компоненти, для яких за замовчуванням визначені інші обробники. Найчастіше для форми використовують обробник OnCreate. Усередині цього програмного фрагмента зручно задавати які-небудь початкові значення для змінних, ініціалізувати параметри, відкривати файли й задавати розміри компонентів, включно із самою формою. Компонент Edit, за допомогою якого здійснюється введення текстової інформації, за замовчуванням настроений на обробник OnChange. Цей перелік можна продовжити. На рис. 2.8 показано деякі обробники, що використовуються за замовчуванням.

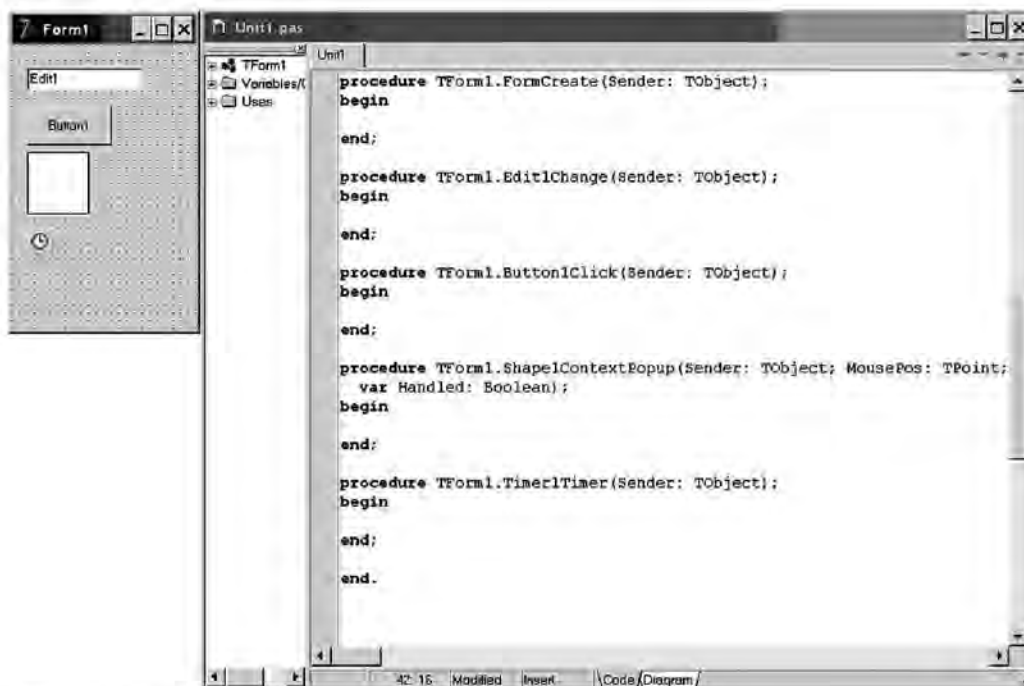


Рис. 2.8. Обробники події за замовчуванням

➔ Поняття форми, елемента керування, події, обробника події

Усі виконувані програми у Windows традиційно поділяють на додатки і служби. Термін «додаток» найчастіше стосується великої програми, що має інтерфейс користувача й широкий вибір функцій. На відміну від додатка, служба — це програма, яка дає дуже вузький діапазон функціональних можливостей, куди зазвичай не входять додаткові функції, що мають самостійне значення. Служби здебільшого виконуються у фоновому режимі й мають невеликий користувацький інтерфейс або ж не мають його зовсім.

Коли програміст створює в операційному середовищі Windows нову програму, кажуть, що з'являється новий додаток для Windows (*Windows application*).

Інтерфейс програми мовою Delphi починається з форми. Безумовно, існують консольні додатки, які не мають форм, а виконуються в окремому вікні без графіки з введенням-виведенням білими символами на чорному тлі, але така форма програмування не має стосунку до візуального програмування задач.

У ході створення в Delphi нового додатка (New → Application) перше, що ви побачите на екрані, — це велика заготовка форми з написом Form1 у заголовку.

Форма являє собою реалізацію вікна операційної системи Windows об'єктозорованою мовою програмування (у цьому випадку — Delphi). Властивості й події форми, як і інших компонентів, можна задавати і змінювати за допомогою інспектора об'єктів. Графічний додаток може складатися з кількох форм, але тільки одна з них вважається головною (головна форма відображується на екрані під час запуску додатка; закриття головної форми приводить до завершення роботи додатка). Якщо в ході розміщення відеокомпонентів на формі постійно перемикає вікна, стежачи за змінами програмного коду, то можна побачити, як текст програми доповнюється новими рядками описів. Робиться це автоматично, тому вікно програмного коду зазвичай сховане на задньому плані.

Розглянемо на прикладі, як користуватися вікном Object Inspector.

Припустімо, у ході створення нового додатка на екрані з'явилася заготовка форми (рис. 2.9). Потрібно перефарбувати її в чорний колір і вивести в заголовку форми слово «Приклад».

Клацанням мишею викликаємо вікно Object Inspector і обираємо властивість Color. Поряд із назвою властивості розташоване поле введення для редагування значення

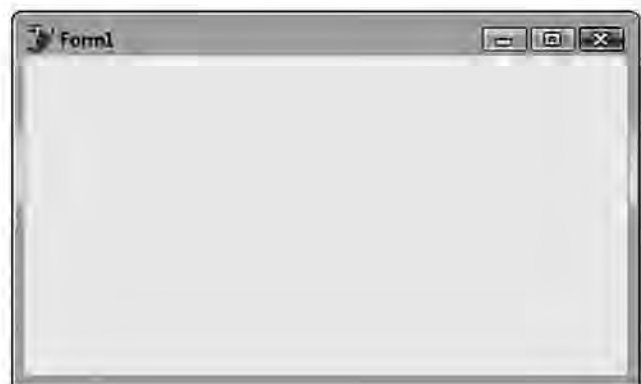


Рис. 2.9. Вихідна форма

параметра; для введення властивості Color передбачено висхідне меню (рис. 2.10).

Отже, *властивість (property)* — це елемент компонента або класу, відповідальний за якусь характеристику компонента або класу (колір, напис, розмір шрифту тощо). Програмне записування або зчитування інформації про властивість компонента приводять до виклику пов'язаних із ним процедур і функцій (методів), яких у явному вигляді ми не викликали. Наприклад, у разі змінення властивості Width (ширина) для певного візуального компонента автоматично цей компонент перерисовується (якщо цей компонент є формою, то перерисовується вся форма). Багато які властивості компонентів відображаються в інспекторі об'єктів і можуть змінюватися на етапі проектування. Доступ до деяких властивостей можливий лише під час виконання програми. Серед останніх можуть бути властивості тільки для читання, безпосередні зміни яких у програмі заборонені.

Оберемо у висхідному меню поля введення необхідне значення (за умовою наведеної вище задачі це чорний колір clBackground), і форма автоматично стане чорною. Аналогічно оберемо властивість Caption і введемо в редагований рядок значення слово «Приклад» замість рядка Form1 (рис. 2.11, а). У результаті після запуску ми одержимо форму, яка нам необхідна за умовою задачі (рис. 2.11, б).

Змінити значення більшої частини властивостей можна не тільки в процесі проектування, але й на етапі виконання програми. Робиться це оператором присвоювання, однак новому додатку необхідно дати сигнал, коли слід починати виконувати конкретні дії. Такий сигнал називається *подією*.

У Delphi реалізовано подієве керування програмою. Відразу після запуску програма зупиняється, чекаючи на який-небудь керівний

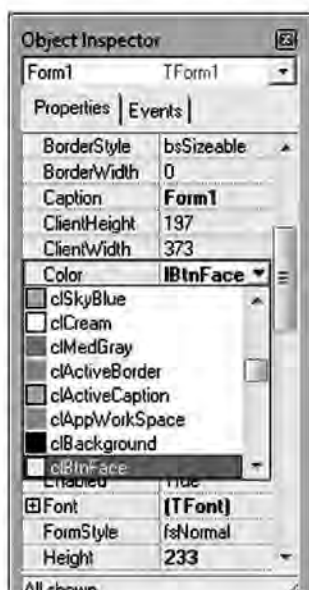


Рис. 2.10. Настроювання кольору форми через Object Inspector



а



б

Рис. 2.11. Настроювання заголовка (а) і готова форма (б)

вплив. Таким впливом може бути натискання клавіші на клавіатурі, клацання мишею, завершення роботи таймера тощо. Кожен компонент на екрані (форма, кнопка, рисунок та ін.) настроєний на реакцію тільки на чітко визначені події. Наприклад, рисунок не призначений для введення тексту, тому він ніяк не реагує на натискання кнопок на клавіатурі. Для кожної передбаченої події у відеокомпонента існує метод, який описує, що слід зробити, коли настає подія.

Отже, *подія (event)* — це зовнішній керівний вплив на компонент, що дозволяє пов'язувати його з обробником події — процедурою, передбаченою для кожної стандартної ситуації (наприклад, у разі одержання компонентом фокуса або клацання на ньому мишею). Імена подій починаються з префікса «On». Події компонента й пов'язані з ними обробники зазначаються на закладці Events інспектора об'єктів.

Вище в описі властивостей і подій були використані терміни «метод» і «компонент». Дамо їм визначення. *Метод (method)* — це процедура або функція, включена в опис компонента (класу). Сукупність методів визначає «дії», що їх можуть виконувати об'єкти цього класу. *Компонент (component)*, або *відеокомпонент*, — це елемент керування відеододатком. Класи компонентів походять від базового компонента-предка — класу TComponent. Майже всі компоненти, за винятком тих, які «вбудовані» в інші компоненти (наприклад ToolButton або TabSheet), розташовуються на палітрі компонентів і можуть бути перенесені на форму під час проектування програми.

У назві компонента зазвичай опускають префікс «T», що належить до імені відповідного класу: так, компонент типу TEdit називається просто Edit. Серед компонентів виділяють візуальні (нащадки класу TControl), які відображаються на формі під час виконання програми, а серед візуальних — віконні (нащадки класу TWinControl), які можуть одержувати фокус введення та обробляти виниклі події (унаслідок натискання клавіші і клацання мишею).

Розглянемо найпростіше програмування методів для оброблення подій. Дещо ускладнимо наведений вище приклад.

Від минулої задачі в нас лишилася порожня форма чорного кольору з написом «Приклад» у заголовку. Потрібно змінити програму так, щоб під час натискання клавіші миші (у цей момент маніпулятор має перебувати над формою) в заголовку з'являвся напис «Натиснуто», а після відпускання клавіші — «Не натиснуто».

Для програмування цієї задачі переходимо на другу закладку інспектора об'єктів (рис. 2.12) і знаходимо рядок OnMouseDown — це метод, що виконується після натискання клавіші миші.

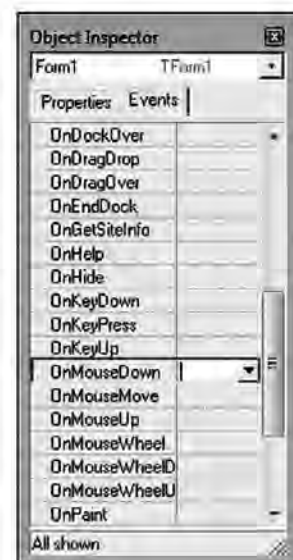


Рис. 2.12. Вибір методу в інспекторі об'єктів

За допомогою подвійного клацання мишею на правому полі в рядку OnMouseDown відкриваємо вікно редагування програмного коду. При цьому рядки програми з назвою процедури та її операторними дужками **begin ...end**; вписуються в текст програми автоматично. Залишається тільки дописати рядок програми, який присвоює, згідно із завданням, нове значення властивості Caption (заголовок) компонента Form1. Вписуємо цей рядок:

```
-----
procedure TForm1.FormMouseDown(Sender: TObject;
Button: TMouseButton; Shift: TShiftState;
X, Y: Integer);
begin
    form1.Caption:='Натиснуто';
end;
```

Аналогічно програмуємо метод OnMouseUp. Пам'ятайте: унаслідок відпускання миші напис у нас має змінитися! Обираємо відповідний рядок на закладці Events інспектора об'єктів і вписуємо дуже схожий рядок у програму:

```
-----
procedure TForm1.FormMouseUp(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
begin
    form1.Caption:='Не натиснуто';
end;
```

Вітаємо! Ви створили перший, хай і дуже простий додаток для Windows. Запустити програму можна натисканням клавіші F9. Наступний крок вам належить зробити самостійно. Іще раз ускладнимо наведений вище приклад з метою закріпити матеріал.



Приклад 2. Для програми, одержаної в попередній задачі, потрібно написати метод, який би виводив у заголовок форми напис «Миша рухається» під час переміщення миші над формою.

Аналізуючи результат, зверніть увагу на два цікаві моменти:

- * якщо під час запуску курсор розташований не над формою, то ми можемо рухати мишу як завгодно, а напис «Миша рухається» не з'явиться (це легко пояснити: написаний нами метод стосується тільки форми, тому подія OnMouseMove генерується, коли курсор заходить на форму);
- * напис «Не натиснуто» тепер у заголовку не з'являється (формально працюють обидва методи, але оскільки відпускання клавіші миші також вважається її рухом, напис «Миша рухається» встигає з'явитися замість напису «Не натиснуто» раніше, ніж ми встигнемо сформувати зоровий образ).



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Що таке обробник події?
2. Перелічіть назви обробників події за замовчуванням для компонентів Form, Edit, Button, Shape і Timer.
3. Що собою являє додаток? Як створити новий додаток у Delphi?
4. Що таке форма? Назвіть основні її характеристики.
5. Дайте опис понять «властивість», «подія» і «метод».
6. Що таке компонент?

Практична робота № 1

Створення найпростішої Windows-програми



Мета роботи: набути практичних навичок створення простих додатків.

Завдання. Написати програму-підказку для навчання англійської мови. Для цього на форму виводиться комплект картинок; унаслідок клацання мишею на кожній із них у нижній частині форми виводиться підпис — назва зображеного на картинці предмета англійською мовою (рис. 2.13).

Хід роботи


1. Запустимо середовище Delphi та створимо новий додаток.
2. На заготовку форми, що з'являється на екрані після запуску Delphi (або внаслідок вибору через основне меню File → New → Application), помістимо 12 об'єктів Image. Відеокomпонент Image, призначений для виведення на форму малюнків у форматах BMP, JPG, EMF, WMF, ICO, розміщується на другій сторінці списку відеокomпонентів і має іконку . Виведення 12 малюнків (Image) однакових розмірів — доволі трудомістка операція. Простіше вивести один Image, скопіювати його натисканням



Рис. 2.13. Програма, що виводить англійські назви предметів на картинках

комбінації клавіш **Ctrl + C**, після чого вивести на форму ще 11 малюнків натисканням клавіш **Ctrl + V**. При цьому середовище Delphi саме нумерує відеоконпоненти **Image**, **Image3** і т. д. аж до **Image12**. Тепер по черзі клацаємо на кожному компоненті **Image** і налаштуємо програму на виведення відповідної картинки. Для цього обираємо для компонента **Image** в **Object Inspector** рядок **Picture**. Клацанням мишею на кнопці з трьома крапками напроти назви властивості **Picture** відкриваємо вікно завантаження файлу (рис. 2.14).

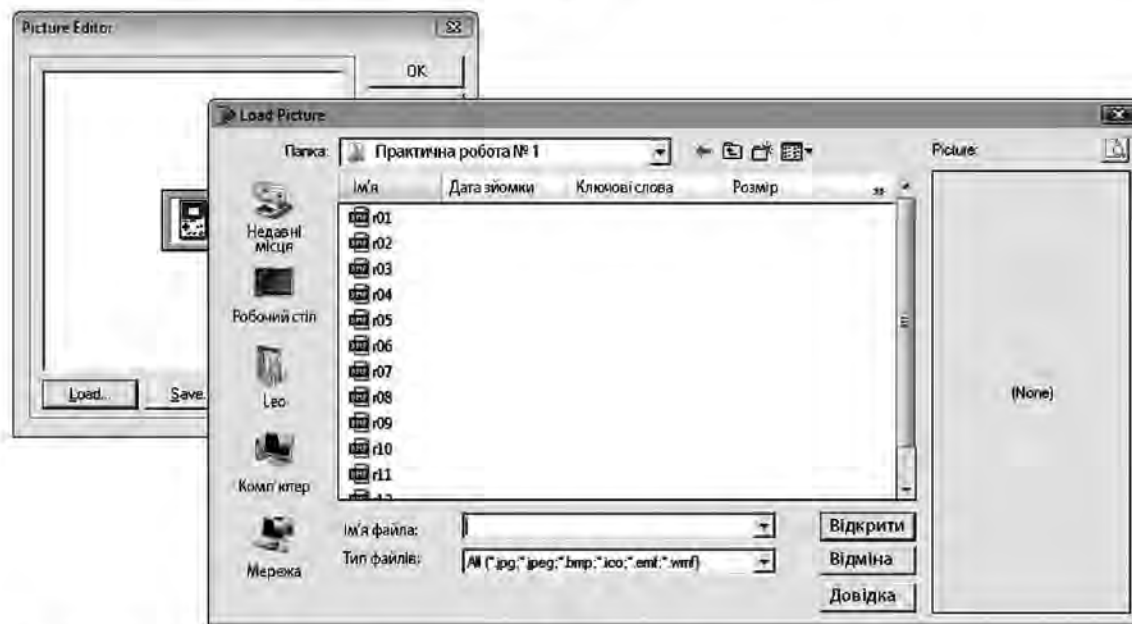


Рис. 2.14. Завантаження малюнка з файлу в компонент **Image** клацанням на кнопці **Load**

3. Виведення малюнка в об'єкти **Image** доведеться налаштувати 12 разів — для кожного об'єкта є свій діалог на вибір малюнка. Завдання можна полегшити, якщо всі файли картинок зібрати в одній папці, після чого зберегти в ній створюваний проект. У цьому випадку в діалозі не доведеться шукати малюнки, переглядаючи різні папки. Якщо розмір малюнка не збігається з розміром контурів компонента **Image**, рисунок можна розтягти, встановивши у властивості **Stretch** (розтягнення) значення **true**.
4. У нижній частині форми помістимо компонент **Label** (іконка **A** на першій закладці **Standard**). За замовчуванням у компонента **Label** розмір шрифту нехай дорівнює 8, букви світлого написання. Через **Object Inspector** настроїмо наш рядок виведення тексту на більші й напівжирні букви: властивість **Font** → **Size** встановимо 12, а властивості **Font** → **Style** → **fsBold** присвоїмо значення **true**.
5. Тепер напишемо текст програми для обробників подій «натиснута клавіша миші під час перебування курсора над об'єктом». Почнемо з першого малюнка — **Image1**. На нього виведено зображення сиру («сир» англійською «cheese»). Вибираємо в **Object Inspector** для компонента **Image1** подію **OnMouseDown** і клацаємо мишею

на вільному полі. Автоматично створюється оголошення методу оброблення події, між **begin** і **end** вписуємо всього один рядок програмного коду:

```
-----  
procedure TForm1.Image1MouseDown(Sender: TObject;  
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);  
begin  
    Label1.Caption:='Cheese';  
end;  
-----
```

6. Очевидно, що цей рядок можна скопіювати і вставити в усі обробники натисканням клавіші миші на решті 11 малюнках, міняючи тільки слово Cheese на інші англійські слова. Для об'єктів на рис. 2.14 це будуть Calculator, Candle, Bouquet, Car, Pumpkin, Palm-tree, Plane, Jeep, Dices, Anchor, Hare (написи в операторах присвоювання виведеного рядка, природно, збігаються з послідовністю завантажених у компоненти Image картинок).
7. Збережемо отриманий додаток на диску й запустимо його на виконання.



Візуальне програмування суттєво полегшує написання складних програм, оскільки творці візуального середовища Delphi вже запрограмували цілий ряд операцій у відеокомпонентах. Проте й у візуальному середовищі слід уміти розробляти алгоритми програмованих задач. У цьому розділі ми познайомимося зі способами описування алгоритмів і правилами побудови їхніх структурних схем.

3.1

Способи опису алгоритмів. Складання й запис алгоритмів. Базові алгоритмічні конструкції

Нині поширилась і бурхливо розвивається наука, що вважається частиною дискретної математики,— теорія алгоритмів. Вона вивчає загальні властивості алгоритмів, порівнює час їх виконання, вводить числові оцінки їхньої складності. Цю теорію заклали Е. Борель (1912) і Г. Вейль (1921). В обчислювальній техніці найчастіше використовується теорія алгоритмів в інтерпретації А. Тьюрінга й Е. Поста (1936), їхні конструкції багато в чому передбачили ідеї, покладені в основу сучасних цифрових обчислювальних машин (машина Тьюрінга).

Прийнято вважати, що існують чотири способи опису алгоритмів:

- *словесний* (мама каже синові: «Віднеси взуття в ремонт, по дорозі назад купи хліба й подивись, чи немає листів у поштовій скриньці»,— цілком однозначний алгоритм);
- *формульний* (так, запис формул $a=1$; $b=2$; $c=-8$;

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$
 — також цілком однозначний алгоритм, хоча в ньому не застосовано жодного слова);
- *за допомогою алгоритмічних схем* (так званих структурних схем алгоритмів, які буде розглянуто нижче);
- *алгоритмічними мовами* (багато хто називає їх мовами програмування, хоча, нагадаємо, мова Pascal, що лежить в основі Delphi, запатентована не для програмування, а для опису алгоритмів — мовою програмування зі своїм транслятором вона стала пізніше).

Найпоширенішою формою запису алгоритмів є структурні схеми. Умовні позначки у вигляді прямокутників, ромбів та інших фігур на таких схемах показують елементарні кроки обчислювального процесу — «обчислення», «логічне розгалуження», «початок/кінець», «ручна операція» та ін. Стрілки ж відображують напрямок ходу обчислень.


Перш ніж описувати елементи алгоритмічних структур, зупинімося на загальних правилах їх використання:



- усі елементи структурних схем алгоритмів відповідають міжнародним стандартам, тому слід застосовувати наявні значки, а не придумувати власні зображення команд;
- елементи розроблено стосовно мови Algol-60, у зв'язку з чим деякі алгоритмічні конструкції зображуються досить складно (наприклад, розгалуження в різних напрямках — **CASE**);
- серед значків структурних схем чимало архаїчних — «запис на перфострічку», «виведення на перфокарти» тощо, тому доцільно описувати обчислювальний процес у загальному вигляді, не заглиблюючись у «машинні» подробиці;
- запис усередині значків слід виконувати математичною мовою. Використання операторів Delphi або C призводить до абсурдного висновку, що краще читати програму, а не ваш алгоритм;
- усі елементи алгоритму мають бути пронумеровані.

Умовні позначки елементів структурних схем алгоритмів наведено в табл. 3.1.

Таблиця 3.1

Елементи структурних схем алгоритмів
на їхнє призначення

Елемент	Призначення елемента
	Обчислювальний процес. Фактично це один або кілька нероздільних операторів присвоювання, що здійснюють обчислювальні перетворення
	Операції введення-виведення без конкретизації типу периферійних пристроїв
	Розгалуження дво- або багатоальтернативне
	Початок і кінець алгоритму. Формально може і не відповідати жодному з операторів програми
	Запис на «магнітний барабан» (тобто вінчестер). Елемент потрібен для тих мов, де запис на диск — проблема
	Міжсторінкове перенесення
	Перенесення в межах сторінки

Елемент	Призначення елемента
	Процедура. «Наперед визначений процес»
	«Модифікація». Зазвичай використовується для запису параметричного циклу

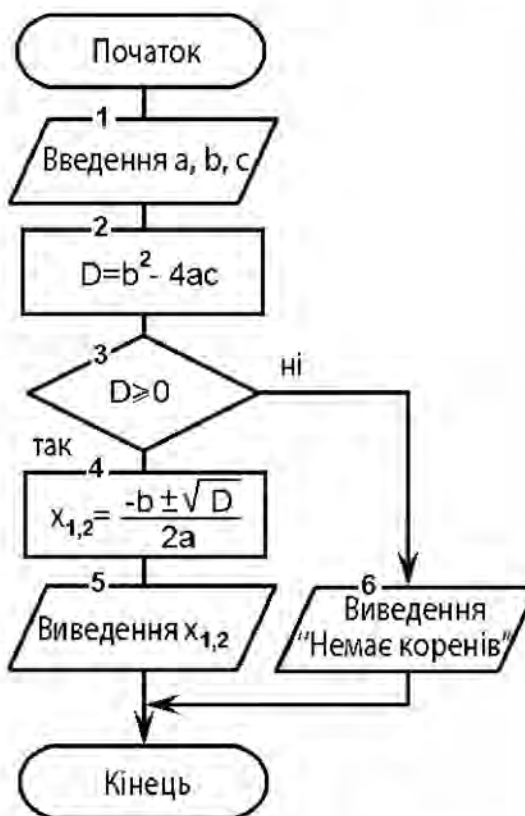


Рис. 3.1. Структурна схема алгоритму знаходження коренів квадратного рівняння

Як приклад на рис. 3.1 наведено структурну схему алгоритму знаходження коренів квадратного рівняння.

Зазначимо правила побудови структурних схем алгоритмів. Елементи таких схем мають вертикальну структуру. Передача керування від одного елемента до іншого, спрямована згори вниз або зліва направо, вважається природною і може не позначатися стрілками. Передача керування в інших напрямках, ламані лінії зв'язку або злиття зв'язків обов'язково позначаються стрілками.

Зверніть увагу: одна й та сама конфігурація компонентів структурних схем може відповідати різним алгоритмам (у графічному задаванні алгоритмів важливе не тільки геометричне зображення елементів, але й записаний у них текст).

Запис алгоритмів за допомогою динамічних змінних вимагає великої праці, а написання програми стековою мовою Forth — іще більшої.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Що вивчає теорія алгоритмів? Хто заклав основи цієї науки?
2. Перелічіть два способи опису алгоритмів.
3. Які елементи використовуються в побудові структурних схем?
4. У яких випадках лінії на структурних схемах обов'язково позначаються стрілками?

3.2

Структура й складові елементи програм, записаних об'єктозорієнтованою мовою програмування

Останнім часом набула поширення мова UML (*Unified Modding Language*) — загальноцільова мова візуального моделювання, розроблена для візуалізації, проектування й документування компонентів програмного забезпечення, а також бізнес-процесів та інших систем, що не належать до програмного забезпечення. Ця мова ввбрала в себе найбільш прогресивні методи програмної інженерії, що з успіхом використовувалися протягом останніх років у моделюванні великих і складних систем. Мова UML дуже різнобічна, причому включений у неї механізм так званих діаграм діяльності (activity diagram) дозволяє ефективно описувати алгоритми. Із цієї причини дедалі частіше в літературі замість структурних схем алгоритмів наводять діаграми діяльності з мови UML.

У класичних діаграмах діяльності наявні позначення станів і переходів, репрезентація дій, однак відсутня сигнатура подій, які ініціювали ці дії. (Для репрезентації даних подій в UML існує інший тип діаграм.)

Кожен стан на діаграмі діяльності відповідає виконанню певної елементарної операції, а перехід у наступний стан здійснюється тільки після завершення операції в попередньому стані. Зовнішній вигляд простої діаграми діяльності подано на рис. 3.2.

Звернемо увагу на відмінності діаграм діяльності від традиційних структурних схем. Вони полягають у такому:

- усі стани на діаграмах діяльності зображуються прямокутниками зі скругленими кутами;
- нумерації цих прямокутників немає, однак внутрішній запис має бути унікальним у межах даного алгоритму (інакше необхідний прямокутник складно знайти);
- переходи зображуються стрілками, які прорисовуються завжди, навіть якщо перехід іде в «природному» напрямку згори вниз або зліва направо;
- оператор розгалуження внутрішніх написів не має: умови розгалуження записані над його вихідними стрілками (діями);
- початок і кінець алгоритму зображуються чорними кружками (закінчення алгоритму — з білою облямівкою).

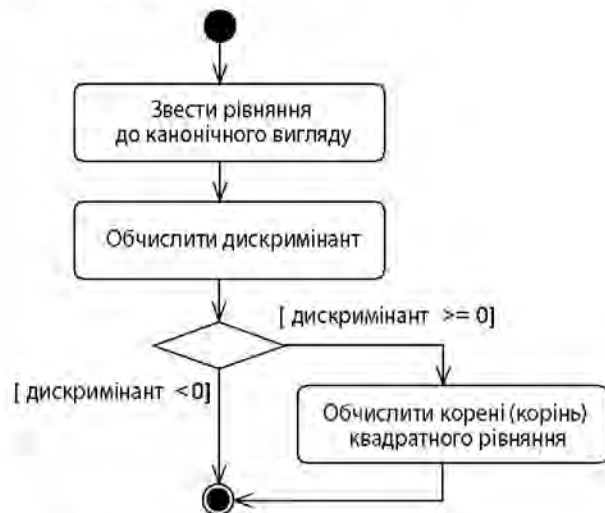


Рис. 3.2. Зображення алгоритму знаходження коренів квадратного рівняння у вигляді діаграми діяльності

Дія може бути записана природною мовою, певним псевдокодом або мовою програмування. Ніяких обмежень щодо записування дій не накладено. Рекомендується як ім'я простої дії використовувати дієслово з пояснювальними словами. Якщо ж дія може бути подана в певному формальному вигляді, то доцільно записати її тією мовою програмування, якою передбачається реалізувати конкретний проект.

На відміну від звичайних структурних схем алгоритмів, діаграми діяльності дозволяють описувати паралельні обчислювальні процеси, як це показано на рис. 3.3.

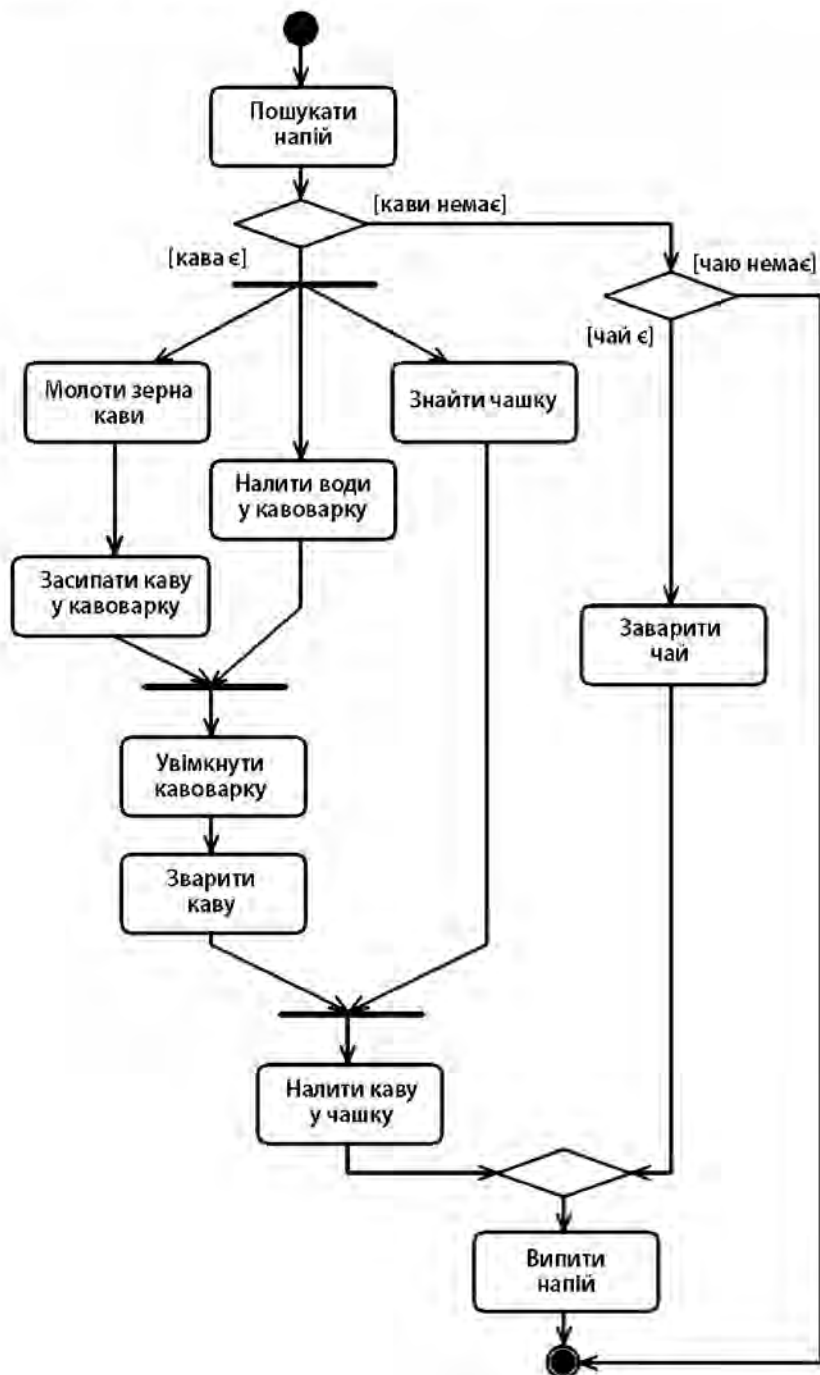


Рис. 3.3. Діаграма діяльності алгоритму, який має паралельні вітки

Очевидно, що в системах із багатоядерними процесорами й розпаралелюванням завдань діаграми діяльності дають більше можливостей записувати алгоритми, ніж звичайні структурні схеми.

Практична робота № 2

Складання й опис алгоритмів

Мета роботи: навчитися складати структурні схеми алгоритмів і читати за ними алгоритми.

Завдання. На рис. 3.4 наведено порожню структурну схему алгоритму. Внесіть у її компоненти відповідні текстові рядки, щоб ваш алгоритм міг розв'язувати одну з таких задач:

- знайти добуток 15 введених чисел;
- знайти суму 20 введених чисел;
- людина має 100 гривень; знайти суму грошей, яка залишиться в неї після оплати чотирьох товарів;
- ввести 10 чисел і вивести суму квадратів введених чисел.

Хід роботи

1. Подібні задачі простіше розв'язати, якщо самостійно скласти алгоритм їх розв'язування, а потім зіставити його із заготовкою алгоритму. Отже, складаємо алгоритм розв'язування однієї з наведених вище задач і зіставляємо його із заготовкою на рис. 3.4.
2. Однією з основних цілей створення структурних схем алгоритмів є пояснення користувачеві, не знайомому з вашою програмою, принципів її функціонування, використаного методу розв'язування та призначення програмних змінних. Як правило, всі розв'язують «пряму задачу» — потрібно скласти алгоритм. У цьому прикладі ми пропонуємо спробувати свої сили в «оберненій задачі»: відновити за наявним алгоритмом умову розв'язуваної задачі.

Завдання. На рис. 3.5 подано структурну схему алгоритму. Визначити, якого роду задачі вона може розв'язувати.

Розв'язання. У трьох виконуваних блоках алгоритму («початок» можна не рахувати) вводяться два числа. Шляхом проміжного переприсвоювання

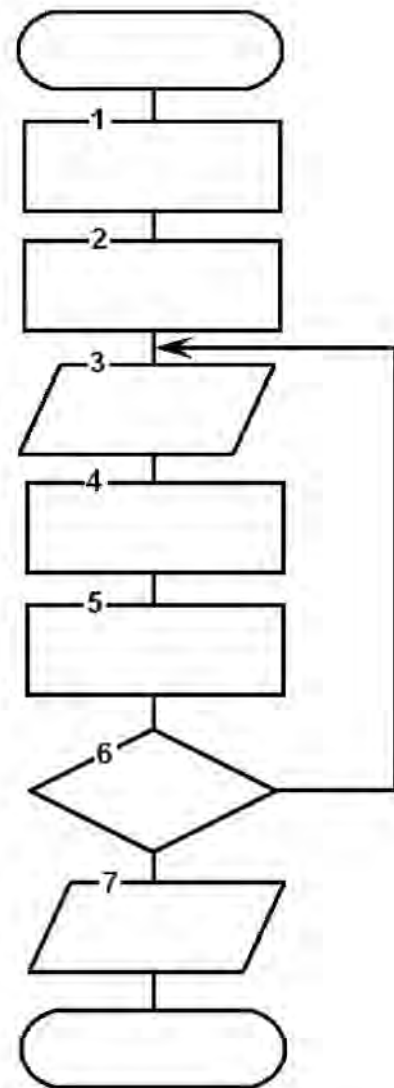


Рис. 3.4. Структурна схема алгоритму (заготовка для заповнення написами)

одне з них стає змінною M , а друге — X . Логічний оператор записує в змінну M значення X , тільки якщо $M \leq X$. У протилежному випадку присвоєння не відбувається. Інакше кажучи, після логічного оператора в змінній M міститься менше з двох введених значень. Далі йде виведення значення змінної M . На підставі аналізу можна сказати, що дана структурна схема алгоритму відповідає задачі пошуку мінімального з двох введених значень.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

За наведеними на рис. 3.6 структурними схемами алгоритмів відновити умови задач і записати їх у зошиті.

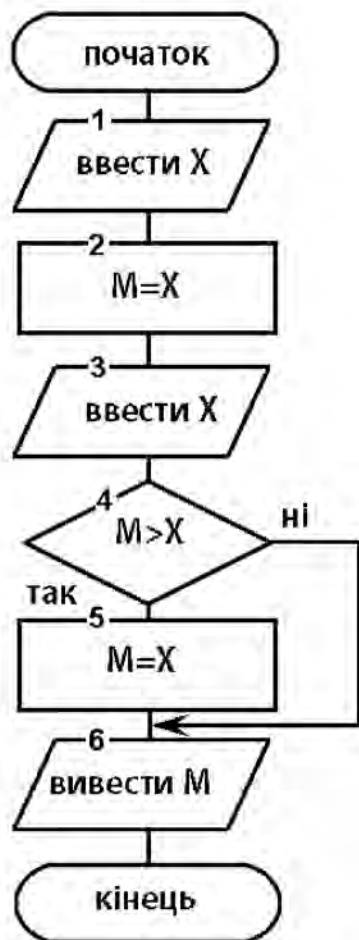


Рис. 3.5. Структурна схема алгоритму (до завдання)

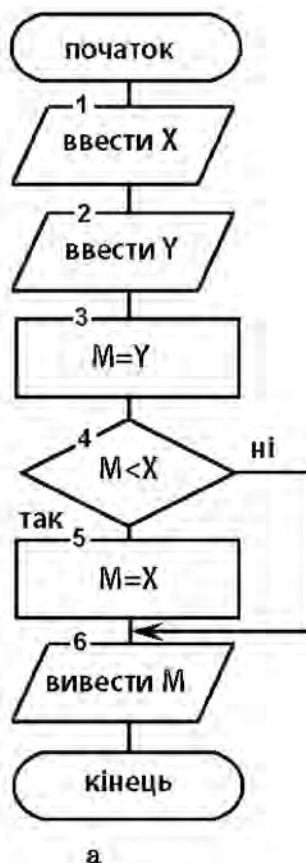
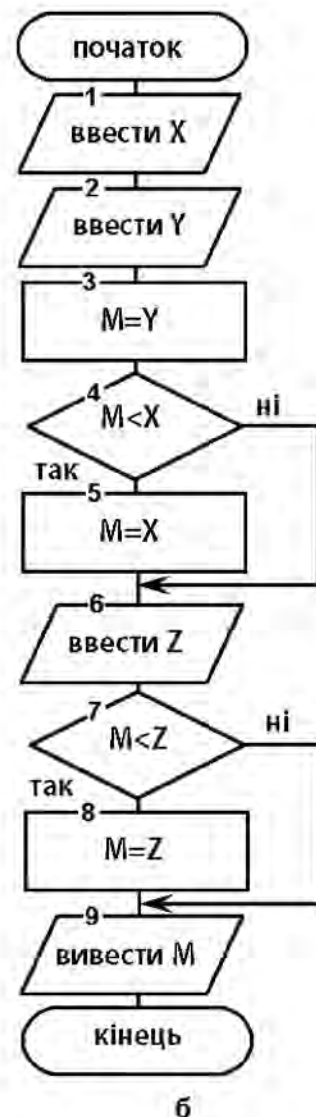


Рис. 3.6. Структурні схеми алгоритмів (для визначення умов задач)





Візуальне програмування — це насамперед об'єктне програмування з використанням форм і компонентів. А коли так, то хоч яким буде проект — простим чи складним, однаково кожен програміст у цьому середовищі муситиме налаштовувати компоненти й форми: змінювати їхні властивості, програмувати оброблення подій, використовувати вікна повідомлень. Знайомству саме з цими елементами візуального програмування й присвячений цей розділ.

4.1

Основні компоненти Windows-програми. Розроблення й застосування форм. Елементи керування та їхні атрибути

➔ **Основні компоненти Windows-програми. Розроблення й застосування форм**

Середовище розробки Delphi орієнтоване передусім на створення програм для Windows. При цьому особлива увага приділяється можливості візуально розробляти додатки за допомогою великого набору готових компонентів Delphi, що дозволяють уникнути ручного кодування. Компоненти Delphi охоплюють практично всі аспекти застосування сучасних інформаційних технологій.

Знайомство з основними компонентами Windows-програми розпочнемо з першої сторінки палітри компонентів Standard (рис. 4.1). На ній розміщені 16 об'єктів, які мають першорядне значення. Набір і порядок розташування компонентів на кожній сторінці є конфігурованими: можна додати до наявних компонентів нові, змінити їх кількість і послідовність.

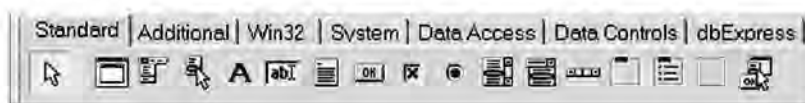


Рис. 4.1. Компоненти, розташовані на сторінці палітри Standard

Перелічимо компоненти Delphi з палітри Standard і наведемо деякі коментарі щодо їх застосування.



Frames — об'єкт-контейнер для виведення вмісту фреймів на форму. Натиснути його можна, тільки якщо ви заздалегідь створили через **Файл** → **New** один або кілька фреймів, тоді **Frames** запропонує вибрати список фреймів, які він може вивести на форму.

Ефектний вигляд має форма, коли залежно від ситуації на ній змінюється конфігурація кнопок. Для створення такої форми програміст має заготувати кілька фреймів, які під час виконання програми можуть бути видимими або невидимими.



MainMenu — компонент для внесення меню в програму. Після розташування MainMenu на формі він має вигляд іконки. Іконки цього типу називають невидимими компонентами, оскільки вони стають невидимими під час виконання програми. Створення меню складається з трьох кроків:

- 1) розташування MainMenu на формі;
- 2) виклик дизайнера меню;
- 3) визначення пунктів меню в дизайнері меню.



PopupMenu — висхідне меню, яке з'являється в результаті клацання правою кнопкою миші.



Label — компонент для відображення тексту на екрані. Ви можете змінити шрифт і колір позначки, двічі клацнувши на властивості Font в Object Inspector. Це легко зробити й під час виконання програми, написавши всього один рядок коду.



Edit — стандартний елемент керування Windows для введення програми. Його можна використати для відображення короткого фрагмента тексту, він дозволяє користувачеві вводити текст під час виконання програми.



Мемо — багаторядковий текстовий редактор. Використовується для введення користувачем і відображення багаторядкового тексту без функцій форматування.



Button — кнопка, що дозволяє здійснити які-небудь дії під час виконання програми. У Delphi все робиться дуже просто. Помістивши Button на форму, подвійним клацанням ви можете створити заготовку обробника події — у цьому випадку це натискання кнопки.





CheckBox — незалежний перемикач. Використовується його властивість Checked (позначено), що має значення true або false, які змінюються від клацання мишею. Відображується у вигляді рядка тексту з малим віконцем ліворуч. На питання в рядку компонента можна дати позитивну відповідь, поставивши позначку у віконці. Наприклад, якщо викликати вікно діалогу налаштувань компілятора (пункт меню Options → Project, сторінка Compiler), то можна побачити, що воно складається переважно з CheckBox.





RadioButton — залежний перемикач, що використовується для вибору тільки одного з кількох варіантів. Для цього компонент


об'єднується з одним або кількома такими самими компонентами в групу. Клацання мишею на компоненті приводить до його виділення і зняття виділення раніше обраного компонента.


 **ListBox** — список вибору, що містить список пропонованих варіантів (опцій) і дає можливість проконтролювати поточний вибір.


 **ComboBox** — список, що розкривається, багато в чому нагадує **ListBox**, за винятком того, що дозволяє вводити інформацію в маленькому полі введення зверху **ListBox**. Є кілька типів **ComboBox**. З них найпопулярніший випадаючий список (**drop-down combo box**). Це багаторядкове меню, у неробочому стані згорнуте до одного активного рядка. Унаслідок активації цей компонент розгортається до повного списку.

 **Scrollbar** — смуга прокручування. Вона являє собою вертикальну або горизонтальну смугу, яка керує візуальним поданням компонентів, що не вміщуються цілком у вікні програми.

 **GroupBox** — компонент-контейнер, що використовується з візуальною метою і для вказування **Windows**, яким є порядок переміщення по компонентах на формі у результаті натискання клавіші **Tab**. Наприклад, якщо кнопки виставлено на форму в довільному порядку і частина з них розміщена в контейнері **GroupBox**, то під час обходу табулятором послідовно буде опитано всі кнопки **GroupBox** незалежно від їхніх номерів.

 **RadioGroup** — група залежних перемикачів. Містить спеціальні властивості для обслуговування кількох пов'язаних між собою залежних перемикачів.

 **Panel** — панель, що, як і **GroupBox**, служить для об'єднання кількох компонентів. Містить внутрішній і зовнішній край, що дозволяє створювати ефекти втиснутості та опуклості.

 **ActionList** — список дій, що слугує для визначення реакції програми на дії користувача, пов'язані з вибором одного з групи однотипних елементів керування, як-от: опції меню, кнопки й т. д.

Навіть для створення нескладних додатків наведений перелік компонентів може виявитися недостатнім, тому розглянемо другу сторінку компонентів **Additional** (рис. 4.2).

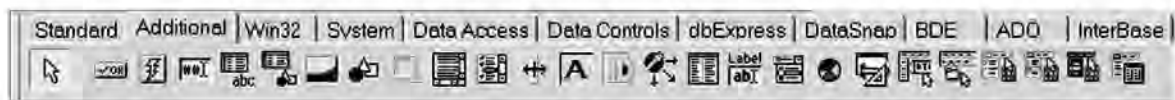


Рис. 4.2. Компоненти, розташовані на сторінці палітри **Additional**

Перелічимо найчастіше використовувані компоненти сторінки Additional із деякими коментарями щодо їх застосування.



BitBtn — командна кнопка. Відрізняється від стандартної кнопки **Button** можливістю відображення піктограми.



SpeedButton — піктографічна кнопка. Використовується зазвичай для швидкого доступу до опцій головного меню.



MaskEdit — аналог **Edit**, що забезпечує можливість форматованого введення. Формат визначається у властивості **EditMask**. У редакторі властивостей для **EditMask** є заготовки для введення-виведення дати, валюти та ін.



StringGrid — таблиця рядків. Цей компонент має потужні можливості для подання текстової інформації в табличному вигляді.



DrawGrid — таблиця зображень. Використовується для подання зображень у табличному вигляді.



Image — рисунок. Використовується для відтворення зображень, у тому числі піктограм і метафайлів.



Shape — фігура. За допомогою цього компонента можна вставити на форму правильну фігуру — прямокутник (квадрат), еліпс (коло).



Bevel — край. Слугує для виділення окремих частин форми тривимірними рамками та смугами.



ScrollBox — панель зі смугами прокручування. На відміну від компонента **Panel**, автоматично вставляє смуги прокручування, якщо розміщені на ній компоненти виходять за її межі.



CheckListBox — список множинного вибору. Відрізняється від стандартного компонента **ListBox** наявністю поряд із кожною опцією незалежного перемикача типу **CheckBox**, що дозволяє обрати відразу декілька опцій.



Splitter — межа. Цей компонент створює межу між двома видимими компонентами і дає користувачеві можливість переміщувати її, збільшуючи один компонент за рахунок іншого.



StaticText — статичний текст. Відрізняється від стандартного компонента **Label** наявністю власного **Windows**-вікна, що дозволяє обводити текст рамкою або виділяти його у вигляді «втиснутої» частини форми.



Chart — діаграма. Цей компонент полегшує створення спеціальних панелей для графічної репрезентації даних.

➔ Елементи керування та їхні атрибути

У ході розроблення програми у візуальних середовищах програміст працює не тільки з формами. Суттєвою підтримкою є візуальні компоненти. Добре підготовлений програміст міг би обійтись і без них, створюючи всі компоненти програми самостійно «з нуля». Але навряд чи це доцільно. В інтерфейсі Delphi заздалегідь заготовано кілька інструментальних закладок, які містять чимало дуже зручних візуальних компонентів.

Візуальні компоненти являють собою логічно завершений програмний код, що виконує операції з даними окремих модулів системи. Практично будь-який програмний код може бути поданий у вигляді компонента. Візуальний компонент — це заздалегідь заготований тип даних, багато в чому подібний до **integer**, **string**, **array**, хоч і набагато складніший. Переносючи компонент з інструментальної лінійки на форму, ми створюємо в програмі «екземпляр об'єкта», тобто змінну даного типу. Наприклад, у нас була порожня форма. Відкривши вікно програми, ми побачимо заготовку для написання коду:

```
-----
type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Перенесемо на форму кнопку (змінну типу TButton), і опис форми автоматично (без нашої участі) зміниться:

```
-----
type
  TForm1 = class(TForm)
    Button1: TButton;
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

На формі з'явилася змінна Button1 типу Tbutton (екземпляр об'єкта «кнопка»).

Візуальні компоненти мають п'ять основних характеристик, які наведено нижче.

Властивості (Properties) — параметри, що описують характерні ознаки об'єкта, наприклад: ширина компонента на формі (Width), висота компонента (Height), доступність (логічна властивість Enabled) та ін.


Набір властивостей для різних компонентів є різним. Зокрема, невидимі об'єкти не мають ні «ширини», ні «висоти».

Події (Events) — набір впливів на компонент, що викликають виконання ним певних програмних операцій. Ці впливи можуть бути як від зовнішніх пристроїв (наприклад, OnMouseDown — натискання клавіші миші в момент, коли покажчик миші перебуває над компонентом), так і програмними (наприклад, OnCreate — дії, які треба виконати в ході створення форми).

Методи (Methods) — набір процедур і функцій, які належить виконати компоненту у відповідь на певну подію. Програміст за допомогою вікна Object Inspector зазвичай записує код лише в потрібні процедури, залишаючи незаповненими непотрібні реакції на події. У складнішому варіанті можна програмувати «акціями» (перший рядок на закладці Events у вікні Object Inspector). У цьому випадку всі методи програмуються окремо й просто призначаються даному компоненту.

Ще дві характеристики — **Повідомлення (Messages)** і **Контекст (Context)** — визначають правила передавання керівної інформації від одного компонента до іншого та режим графічного відображення компонентів. У цьому навчальному посібнику розглядати їх не будемо, оскільки їх програмування виходить далеко за рамки елементарного візуального програмування. Однак, якщо вас зацікавлять які-небудь спеціальні пакети в Delphi (наприклад OpenGL), програму доведеться писати з використанням повідомлень.

Зараз же ми напишемо невелику програму, що зведеться до призначення деяких властивостей обмеженого числа компонентів і програмування елементарних методів.

Як приклад розглянемо створення діалогу за допомогою візуального компонента SpeedButton , розміщеного на закладці Additional.

Приклад 3. Припустимо, нам необхідно створити програму, у якій на форму винесено дві великі кнопки з картинками (саме для цього й використовується компонент SpeedButton). На одній кнопці зображена кішка, на другій — мишка. Спочатку кішка стоїть навпроти мишачої нірки, звідки стирчить тільки мишачий хвостик. Коли користувачеві набридає чекати, він може натиснути на кнопку «Кішка». Кішка нявкає й лягає спати. З нірки з'являється мордочка миші. Якщо тепер нажати на кнопку «Мишка», миша вискакує з нірки, бачить кішку, лякається й ховається в нірку, лишивши зовні тільки хвостик. При цьому кішка прокидається і знову стоїть біля нірки. Усі чотири фази дії цієї програми показано на рис. 4.3.

Якщо відволіктися від образного опису завдання, можна узагальнити: програміст має тут справу з проблемою триразової зміни картинок на кнопках. Картинка кішки: в активному стані вона «стоїть», у недоступному — «спить», у натиснутому — «нявкає». Картинка мишки: в активному стані вона «визирає», у недоступному — «сховалась», у натиснутому — «вибігає».



Можливість замінювати так картинки вже передбачена самим компонентом `SpeedButton`. У цьому випадку, хоча це й може здатися дивним, написання програми полягає в основному не в записуванні програмного коду, а в роботі в графічному редакторі. З нього й почнемо.

Компонент `SpeedButton` має властивість `Glyph`, у яку вписується ім'я файлу, що містить картинку для виведення на кнопку. Визначення цієї властивості слід робити, коли картинка буде готова. Особливість компонента полягає в тому, що в різних режимах він може виводити одну, дві або три картинки залежно від стану кнопки.

Картинки не масштабуються. Тому, якщо наші кнопки, наприклад, мають розміри 50×50 , у графічному редакторі слід малювати картинку заввишки 45 пікселів і завширшки 135 пікселів (тобто тричі по 45 пікселів). Зручніше це зробити, нарисувавши спочатку три одноколірні рамочки (рис. 4.4), які потім будуть стерті.

Порядок малювання картинки також визначається компонентом `SpeedButton`: її лівий фрагмент виводиться з ненатиснутою активною кнопкою, середній — з відключеною (властивість «доступність» `Enabled` установлена в стан `false`), правий — тимчасово виводиться в момент натискання кнопки. Щоб картинка не зміщувалася, лівий і правий її фрагменти краще копіювати, змінюючи тільки частину деталей (рис. 4.5). Після закінчення малювання опорні рамочки потрібно стерти.



Вихідний стан



Після натискання кнопки «Кішка»



Під час натискання кнопки «Кішка»



Під час натискання кнопки «Мишка»

Рис. 4.3. Реакція програми на різні натискання кнопок



Рис. 4.4. Прорисовування рамочок для виконання трьох рівних фрагментів малюнка



Рис. 4.5. Рисунок кішки (три фрагменти), підготований для вставлення в `Glyph`

Так само виконуємо три фрагменти малюнка мишки (рис. 4.6).

Тепер внесемо картинку кішки на кнопки. Виділяємо кнопку, обираємо властивість `Glyph`, клацанням мишею на полі введення вносимо потрібні значення й у діалозі, що відкривається, обираємо ім'я файлу. За замовчуванням у властивості `NumGlyphs` (кількість картинок) встановлено значення 1, тобто одна картинка (виводиться центральний фрагмент виконаного малюнка). Міняємо це значення на 3 — у нас заготовані три картинки на кнопку. Аналогічно чинимо й з картинками мишки (рис. 4.7).



Рис. 4.6. Рисунок мишки (три фрагменти), підготований для вставлення в `Glyph`



Рис. 4.7. У режимі проектування малюнок вставлено в `SpeedButton`

Після цього розроблену нами форму можна запустити. Незважаючи на те що ми не написали жодного рядка коду, створений додаток має цілком працездатний вигляд: він запускається, кнопки натискаються, на кнопках змінюються картини, однак для одержання програми, відповідної до завдання, все-таки доведеться написати кілька рядків програмного коду.

Отже, подвійним клацанням мишею на порожній кнопці відкриваємо заготовку методу для написання коду. Це саме можна реалізувати, вибравши в `Object Inspector` рядок обробника `OnClick`. На кнопці `SpeedButton1` із зображенням кішки записуємо в процедуру рядки:

```
-----
SpeedButton1.enabled:=false;
SpeedButton2.enabled:=true;
-----
```

Далі на кнопці `SpeedButton2` із зображенням мишки записуємо аналогічні рядки, але тепер «недоступною» стає кнопка 2, а «доступною» — кнопка 1.

```
-----
SpeedButton2.enabled:=false;
SpeedButton1.enabled:=true;
-----
```

Тепер програма діє відповідно до завдання.

Звернемо увагу на правила інтерпретації малюнка. Коли вставляється компонент `Image`, ліва нижня точка малюнка (піксел) визначає

«колір прозорості», якщо у властивості `Transparent` встановлено значення `true`. На кнопці відображаються не всі точки цього кольору, і на малюнку «просвічує» сіре тло.

У властивість `NumGlyphs` можна вводити числа — від 1 до 4. Якщо там міститься 1, то одна й та сама картинка виводиться на поверхню кнопки незалежно від її стану. Якщо `NumGlyphs = 2`, лівий фрагмент малюнка виводиться на кнопку в активному стані (у тому числі й натиснутому), а правий — у недоступному. Що буде, якщо задати цифру 3, описано в наведеному вище прикладі. Якщо ж у зазначену властивість введено число 4, то в нас є четверта картинка для фіксації кнопки в натиснутому стані. Для «кнопок із фіксацією» на формі потрібно розташувати кілька `SpeedButton`, у яких у властивості `GroupIndex` стоїть одне й те саме число. У такому випадку ці кнопки вважаються «групою»; у разі натискання однієї кнопки вона фіксується, а решта «відтискаються» у вихідний активний стан. На зафіксованій кнопці й виводиться четвертий фрагмент малюнка. Такий режим є зручним для програмування яких-небудь автоматизованих тестів із вибором одного з декількох варіантів відповідей.

Пропонуємо вам іще один жартівливий приклад.

Приклад 4. За аналогією до попереднього завдання потрібно розробити програму з формою та двома `SpeedButton` — тепер на тему «Березневі коти». На кнопках мають бути картинки кота, який не спить, і кота, який спить. Якщо натиснути на кнопку з котом, який не спить, він «кричить», потім лягає спати, після чого другий кіт (друга кнопка) прокидається — і навпаки (рис. 4.8).

Особливість цього прикладу полягає в тому, що коли у вас є готова програма з попереднього прикладу, то вам не доведеться змінювати жоден рядок програмного коду або ж малювати нові картинки. Досить змінити тільки одну властивість: у `SpeedButton2` поміняти назву файлу в параметрі `Glyph` (тепер це один і той самий файл малюнка і для першої, і для другої кнопки). Програмний код збігається з кодом програми «про кота й мишку».



Рис. 4.8. Інтерфейс програми «Березневі коти»



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Для чого призначені компоненти `MainMenu`, `PopupMenu`, `Label` і `Edit`, що містяться в палітрі компонентів `Standard`?
2. Яке призначення мають компоненти `Memo`, `Button`, `CheckBox` і `RadioButton`, що є в палітрі компонентів `Standard`?
3. Яке призначення мають компоненти `ListBox`, `ComboBox`, `GroupBox` і `RadioGroup`, що містяться в палітрі компонентів `Standard`?

4. Для чого призначені компоненти `BitBtn`, `SpeedButton`, `StringGrid` і `DrawGrid`, що є в палітрі компонентів `Additional`?
5. Яким є призначення компонентів `Image`, `Shape`, `CheckListBox` і `StaticText`, що містяться в палітрі компонентів `Additional`?
6. Що собою являють візуальні компоненти?
7. У чому полягає особливість повідомлень і контекстів?

4.2

Використання вікон

Візуальне середовище Delphi дозволяє застосовувати в програмах стандартні вікна, імпортовані з WinAPI. Вам, без сумніву, траплялося бачити ці маленькі сірі віконця в середовищі Windows, коли на екран виводилося попередження про неможливість виконати якусь операцію або ж було потрібно підтвердити введення пароля. Вікон таких кілька, і можливості вводити дані за їх допомогою дуже обмежені. Проте вікна мають і позитивні властивості. По-перше, їх легко програмувати (по суті, це просто рядок звернення до відповідної функції WinAPI); по-друге, вони є вікнами модельного діалогу — можуть переривати програму й дозволяють ввести дані або підтвердити яку-небудь дію навіть із циклу або рекурсивної функції.

➡ Вікно повідомлень

Функція `MessageBox` є зручним засобом для виведення на екран невеликого повідомлення:

`k := messagebox` (номер вікна, заголовок, рядок, код кнопок)

Розглянемо параметри цієї функції.

Номер вікна-власника — це числове значення, атрибут вікна, яке викликало вікно попереджень. Використовується для багатовіконних користувацьких додатків. В одновіконному додатку в «номері вікна» ставиться 0 (без вікна-власника).

Заголовок — рядок, що буде написаний у заголовку вікна повідомлень.

Рядок — текстова інформація, що виводиться у вікні повідомлень. Зауважимо, що ширина виведеного вікна визначається довжиною цього рядка або сумарною шириною кнопок (якщо рядок виявиться коротшим за сумарну ширину кнопок). Висота вікна — це висота рядка плюс висота кнопок. Якщо потрібен рядок, що складається з декількох рядків, у його текст необхідно вписати коди переведення рядка. Наприклад: 'Два веселі їжаки' + #13 + 'Накололи на голки' + #13 + 'Всі листки'

Шістнадцятковий запис #13 у рядку відповідає клавіші Enter.

Код кнопок визначає, який їх набір буде виведений у вікні повідомлень:

MB_ABORTRETRYIGNORE	— Abort, Retry i Ignore
MB_OK	— OK (Так за замовчуванням)
MB_OKCANCEL	— OK i Cancel
MB_RETRYCANCEL	— Retry i Cancel
MB_YESNO	— Yes i No
MB_YESNOCANCEL	— Yes, No i Cancel

Іконка у вікні визначається константами:

MB_ICONEXCLAMATION, MB_ICONWARNING — знак «!».
 MB_ICONINFORMATION, MB_ICONASTERISK — знак «i».
 MB_ICONQUESTION — знак «?».
 MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND — знак «x».

Можна задати, яка з кнопок буде виділена за замовчуванням:

MB_DEFBUTTON1 ... MB_DEFBUTTON4.

Для оборотних значень залежно від того, яку кнопку натиснуто, є відповідні константи: IDABORT, IDCANCEL, IDIGNORE, IDNO, IDOK, IDRETRY, IDYES. Або замість них можна використовувати цілі числові значення, наведені в табл. 4.1.

Таблиця 4.1

Кнопки вікна повідомлень та їхні позначення

Кнопка	Константа	Значення
OK	IDOK	1
CANCEL	IDCANCEL	2
ABORT	IDABORT	3
RETRY	IDRETRY	4
IGNORE	IDIGNORE	5
YES	IDYES	6
NO	IDNO	7

Функцію MessageBox можна використовувати як процедуру без повернення значення.

Як приклад напишемо програму, що виводила б вікно повідомлень так, як показано на рис. 4.9. Залежно від вибраної відповіді потрібно вивести в Label1 «Так» або «Ні».

Текст програми матиме такий вигляд:

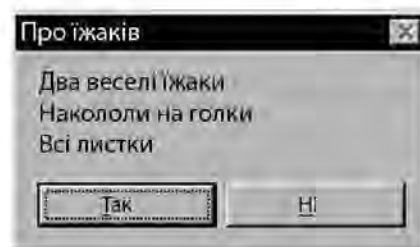


Рис. 4.9. Вікно повідомлень із декількома рядками

```

-----
procedure TForm1.Button1Click(Sender: TObject);
var
  k:integer;
begin
  k:=messagebox(0,'Два веселі іжаки' + #13 + 'Накололи на голки'
+ #13 + 'Всі листки', 'Про іжаків',mb_okcancel);
  if k=1 then
    label1.caption:= 'Так'
  else
    label1.caption:= 'Hi';
end;
-----

```

➤ Вікно попереджень

На відміну від вікна повідомлень, у вікні попереджень можна розмістити будь-яку комбінацію з восьми допустимих кнопок, крім того, воно дозволяє повернути два числові параметри. Викликається вікно попереджень таким рядком:


```
k:=messagedlg(строка, тип_вікна, перелік_кнопок, число_help)
```


Розглянемо значення параметрів у разі виклику вікна попереджень.


Рядок — текст, що виводиться у вікні попереджень.


Тип вікна — цей параметр визначається значком, який буде зображений у вікні. Можливі такі константи:

mtWarning — знак «!» у трикутнику ;

mtError — знак «x» у кружечку ;

mtInformation — літера «i» (синя) в білому кружечку ;

mtConfirmation — знак «?» у кружечку ;

mtCustom — порожнє вікно без знака .

Список кнопок — множина зарезервованих констант. Допустимими є такі значення, кожне з яких визначає відповідну кнопку: mbOk — кнопка «Ok», mbCancel — кнопка «Cancel», mbYes — кнопка «Yes», mbNo — кнопка «No», mbAbort — кнопка «Abort», mbRetry — кнопка «Retry», mbIgnore — кнопка «Ignore», mbAll — кнопка «All», mbNoToAll — кнопка «NoToAll», YesToAll — кнопка «YesToAll».

Кнопки на вікні визначаються множиною у квадратних дужках. Якщо випадково двічі укажемо mbYes, mbYes, mbNo, то на формі з'являться тільки кнопки No і Yes, причому, як у звичайних множинах, порядок їх запису в операторі не має значення — вони виводяться в порядку зростання кодів повернення.

Наприклад, потрібно завершити діалог натисканням однієї з трьох кнопок: All, NoToAll або YesToAll. Для цього у квадратних дужках слід записати: [mbAll, mbNoToAll, mbYesToAll].

Зверніть увагу на те, що натискання кнопки повертає відповідне значення:

mrNone — нічого не натиснуто (0); mrOk — кнопка Ok (1);
 mrCancel — кнопка Cancel (2); mrAbort — кнопка Abort (3);
 mrRetry — кнопка Retry (4); mrIgnore — кнопка Ignore (5);
 mrYes — кнопка Yes (6); mrNo — кнопка No (7);
 mrAll — кнопка All (10).

Число_help — числовий код для виклику контекстного Help-у.

Функцію виклику вікна попереджень можна використовувати як процедуру без повернення значення, тоді вона виконує завдання тимчасового припинення програми з виведенням заданого значення.

Ширина вікна визначається довжиною рядка або сумарною шириною кнопочок. Висота — це рядок плюс кнопочка.

Наведемо приклад використання вікна попереджень.

Потрібно вивести на екран вікно `MessageDlg` із двома кнопками Yes і No (рис. 4.10, а), а потім — вікно з відповіддю «Так» (рис. 4.10, б) або «Ні» (рис. 4.10, в) залежно від вибору кнопки в першому вікні.

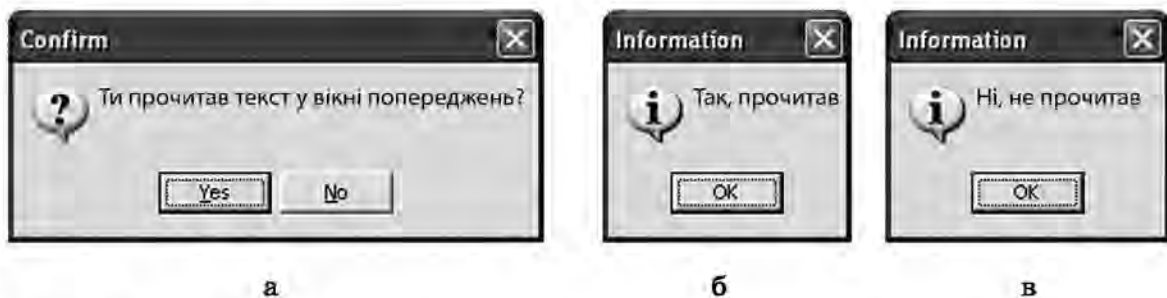


Рис. 4.10. Вікна попереджень

Текст програми матиме такий вигляд:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin
  if MessageDlg('Ти прочитав текст у вікні попереджень?',
    mtConfirmation, [mbYes, mbNo], 0) = mrYes then
    MessageDlg('Так, прочитав', mtInformation, [mbOk], 0)
  else
    MessageDlg('Ні, не прочитав', mtInformation, [mbOk], 0);
end;
-----
```

➔ Позиціоноване вікно попереджень `MessageDlgPos`

Вікна цього типу повністю повторюють звичайне вікно попереджень, однак вони можуть виводитись у довільній точці екрана:

```
k:=messagedlg(рядок, тип_вікна, перелік_кнопок,
число_help, x,y)
```

Параметри мають ті самі значення, що й у `MessageDlg`. Додано ще два параметри — зсуви по горизонталі й по вертикалі. Відлік ведеться від верхнього лівого кута екрана, а не від форми програми.

Наведемо текст програми, що виводить позиціоноване вікно попереджень так, як показано на рис. 4.11:

```
var
  s: string;
begin
  messagedlgpos ('Виведення в будь-яку точку екрана',
    mtconfirmation, mbYesNoCancel, 0, 100, 130);
end;
```

➔ Вікно введення `InputBox`

Це вікно призначене для введення рядка. Вікно викликається такою функцією:

`s := messagebox` (заголовок, підказка, рядок_за_замовчуванням)

Параметри функції:

заголовок — напис на заголовній смузі вікна `InputQuery`;

підказка — рядок тексту, що виводиться над елементом `Edit` у цьому вікні;

рядок_за_замовчуванням — вихідний текст у вікні до початку введення.

Розглянемо приклад використання вікна введення.

Потрібно написати параметри для реєстрації гравця в разі успішно завершеної гри (рис. 4.12).

Текст програми матиме такий вигляд:

```
Var
  Inputstring: string;
begin
  inputstring := inputbox ('Новий рекорд', 'Введи своє ім'я',
    'Noname');
end;
```

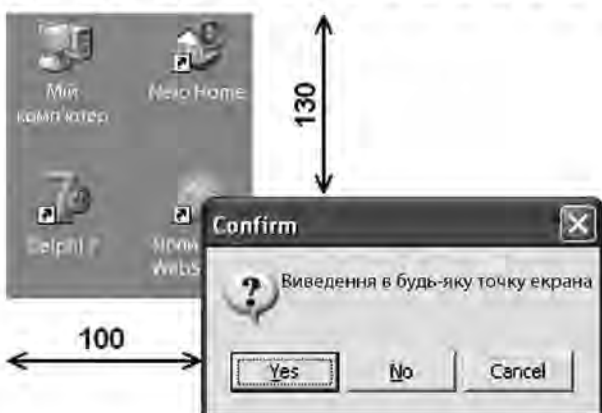


Рис. 4.11. Фрагмент Робочого столу й позиціоноване вікно попереджень

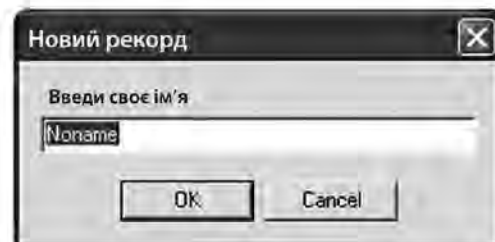


Рис. 4.12. Введення рядка за допомогою вікна `InputBox`

Рядок, набраний в елементі введення, повертається в програму й може бути присвоєний будь-якій рядковій змінній.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Перелічіть можливості, які дає вікно повідомлень `MessageBox`.
2. Перелічіть можливості, які дає вікно попереджень `MessageDlg`.
3. Для чого призначене вікно введення `InputBox`?

Практична робота № 3

Розроблення форм та розміщення на них елементів керування

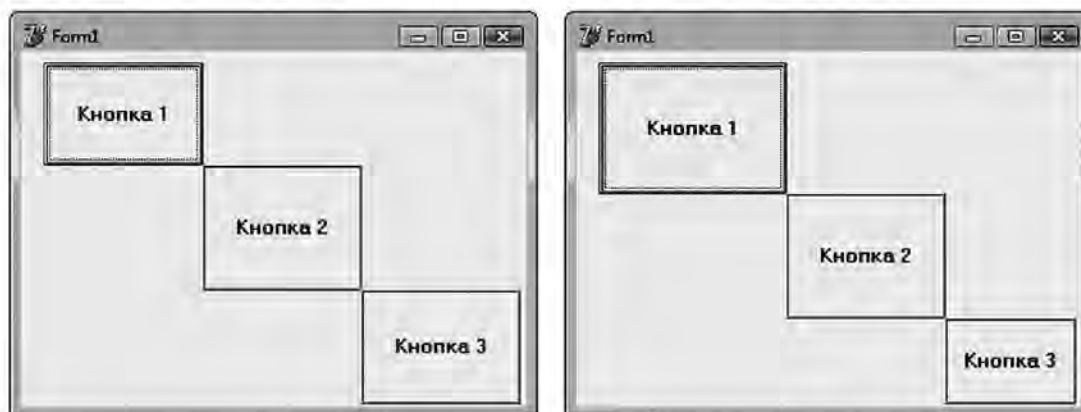
Мета роботи: навчитися створювати прості форми, виносити на них елементи і програмно керувати ними.



Завдання. На формі розміщені три кнопки. Записати процедури, які містяться в кнопках 1 і 2. Унаслідок натискання кнопки 1 її ширина й висота збільшуються на п'ять пікселів. Форма і кнопка 2 не змінюють розмірів. У міру збільшення кнопки 1 кнопка 2 зміщується, а кнопка 3 зменшується в розмірах. Якщо натиснути кнопку 2, описаний процес іде у зворотному порядку — кнопка 3 збільшується, кнопка 1 зменшується. Унаслідок натискання на кнопку 3 форма закривається. Результат натискання на кнопку 1 показано на рис. 4.13.

Хід роботи

1. Запустимо середовище Delphi та створимо новий додаток.
2. Розмістимо на формі три кнопки, перенісши їх з палітри компонентів.
3. Ширина й висота кнопки (як і інших видимих компонентів) визначаються параметрами `width` і `height`, значення яких можна задавати в ході проектування форми розтягненням компонентів за куточки за допомогою миші або зміною числових значень в `Object Inspector`. У нашому випадку ці значення слід змінювати



а

б

Рис. 4.13. Керування розмірами кнопок: *а* — до натискання кнопки 1, *б* — після її натискання

програмно. Тому здійснимо подвійне клацання мишею на кнопці 1 і впишемо у відкритий метод обробки натискання `TForm1.Button1Click` такі рядки:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    button1.Width := button1.Width + 5;
    button1.Height := button1.Height + 5;
end;
-----
```

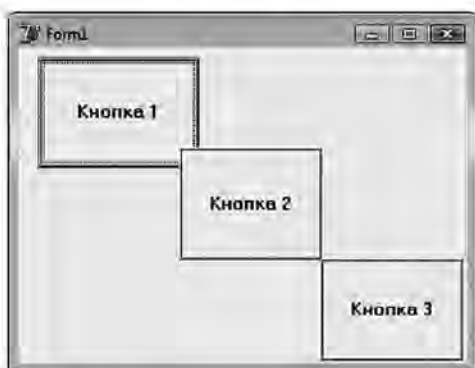


Рис. 4.14. Збільшення кнопки 1 не змінює позицій і розмірів кнопок 2 і 3

4. Якщо тепер запустити програму на компіляцію й виконання, а потім натиснути кнопку 1, одержимо ситуацію, зображену на рис. 4.14: кнопка 1 справді збільшилась, однак кнопки 2 і 3 лишилися без змін. Це природно, оскільки ніякого коду для керування цими кнопками ми не писали.

5. Змістимо кнопку 2 праворуч униз. За розташування кнопки на формі відповідають параметри `left` і `top`. Їх також можна задавати в ході проектування форми розтягненням компонентів за куточки за допомогою миші або зміною числових значень в `Object Inspector`. У нашому випадку ці значення слід змінювати програмно.

Нагадаємо, що чим більшим є значення `left`, тим більше зміщується праворуч компонент на формі. Чим більше `top`, тим нижче розміщується компонент. Вписуємо в метод `TForm1.Button1Click` такі рядки:

```
-----
button2.Top:=button2.Top + 5;
button2.Left:=button2.Left + 5;
-----
```

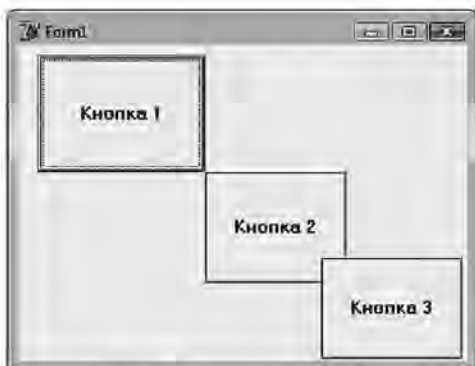


Рис. 4.15. Кнопка 1 збільшилася, кнопка 2 змістилася, кнопка 3 — без змін

Якщо тепер запустити програму на виконання, то в разі натискання на кнопку 1 вона буде збільшуватись, а кнопка 2 зміщуватись праворуч униз, при цьому кнопка 3 залишатиметься на місці без змін у розмірах (рис. 4.15).

6. Зменшимо розміри кнопки 3, дописавши в метод `TForm1.Button1Click` такі рядки:

```
-----
button3.Width:=button3.Width - 5;
button3.Height:=button3.Height - 5;
-----
```

Тепер після запуску програми й натискання на кнопку 1 перші дві кнопки поводяться, як і раніше, а третя зменшується у розмірах (рис. 4.16).

7. Залишилося тільки змістити кнопку 3. Як і у випадку з кнопкою 2, це досягається програмними рядками:

```
button3.Top:=button3.Top + 5;
button3.Left:=-button3.Left + 5;
```

Тепер кнопка 1 поводить ся так, як це описано на самому початку (див. рис. 4.13).

8. Метод, який виконується внаслідок натискання на кнопку 2, фактично повторює дії методу кнопки 1, але змінює всі числові значення у зворотному напрямку. Іншими словами, коли процедура в кнопці 1 збільшує якийсь параметр, то процедура в кнопці 2 — зменшує, і навпаки. Подібні операції легко запрограмувати, скопіювавши зміст кнопки 1 на кнопку 2 і замінивши знаки «+» на «-» і навпаки. Очевидно, що для цього буде використано метод TForm1.Button2Click:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    button1.Width:=button1.Width - 5;
    button1.Height:=button1.Height - 5;
    button2.Top:=button2.Top - 5;
    button2.Left:=button2.Left - 5;
    button3.Width:=button3.Width + 5;
    button3.Height:=button3.Height + 5;
    button3.Top:=button3.Top - 5;
    button3.Left:=button3.Left - 5;
end;
```

9. Закриття форми за допомогою кнопки 3 здійснюється викликом процедури закриття форми:

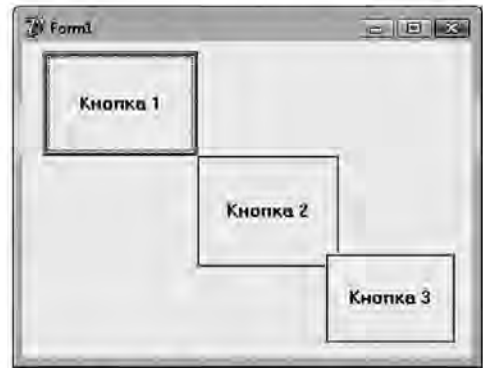


Рис. 4.16. Кнопка 1 збільшилася, кнопка 2 змістилася, кнопка 3 зменшилася

```

procedure TForm1.Button3Click(Sender: TObject);
begin
    close;
end;

```

10. Задачу розв'язано. Збережемо створений нами додаток на диску (бажано заздалегідь створити папку) і запустимо його на виконання.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. Записати процедури, які містяться в кнопках 1 і 2. Унаслідок натискання кнопки 1 її ширина зменшується на 6 пікселів за рахунок збільшення кнопок 2 і 3, а внаслідок натискання кнопки 2 все відбувається навпаки. Натисканням кнопки 3 форма закривається (рис. 4.17).

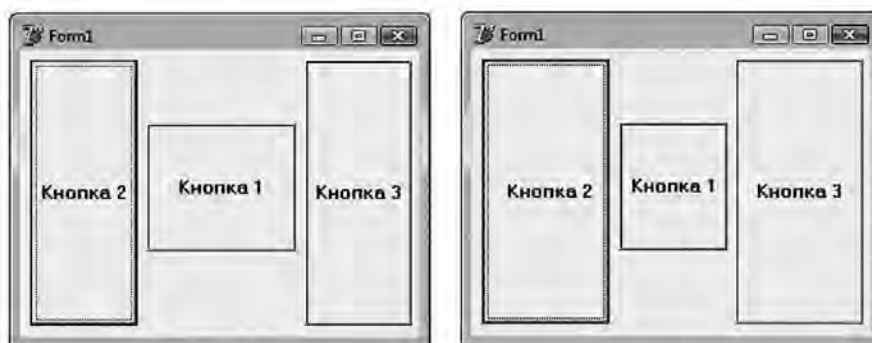


Рис. 4.17. Розташування кнопок і змінення їхніх розмірів, ширина кнопки 1 зменшується

2. Записати процедури, що містяться в кнопках 1 і 2. Унаслідок натискання кнопки 1 вона стає ширшою й вищою на 5 пікселів за рахунок кнопок 2 і 3. Якщо натиснути кнопку 2, все відбувається навпаки. Натисканням кнопки 3 форма закривається (рис. 4.18).

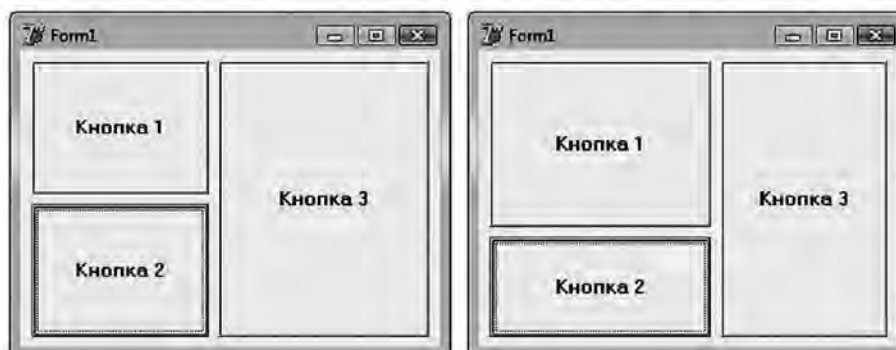


Рис. 4.18. Розташування кнопок і змінення їхніх розмірів, висота кнопки 2 та ширина кнопки 3 зменшуються

3. Записати процедури, що містяться в кнопках 1 і 2. Унаслідок натискання кнопки 1 вона стає ширшою на 5 пікселів за рахунок кнопки 3, одночасно збільшується й кнопка 2. Якщо натиснути кнопку 2, все відбувається навпаки. Натисканням кнопки 3 форма закривається (рис. 4.19).

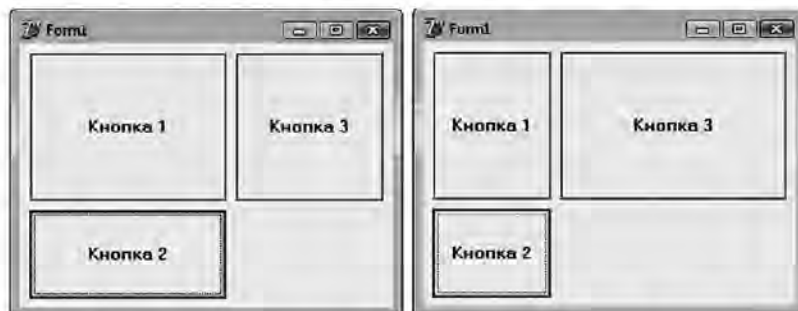


Рис. 4.19. Розташування кнопок і змінення їхніх розмірів, ширина кнопок 1 і 2 зменшується

4. Записати процедури, що містяться в кнопках 1 і 2. Унаслідок натискання кнопки 1 вона разом із кнопкою 2 переміщується в праву частину форми, а кнопка 3 — у ліву. Якщо натиснути кнопку 2, всі три кнопки повертаються назад. Вважати, що за шириною кнопки 1 і 3 займають усю форму (між ними немає просвіту). Натисканням кнопки 3 форма закривається (рис. 4.20).



Рис. 4.20. Зміна розташування кнопок на формі

5. Записати процедури, що містяться в кнопках 2 і 3. Унаслідок натискання кнопки 2 вона змінюється, як це показано на рис. 4.21. Кнопка 3 займає місце кнопки 2. Якщо натиснути кнопку 3, це переміщення відбувається у зворотному порядку. Натисканням кнопки 1 форма закривається.

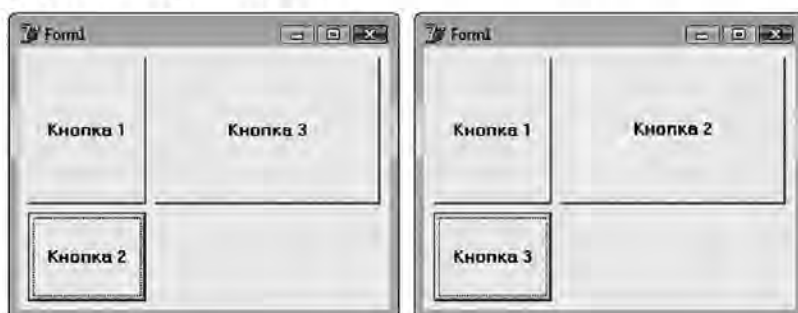


Рис. 4.21. Розташування кнопок і змінення їхніх розмірів, ширина кнопок 1 і 3 зменшується



Звичайна програма оперує константами й змінними, у візуальному програмуванні до цього додаються відеокомпоненти, властивості яких також можуть використовуватись у програмі як змінні. У цьому розділі ми будемо розглядати типи даних, що існують у Delphi, описувати змінні й власні типи даних, використовувати результати розрахунків для змінювання властивостей відеокомпонентів, отримуючи цим візуальний ефект від реалізації заданого алгоритму.

5.1

Поняття змінної і константи. Поняття ідентифікатора. Установлення значень атрибутів форм і елементів керування в програмі

➔ **Поняття змінної і константи**

Ідея позначати проміжні і кінцеві результати обчислень літерами, а потім використовувати їх як змінні виникла в математиків ще в часи стародавнього світу. Природно, що в перших мовах програмування, які використовувалися для прискорення формульних обчислень, механізм змінних і констант був цілком запозичений (і надалі розвинений) із математики.

Як і в інших мовах програмування, у мові Pascal (і в Delphi, яка успадкувала від неї основні конструкції) дані поділяються на константи і змінні. У програмі вони визначаються ідентифікаторами (іменами), за якими до них можна звертатися для одержання поточних значень і включати їх в арифметичні, логічні або рядкові вирази.

Константами називаються елементи даних, значення яких установлені в описовій частині програми й далі не змінюються. Наприклад, якщо ви використовуєте в програмі значення кількості учнів у вашому класі, то найкраще задати його константою, бо в разі обчислення, наприклад, тих чи інших параметрів успішності ця кількість не змінює свого значення. Якщо ж до вашого класу прийде новий учень, досить збільшити константу на одиницю, і всі розрахунки будуть виконуватися з новим значенням чисельності. Це значно простіше, ніж переглядати всю програму, змінюючи кількість, задану числом, на нове значення.

Константи описуються в спеціальному розділі, що починається зарезервованим словом **const** (від англ. *constant* — константа) у форматі:

const

<ідентифікатор> = <значення константи>;

Наприклад:

const

MySchool = 'ліцей «Зоряний»';

MinTemp = -273;

Inch = 2.54;

Тип користувацьких констант розпізнається компілятором без попереднього опису.

У Pascal є ряд констант, до значень яких можна звертатися без попереднього визначення, наприклад π . Їх називають *зарезервованими константами*.

Змінними називають величини, які можуть змінювати свої значення в процесі виконання програми. Кожна змінна, як і константа, належить до певного типу даних.

Тип змінних має бути відомий компілятору до початку трансляції виразів і операторів із використанням цих змінних. Крім того, згідно з відомим типом даних компілятор виділяє під змінні в оперативній пам'яті необхідне місце (потрібна кількість байтів повністю визначається застосовуваним типом).

Змінні описуються в спеціальному розділі, що починається зарезервованим словом **var** (від англ. *variable* — змінна), у форматі:

var

<ідентифікатор> : <тип>;

Наприклад:

var

a, b, c : integer;

summ : real;

Крім констант і змінних, існують так звані типізовані константи, значення яких можуть змінюватися в ході виконання програми. Для них слово «константа» вказує лише на те, що дані цього типу описуються в розділі **const**, а слово «типізована» — на те, що цей тип даних має позначатися, як у змінних.

Формат опису типізованих констант може бути реалізований в одному з двох таких варіантів:

const

<ідентифікатор> : <тип> = <значення>;

або:

var

<ідентифікатор> : <тип> = <значення>;

Приклад:

```
var
  r:real = 57.29;
const
  language : string='Delphi';
```

У процесі програмування типізовану константу можна використувати як змінну із заздалегідь заданим значенням, але для цього необхідно виконати одну настройку в інтерфейсі Delphi. Заходимо через основне меню в пункт Project. У висхідному списку обираємо рядок Options. Відкривається багатосторінковий діалог, де на закладці Compiler для дій із типізованими константами має стояти галочка в рядку «Assignable typed constants» (рис. 5.1).

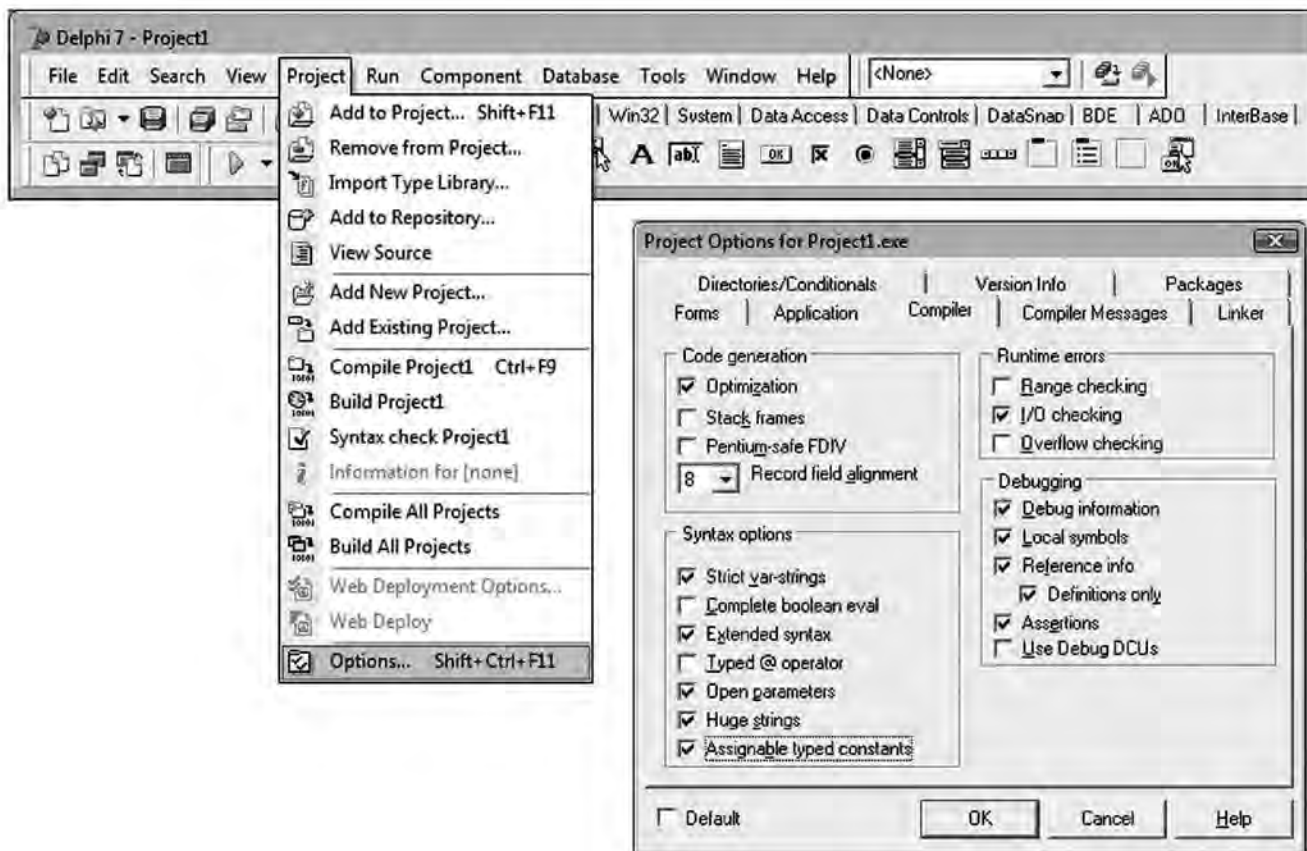


Рис. 5.1. Установлення в інтерфейсі дозволу на зміну типізованих констант

➡ Поняття ідентифікатора

У математиці для позначення змінних і констант використовують літери латинського алфавіту. Літер цих мало — усього 26, тому іноді залучають грецькі й готичні літери, надрядкові та підрядкові індекси: y , x_1 , $S_{\text{середн}}$, δ , \mathfrak{K} та ін. Обчислювальна техніка від самого початку орієнтувалася на використання латинських літер без надрядкових

і підрядкових значків. Тому для позначення змінних і сталих величин, різних процедур та функцій використовують *ідентифікатори* — імена, що встановлюють відповідність між об'єктом і певним набором символів (латинських літер, знаків підкреслення й цифр; цифра не може бути першим символом).

Можливості програміста у виборі імен для своїх змінних досить широкі. Не можна тільки використовувати як ідентифікатори зарезервовані слова, які компілятор сприймає як базові директиви або оператори програми. Наприклад, не можна назвати змінну **begin** або **for**. Не рекомендовано брати як змінні й імена стандартних ідентифікаторів.

Стандартні ідентифікатори введені творцями мови для позначення типів даних, констант, процедур і функцій, наприклад: `integer`, `sin`, `read`, `readln`, `write`, `writeln`. Стандартний ідентифікатор `sin`, поданий у цьому далеко не повному списку, викликає функцію, що обчислює синус заданого кута; ідентифікатори `read` і `readln` викликають процедуру, яка організовує введення даних, а `write` і `writeln` — процедуру, яка організовує виведення даних. На відміну від зарезервованих слів, кожен зі стандартних ідентифікаторів можна перевизначити, але після цього програму важко читати й змінювати, так що краще цієї можливості не користуватися.



Користувацькі ідентифікатори програміст вводить для позначення констант, змінних, процедур і функцій у програмі. При цьому ідентифікатори мають бути унікальними: у кожному програмному блоці один і той самий ідентифікатор не може використовуватися для позначення більш ніж однієї змінної, функції або константи. Дублювання ідентифікаторів спричиняє помилку на етапі компіляції.

В ідентифікатор мовою Delphi не можуть входити пробіли, спеціальні символи й літери будь-яких алфавітів, окрім англійського (латинського).

Правильно обрані ідентифікатори значно полегшують читання й розуміння програми, а також зменшують імовірність появи помилок, якщо в неї вноситимуть зміни.

➡ **Установлення значень атрибутів форм і елементів керування в програмі**

У ході візуального проектування форми програміст виносить на поле форми необхідні йому відеокомпоненти. Робиться це простим перетяганням мишею необхідного відеокомпонента з інструментальної лінійки компонентів на форму. Як уже зазначалося в попередніх розділах, вигляд відеокомпонента можна суттєво змінювати, міняючи його властивості через `Object Inspector`. Властивості форм і відеокомпонентів (або, як ще кажуть, їхні атрибути), по суті, є змінними і як такі можуть використовуватись у програмі. Їхня основна відмінність від користувацьких змінних — нерозривний зв'язок з об'єктом, якому вони належать. Розроблений творцями Delphi метод визначає, що зміниться в поведінці об'єкта в разі зміни атрибута.

Наприклад, на формі розташовані два компоненти — Label (іконка  на першій закладці Standard) і Timer (іконка  на першій закладці System). Запишемо вже знайомий нам із попередніх розділів обробник події форми OnMouseDown. Цей фрагмент програми буде виконуватися після натискання лівої клавіші миші в той момент, коли покажчик перебуває над формою:

```
-----
procedure TForm1.FormMouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
var
    inter:integer;
    capt:string;
begin
    Timer1.Interval:=100;
    Label1.Caption:='Happy New Year!';
    inter:=2000;
    capt:='Hello!';
end;
-----
```

У пропонованій процедурі відбуваються чотири операції присвоювання. І якщо зміни властивостей компонентів Timer1 і Label1 очевидні (починають швидше виконуватися події таймера, змінюється напис), то зміни змінних до жодних видимих наслідків не приводять. У цьому й полягає відмінність змінних від властивостей відеокомпонентів. Змінні застосовуються тільки для розрахунків і вимагають написання процедур для їх виведення або використання в програмі (для зміни графічних зображень, мультиплікації тощо). А для властивостей відеокомпонентів режим їх використання передбачений у самому середовищі Delphi. Нам досить змінити ці властивості, щоб одержати видимий результат.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення поняття «константа». У чому полягає особливість її використання?
2. Що собою являють змінні? Наведіть приклади їх описів.
3. Дайте визначення поняття «ідентифікатор».
4. Чим відрізняються стандартні і користувацькі ідентифікатори?
5. У чому полягає різниця між змінними і властивостями об'єктів?

5.2

Типи даних. Оператор присвоювання

⇒ Типи даних

Опис типів даних указує компіляторові, як зберігати та обробляти інформацію в програмі, які процедури й функції можна застосувати до даного типу.

До створення мови Pascal, попередниці Delphi, у мовах програмування була повна плутанина. В одних мовах (Fortran) тип даних визначався першою літерою імені, в інших (Basic) — останньою. Існувало і явне задавання типів.

Однією з причин поширення Pascal (а за ним і Delphi) стала перевага на той момент система типів даних, що була найповнішою і найбільш несуперечливою.

Типи даних поділяються на стандартні і створювані. Стандартні типи вже визначені в мові, а змінну такого типу можна оголосити, просто пославшись на ім'я цього типу. Створювані типи програміст мусить конструювати сам відповідно до призначення програми.

Типи даних, обумовлені користувачем, задаються:

- у розділі визначення типів програми;
- у модулі (**unit**), у розділі **interface** або **implementation**;
- у підключеному зовнішньому модулі (**unit**);
- усередині процедур;
- усередині функцій.

Оголошення типів діють у межах того блоку, у якому вони розміщені. Поза цим блоком посилатися на такі типи не можна. Усередині ж вони замінюють (перекривають) зовнішні типи з тим самим ім'ям. Оголошені типи даних можна застосовувати в межах ділянки їх видимості.

Оголошення типів у Pascal є для компілятора правилами, які він повинен «запам'ятати» на випадок, якщо раптом зустріне в програмі посилання на той чи інший тип. Саме по собі оголошення типу не вносить у програму ніяких змін. Тип змінної обмежує її значення й ті операції, які можна виконувати з цими значеннями.

Оголошення типів і змінних схематично можна подати так:

type

```
type1 = type_definition1; //Кожному новому типу
//присвоюється ім'я, потім він визначається через
//уже наявні типи.
type2 = type_definition2; //В одному розділі «type» можна
//оголосити кілька типів. Найпростіше визначення
//типу складається з імені типу, визначеного раніше.
type3 = type1;
```

Нові змінні оголошуються у **var**:

```
var
  var1: type_definitions;
```

Кожній новій змінній присвоюється спочатку ім'я, а потім — тип.
 var2, var3: type_definition4;
 var4: type1;

В одному розділі **var** можна оголосити кілька змінних і присвоїти їм той самий тип. При цьому тип може бути сконструйовано заздалегідь або побудовано безпосередньо у **var**.

До основних типів даних Object Pascal належать:

- прості;
- рядкові — для зберігання послідовностей символів;
- структуровані;
- вказівні — для посилань на змінні заданих типів;
- процедурні — для звертання до процедур і функцій як до змінних;
- варіантні — для зберігання в одній змінній даних різних типів.

Зазвичай ідентифікатори типів (виділені для назви змінні) використовуються тільки у визначенні нових типів або оголошенні змінних. Хоча зазначимо, що є кілька функцій, у яких ім'я типу може використовуватися як аргумент. Наприклад, функція `SizeOf (T)` повертає кількість байтів, які займає змінна типу `T`.

Допустимим є використання імені типу як функції для перетворення типів. Наприклад:

```
Label1.caption:=inttostr(integer('A'));
```

Деякі з визначених в Object Pascal типів мають дуже складну структуру й можуть займати в пам'яті багато місця. При цьому елементи таких типів створені скоріше для подання значень у певному логічному порядку, ніж для того, щоб зменшувати займане місце в пам'яті.

У Delphi визначено й доповнено структуру типів Object Pascal:

- Simple (прості):

ordinal (порядкові типи);	integer (цілий);
character (літерні типи);	boolean (логічний);
enumerated (перелічений);	subrange (діапазон);
real (дійсний);	tdatetime (дата-час).

- Structured (структуровані, складені):

array (масив);	string (рядкові типи);
list и stringlist (списки);	set (множина);
record (запис);	file (файлові типи);
textfile (текстові файли);	class (класи);
class reference (посилання на класи);	interface (інтерфейси);

- pointer (показчик);
- variant (варіант);
- procedural (процедурний);
- type identifier (ідентифікатор типу).

У цей список не внесено типи даних, які підтримуються в Delphi, однак є для цієї мови «зовнішніми», запозиченими з інших програм і додатків. Це, наприклад, тип `bitmap`, призначений для успішного зберігання картинок, а також типи, пов'язані з часом і датою (`DateTime`). Чимало типів даних (`TTable`, `TQuery` та ін.) введено для підтримки баз даних.

Загалом, типів даних дуже багато, і з кожною новою версією Delphi їх кількість тільки зростає. Та знати всі типи даних для програмування в цьому середовищі зовсім не обов'язково. Потрібно добре орієнтуватися тільки в основних типах.

➔ Оператор присвоювання

Присвоювання — механізм у програмуванні, що дозволяє змінювати значення змінних або властивостей об'єктів шляхом заміни старих значень новими. З фізичної точки зору, операція присвоювання полягає в записуванні значень у комірку пам'яті, яка під час компіляції була виділена для зберігання цієї змінної.

Загальний синтаксис простого присвоювання має такий вигляд:

`<вираз ліворуч> := <вираз праворуч>`

Як «вираз ліворуч» можуть використовуватися змінна, типізована константа або властивість об'єкта.

«Вираз праворуч» має позначати величину, яка буде присвоєна об'єкту даних. Це може бути змінна, типізована константа, значення властивості об'єкта або ж допустимий вираз (арифметичний, логічний, рядковий) на їхній основі.

У ролі операторів присвоювання в Delphi виступає знак «:=». Зовні він дуже нагадує математичний знак «=» (порівняння), що може привести до хибного висновку, наче оператор присвоювання є точною копією цього математичного знака. У такому випадку, наприклад, запис $x = x + 1$ з математичної точки зору є повною нісенітницею, однак у програмуванні це вельми поширений оператор $x := x + 1$. Істотною особливістю його є те, що праворуч від знака присвоювання розміщується «старе» значення змінної x , над яким мають бути виконані якісь дії. Результат цих дій (результат виконання арифметичного виразу) повинен бути занесений у ту саму комірку x . Наприклад, якщо в програмі до цього оператора в змінній x зберігалася значення 5, то після виконання оператора в ній буде значення 6.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Яке призначення мають типи даних? Де вони описуються?
2. Перелічіть типи даних, що належать до простих типів.
3. Перелічіть типи даних, що належать до структурованих (складених) типів.
4. Для чого призначена операція присвоювання?



Практична робота № 4

Введення й виведення даних за допомогою елементів керування

Мета роботи: навчитися вводити дані в програму, здійснювати прості арифметичні дії й виводити результат на форму.

Завдання. Ввести три цілих числа й натисканням кнопки вивести на екран їх добуток.

Хід роботи

1. Запустимо середовище Delphi та створимо новий додаток.
2. Спроекуємо форму і винесемо на неї три компоненти Edit, кнопку Button і позначку Label (рис. 5.2).
3. У цій задачі програмується тільки обробник події OnClick кнопки Button1. Оберемо цей метод в Object Inspector і впишемо необхідний програмний код:

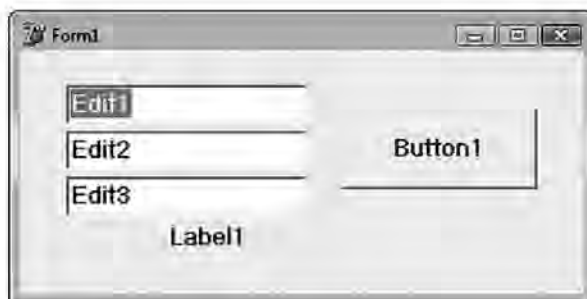


Рис. 5.2. Розміщення відеокомпонентів на формі

```

procedure TForm1.Button1Click(Sender: TObject);
var
    a,b,c,p:integer;
begin
    a:=strtoint(Edit1.Text);
    b:=strtoint(Edit2.Text);
    c:=strtoint(Edit3.Text);
    p:=a*b*c;
    Label1.Caption:=inttostr(p);
end;
  
```

4. Розглянемо рядок за рядком текст розв'язання. Спочатку йде оголошення змінних — у програмі маємо три співмножники цілого типу *a*, *b*, *c* і змінну для збереження добутку *p*. Саме ці змінні описані в розділі **var**.

У виконуваний частині процедури спершу йдуть три рядки, що зчитують вхідні значення з операторів введення Edit (цей відеокомпонент міститься на закладці компонентів Standard). Властивість **text** компонента Edit містить у текстовому вигляді інформацію про рядок, введений у відеокомпонент. Іншими словами, навіть якщо в компонент коректно введено ціле число, відеокомпонент інтерпретує його як рядок. Функція **strtoint** перетворює введений рядок на ціле число. Зверніть увагу: число має бути введене без помилок — якщо в Edit ввести помилковий текст або коректне число,

але дійсного типу, то в разі виклику функції `strtoint` виникне помилка періоду виконання.

Наступний рядок `p:=a*b*c` знаходить добуток трьох цілих чисел `a`, `b`, `c` і присвоює результат змінній `p`.

Нарешті, останній рядок виконуваного коду перетворює отриманий добуток на рядок і присвоює результат властивості `Caption` компонента `Label1`. Функція `inttostr` перетворює ціле число на відповідний рядок.

5. Після розв'язання задачі збережемо проект на диску й запустимо його.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. На формі розміщені п'ять компонентів `Edit`, кнопка й `Label`. Ввести в `Label` середнє арифметичне введених чисел.
2. На формі розміщені три компоненти `Edit`, кнопка й `Label`. Ввести три дійсних числа (це довжина трьох сторін трикутника). Знайти за формулою Герона площу цього трикутника.

Примітка. Функція «корінь квадратний» записується в Delphi як `sqrt` (аргумент).

3. На формі розміщені три компоненти `Edit`, кнопка і два `Label`. Ввести три дійсні коефіцієнти a , b , c квадратного рівняння $ax^2 + bx + c = 0$. Після натискання кнопки вивести у два різні `Label` два значення коренів x_1 і x_2 .

Примітка. У цій задачі передбачено роботу з дійсними числами (на відміну від попередніх прикладів, перетворення дійсного числа на рядок і навпаки здійснюється функціями `floattostr` и `strtfloat` ()).




Написати працездатну складну програму з першого разу дуже непросто. Звичайно, середовище Delphi допоможе у виправленні синтаксичних помилок, однак припуститися неточності або помилки можна й у формально правильно записаному операторі. У подібних випадках нам допоможе режим налагодження та покрокового виконання, в якому програму можна виконати оператор за оператором, аналізуючи значення проміжних змінних. У цьому розділі ми розглянемо технічні можливості налагодження додатків у середовищі Delphi, ознайомимося з методикою виправлення помилок.

6.1

Використання налагоджувача програм у візуальному середовищі програмування. Покрокове виконання програм, перегляд значень змінних під час виконання програми. Різновиди помилок, методи їх пошуку та виправлення

Кваліфікацію програміста визначає не його вміння писати безпомилкові програми (написати складну програму без помилок практично неможливо). Майстерність визначається вмінням розробника швидко, ефективно й надійно налагоджувати і тестувати свій додаток.

Компіляція з далшим виконанням додатка здійснюється за допомогою команди Run (в основному меню) → Run або «гарячої» клавіші F9 (або вибір ). Виконання буде здійснюватися тільки у випадку, якщо в ході компіляції не виявлено помилок і завантажувальний модуль створено. Компіляція без дальшого виконання додатка здійснюється за допомогою команди Project → Compile Project або «гарячих» клавіш Ctrl + F9. У результаті компіляції можуть бути зроблені зауваження — Hint, попередження — Warning і повідомлення про помилки — Error. Наприклад, у лістингу програми, поданої на рис. 6.1, є рядок із попередженням про необхідність використати як параметр циклу FOR локальну змінну та рядок із повідомленням про помилку невідповідності типів Integer і Extended.

Якщо наш додаток відкомпілювався і став виконуватися, це, на жаль, іще не означає, що він правильно працює. У ньому може бути ще безліч помилок, які даватимуться взнаки під час виконання додатка. Це можуть бути постійні помилки, які весь час виникають у ході виконання певних частин нашої програми, або помилки, що виявляються тільки у випадку якихось сполучень даних: помилки ділення на нуль, переповнення, відкриття неіснуючого файлу тощо. Нарешті, можуть бути випадкові перемежовані помилки, коли одна й та сама задача іноді виконується нормально, а іноді — ні. Такі помилки, які найважче

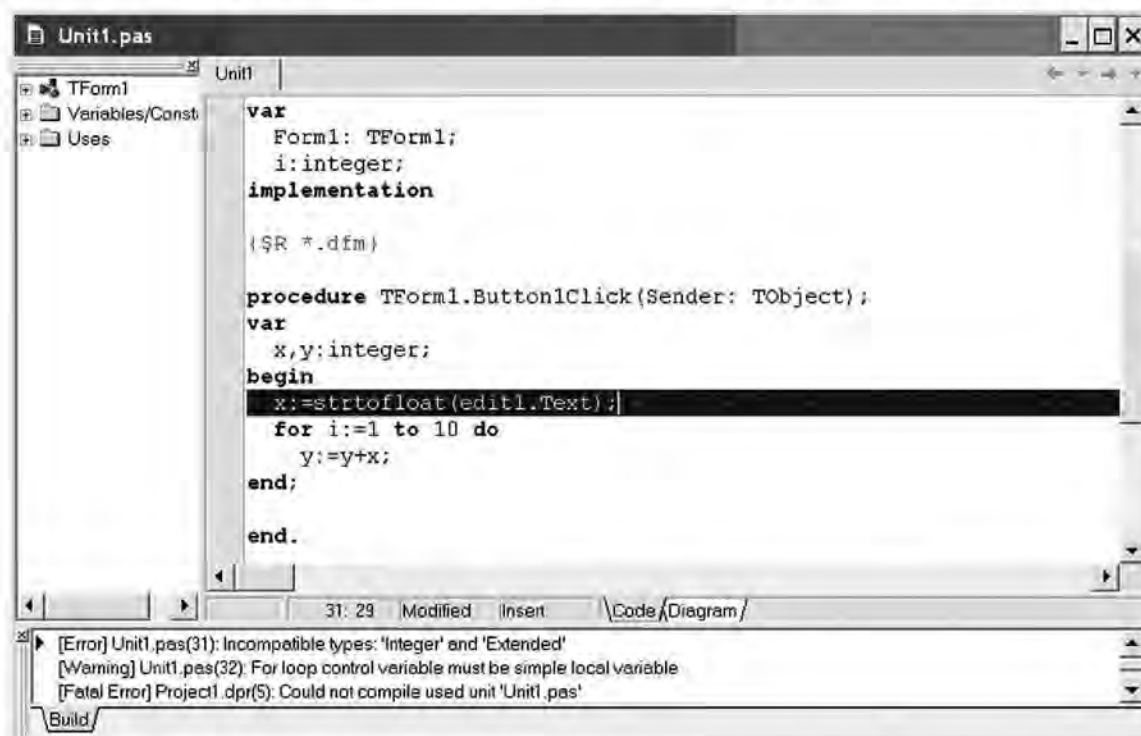


Рис. 6.1. Лістинг програми, що містить попередження про помилку

виявити, зазвичай пов'язані з відсутністю ініціалізації змінних у різних режимах роботи. У цьому випадку виконання додатка залежить від випадкового стану пам'яті комп'ютера.

Дуже часто для того, щоб з'ясувати, у чому причина помилки, необхідно, виконуючи фрагмент програми, відстежувати, як змінюються змінні під час виконання кожної команди. Для покрокового проходження фрагмента програми можна використовувати команди, наведені в табл. 6.1.

Таблиця 6.1

Команди покрокового виконання

Команда	«Гарячі» клавіші	Пояснення
Step Over (Покроково без заходження в...)	F8	Покрокове виконання рядків програми (виклик функції або процедури вважається за один рядок), входження у функції та процедури не відбувається
Trace Into (Трасування із заходженням у...)	F7	Покрокове виконання програми із заходженням у функції та процедури, що викликаються
Trace to Next Source Line (Трасування до наступного рядка)	Shift + F7	Перехід до наступного виконаного рядка

Команда	«Гарячі» клавіші	Пояснення
Run to Cursor (Виконати до курсора)	F4	Команда виконує програму до того виконуваного оператора, на якому розташований курсор у вікні редактора коду
Show Execution Point (Показати точку виконання)	—	Команда поміщує курсор на операторі, що буде виконуватися наступним

Використання на практиці описаних у табл. 6.1 команд проілюструємо прикладом (рис. 6.2).

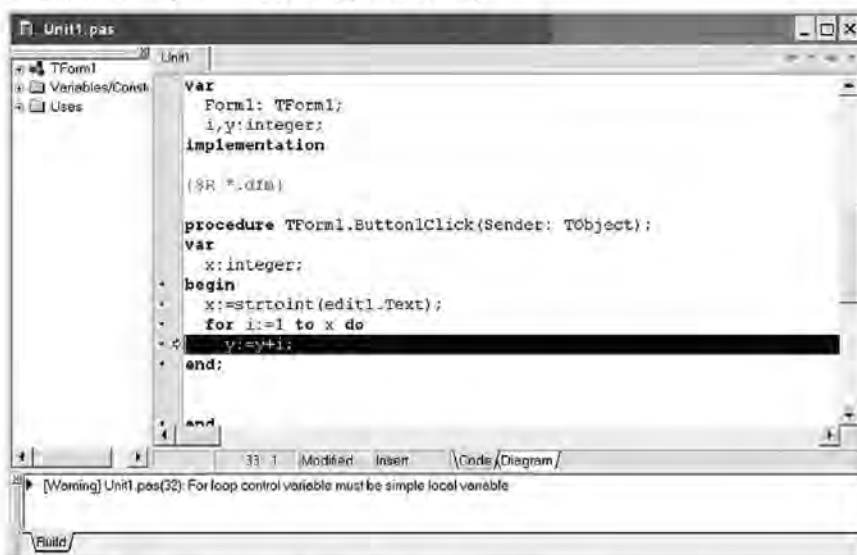


Рис. 6.2. Покрокове виконання програми

Розглянемо потужніший інструмент — введення в додаток точок переривання (breakpoint). Щоб установити точку переривання, досить у вікні редактора коду клацнути мишею на смужці лівіше від коду необхідного рядка. Рядок набуде червоного кольору, і на ньому з'явиться яскрава червона точка (рис. 6.3).

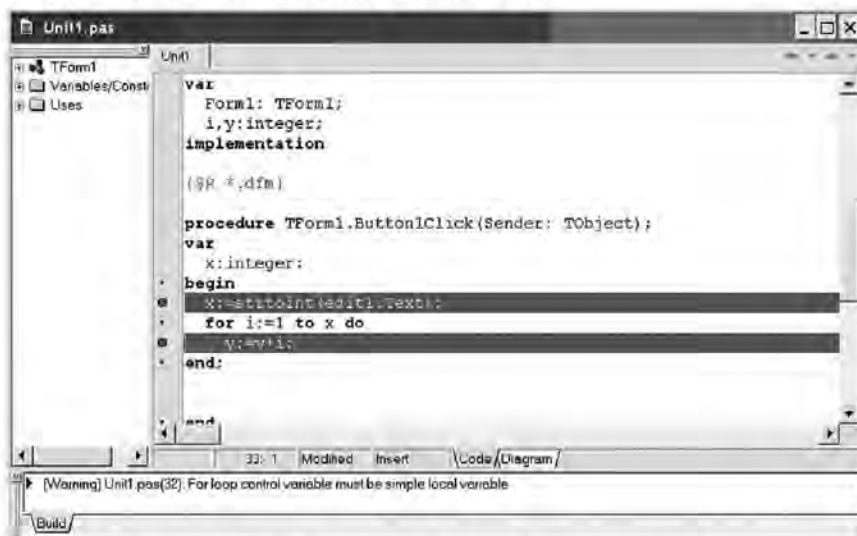


Рис. 6.3. Введення в додаток точок переривання

Якщо тепер ми запустимо додаток на виконання й почнемо з ним працювати, то як тільки керування перейде до рядка, у якому вказано точку переривання, виконання програми перерветься. Таким чином, ми одержали той самий результат, що й у разі описаного раніше виконання програми до точки, вказаної курсором (натискання клавіші F4). Однак перевагою введення точок переривання є те, що одночасно можна вказати кілька точок у різних місцях коду й у різних модулях. Додаток буде виконуватися доти, доки керування не перейде до першої зустрінутої в програмі точки переривання. Наступний запуск приведе до зупинки в другій точці переривання і т. д.

Для того щоб прибрати точку переривання, досить клацнути мишею на червоній точці лівіше від коду відповідного рядка. Точки переривання можна встановлювати тільки на виконуваних операторах. Якщо ви, наприклад, спробуєте встановити точку переривання на рядку, що містить оголошення змінної, то в момент запуску додатка в червоній точці, яка виділяє рядок переривання, з'явиться хрестик. Цим Delphi попереджає, що переривання не буде, оскільки оператор невиконуваний (рис. 6.4).

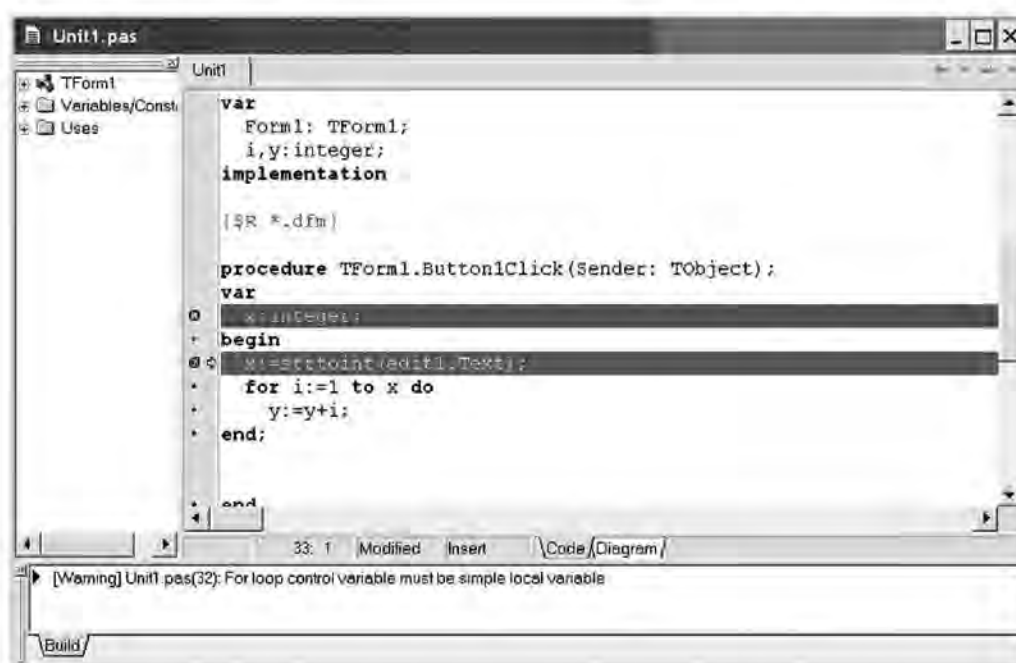


Рис. 6.4. Спроба встановлення точок переривання на виконуваному і невиконуваному операторах

Щоб виправити помилки, які виникають у ході виконання програми, нам часто необхідно бачити водночас значення декількох змінних. Таку можливість дає вікно спостережень Watches.

Зробити його видимим можна за допомогою команди View → Debug Windows → Watches або підвівши курсор у код до потрібної нам змінної й натиснувши одночасно клавіші Ctrl + F5. При цьому вікно спостережень автоматично відкриється, і в ньому з'являться

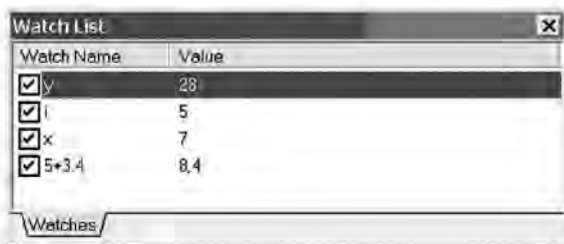


Рис. 6.5. Приклад вікна спостережень Watches

ім'я змінної та її значення (рис. 6.5). Далі можна підвести курсор до іншої змінної й знову натиснути **Ctrl + F5**, після чого у вікні спостережень з'явиться новий рядок. Понад те, ми можемо виділити курсором якийсь вираз, натиснути **Ctrl + F5** і побачити у вікні спостережень значення цього виразу.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Як можна відкомпілювати додаток у Delphi?
2. Наведіть приклади помилок, які можуть виникнути в програмі.
3. Назвіть команди покрокового виконання програми. Перелічіть їхні відмінності.
4. У чому полягає призначення точок переривання?
5. Для чого призначене вікно спостережень Watches? Як його викликати?



Практична робота № 5 Налаштування програм

Мета роботи: опанувати роботу з налагоджувачем програм.

Завдання: використовуючи весь арсенал засобів налагодження програм, розглянутих у попередніх темах, відстежити значення змінних на кожній ітерації роботи циклу програми.

Хід роботи

1. Запустимо середовище Delphi.
2. Створимо новий додаток, помістимо на форму кнопку.
3. Помістимо в обробник події **OnClick** такий код програми:

```
var
    i, x, p: integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
    x:=0;
    p:=1;
    i:=0;
    while i<12 do
```

```

begin
  if i mod 2 = 0 then
    x:=x+i
  else
    p:=p*i;
    i:=i+1;
  end;
  Form1.caption:=inttostr(x)+' '+inttostr(p);
end;

```

4. Для даного фрагмента необхідно зафіксувати значення змінних *i*, *x*, *p* за допомогою вікна спостережень Watches і визначити, що обчислює програма.

Отже, установимо точку переривання, як показано на рис. 6.6.

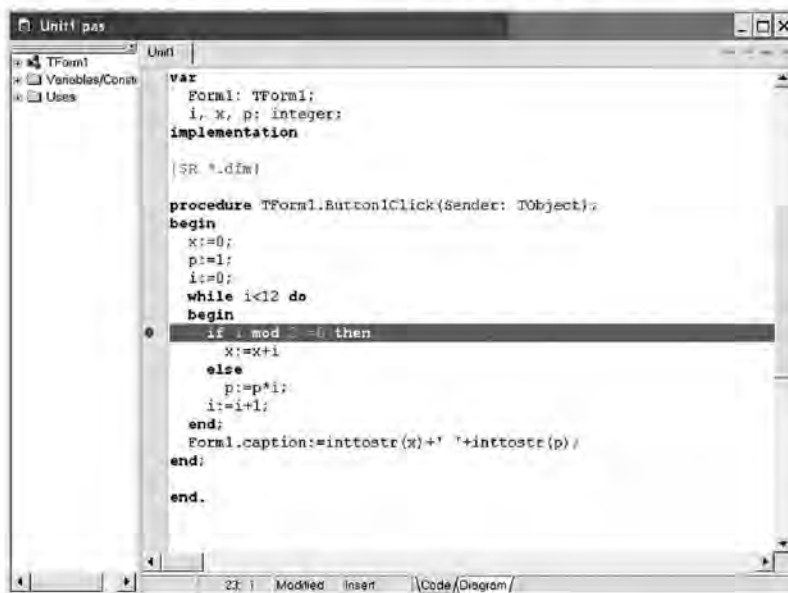


Рис. 6.6. Установлення точки переривання

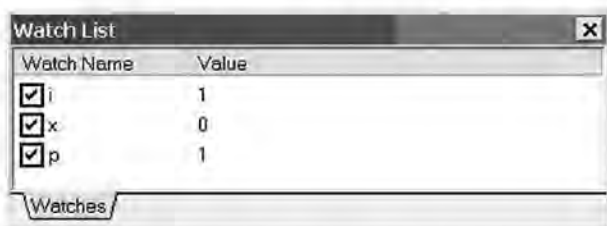
5. За допомогою комбінації клавіш Ctrl + F5 викличемо вікно Watches, як показано на рис. 6.7.



Рис. 6.7. Виклик вікна спостережень Watches

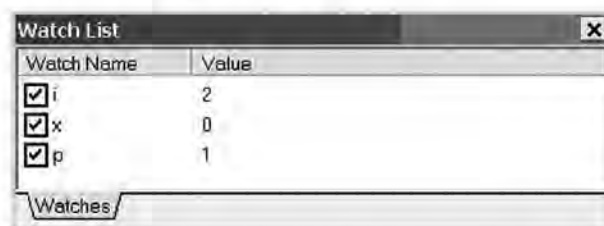
6. Натисканням клавіші F7 (Trace Into) виконаємо першу ітерацію циклу **WHILE**. Легко помітити, що з двох альтернатив, які містяться в умовному операторі **IF**, буде виконано дію **THEN**. Отже, до початкового значення змінної x буде додано значення i ($i=0$), потім змінну i буде збільшено на 1. Стан змінних після першої ітерації виконання циклу показаний на рис. 6.8.

7. Далі, здійснюючи вказані дії, ми бачимо, що програма виконується відповідно до альтернативи **ELSE**, тож домножимо значення змінної p на 1, після чого змінна i збільшується на 1. Після другої ітерації виконання циклу стан змінних зміниться (рис. 6.9).



Watch Name	Value
<input checked="" type="checkbox"/> i	1
<input checked="" type="checkbox"/> x	0
<input checked="" type="checkbox"/> p	1

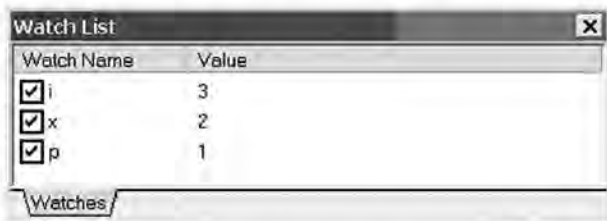
Рис. 6.8. Стан змінних на першому кроці



Watch Name	Value
<input checked="" type="checkbox"/> i	2
<input checked="" type="checkbox"/> x	0
<input checked="" type="checkbox"/> p	1

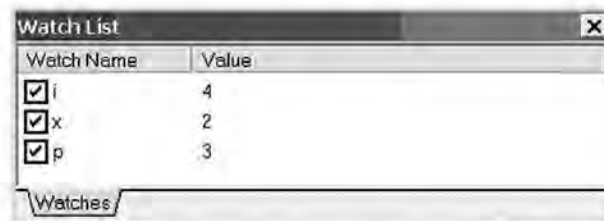
Рис. 6.9. Стан змінних на другому кроці

8. Виконаємо описану вище послідовність дій для третьої й четвертої ітерацій (результати цих дій показано на рис. 6.10, 6.11).



Watch Name	Value
<input checked="" type="checkbox"/> i	3
<input checked="" type="checkbox"/> x	2
<input checked="" type="checkbox"/> p	1

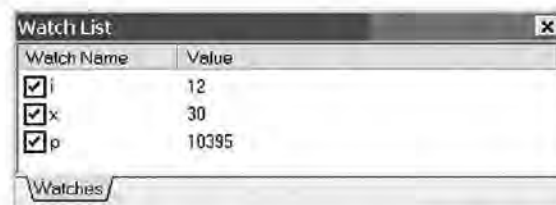
Рис. 6.10. Стан змінних на третьому кроці



Watch Name	Value
<input checked="" type="checkbox"/> i	4
<input checked="" type="checkbox"/> x	2
<input checked="" type="checkbox"/> p	3

Рис. 6.11. Стан змінних на четвертому кроці

9. Продовжимо налаштування програми в покроковому режимі. Після 12-ї ітерації (рис. 6.12) виконання циклу **WHILE** завершиться.



Watch Name	Value
<input checked="" type="checkbox"/> i	12
<input checked="" type="checkbox"/> x	30
<input checked="" type="checkbox"/> p	10395

Рис. 6.12. Стан змінних на 12-му кроці

10. Таким чином, ми з'ясували значення змінних i , x , p на кожному кроці роботи програми. Маємо такий результат: змінна x дорівнює $x=0+0+2+4+6+8+10$, тобто являє собою суму парних чисел, що перебувають у діапазоні від 0 до 10, а значення змінної p дорівнює

$p=1 \cdot 1 \cdot 3 \cdot 5 \cdot 7 \cdot 9 \cdot 11$, що в остаточному підсумку являє собою добуток непарних чисел від 1 до 11.

11. Запишемо в зошит усі проміжні значення змінних на кожному кроці.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. Для поданого фрагмента програми з'ясувати в покроковому режимі значення всіх змінних, що використовуються в програмі.

```
var
  x,y,i: integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
  x:=0;
  i:=10;
  y:=0;
  while i>1 do
  begin
    y:=y + x mod 10 + i;
    x:=x + 4;
    i:=i - 1;
  end;
end;
```

2. Для поданого фрагмента програми з'ясувати в покроковому режимі значення всіх змінних, що використовуються в програмі.

```
var
  x,y,i: integer;
procedure TForm1.Button1Click(Sender: TObject);
begin
  x:=10;
  i:=10;
  y:=0;
  while i>1 do
  begin
    i:=i - 1;
    x:=x - 5;
    y:=y+x div 3;
  end;
end;
```

3. Для поданого фрагмента програми в покроковому режимі з'ясувати значення всіх змінних, що використовуються в програмі.

```
-----  
var  
  x,y,i: integer;  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
  x:=8;  
  i:=2;  
  y:=7;  
  while i<20 do  
  begin  
    i:=i+2;  
    x:=x + x div 10;  
    y:=y+x;  
  end;  
end;  
-----
```



Візуальне середовище дозволяє відносно просто виконати красивий, наділений великими можливостями графічний інтерфейс. Однак мета програми полягає в розрахунках, аналізі, порівнянні й організації зберігання вихідних даних. Для цього доведеться писати відповідні оператори, основною частиною яких стане обчислення виразів — арифметичних, логічних, рядкових. Цей розділ присвячений правилам побудови, обчислення й використання різних виразів.

7.1

Поняття операції та виразу. Пріоритет операцій. Арифметичні операції. Операції над рядками

➔ Поняття операції та виразу. Пріоритет операцій

Вираз — це комбінація символів, яка визначає порядок виконання дій над даними, що вводяться в комп'ютер, і складається з операндів, круглих дужок та знаків операцій. Операнди являють собою константи, змінні й виклики функцій. **Операція** — це дія, що виконується над операндами. Наприклад, у виразі $a * \sin(x)$ пропонується обчислити значення синуса функції від змінної x , а потім помножити результат на a .

Операції в мові Delphi поділяються на арифметичні, операції відношення (порівняння), логічні (булеві), операції над рядками, одержання адреси тощо. Вирази відповідно називаються арифметичними, виразами відношення (порівнянними), логічними (булевими), рядковими тощо залежно від того, якого типу операнди та операції в них використовуються.

У складних виразах порядок обчислення значень позначається дужками, а якщо поспіль ідуть кілька операцій, не розділених дужками, то порядок обчислення визначається пріоритетом (старшинством) і асоціативністю операцій. За пріоритетом операції поділяються на чотири рівні (табл. 7.1).

Таблиця 7.1

Пріоритет операцій у мові Delphi (Object Pascal)

Операція	Пріоритет
@, not	Перший (найвищий)
*, /, div, mod, and, shl, shr, as	Другий
+, -, or, xor	Третій
=, <>, <, >, <=, >=, in, is	Четвертий (найнижчий)

Операції з вищим пріоритетом виконуються раніше, ніж операції з нижчим пріоритетом. Якщо ж поспіль ідуть кілька операцій із рівним пріоритетом, то порядок обчислення визначається асоціативністю,

яка в усіх операцій в Object Pascal однакова — зліва направо. Нижче в табл. 7.2 наведено приклади виконання операцій відповідно до зазначених правил.

Таблиця 7.2

Приклади виконання операцій

Вираз	Порядок обчислення	Пояснення
$a - b + c$	$((a - b) + c)$	Усі операції одного рівня пріоритету
$a * b / c$	$((a * b) / c)$	Усі операції одного рівня пріоритету
$a * b + c / b$	$(a * b) + (c / b)$	Операції $*$ і $/$ мають вищий пріоритет, ніж операція $+$
$a = b \text{ or } c = d$	$(a = (b \text{ or } c)) = d$	Операція or має вищий пріоритет, ніж операція $=$. Після виконання or операції $=$ виконуються зліва направо

➔ Арифметичні операції

Найчастіше в програмах використовуються арифметичні операції; вони виконують арифметичні дії над значеннями операндів цілочислових і дійсних типів даних (табл. 7.3).

Таблиця 7.3

Позначення арифметичних операцій в Delphi

Операція	Дія	Тип операндів	Тип результату
+	Додавання	Цілий, дійсний	Цілий, дійсний
-	Віднімання	Цілий, дійсний	Цілий, дійсний
*	Множення	Цілий, дійсний	Цілий, дійсний
/	Ділення	Цілий, дійсний	Дійсний
div	Цілочислове ділення	Цілий	Цілий
mod	Остача від ділення	Цілий	Цілий

Операції додавання, віднімання та множення відповідають аналогічним операціям у математиці. На відміну від них, операція ділення має три види: звичайне ділення ($/$), цілочислове ділення (**div**), остача від ділення (**mod**). Результат, одержуваний унаслідок використання різних операцій ділення, проілюструємо прикладами:

Вираз	Результат	Вираз	Результат
$-12 / 4$	-3.0	$12 / 5$	2.4
$25 \text{ div } 7$	3	$6 \text{ div } 8$	0
$25 \text{ mod } 7$	4	$6 \text{ mod } 8$	6
$-10 \text{ div } 4$	-2	$20 \text{ mod } 5$	0

Примітка. Логічні (булеві) операції та операції відношення (порівняння) буде розглянуто в наступному розділі.

➔ Операції над рядками

Рядковий тип даних — один із найчастіше використовуваних у візуальних програмах. Багато в чому це зумовлене тим, що більшість відеокомпонентів для введення даних (навіть числових) працюють із рядками.

Рядок — це послідовність символів. У Delphi (Object Pascal) існує кілька типів рядків. Основні з них наведено в табл. 7.4.

Таблиця 7.4

Основні типи рядків Delphi

Тип рядка	Максимальна довжина	Використовувана пам'ять	Сфера застосування
ShortString	255 символів	Від 2 до 256 байт	Погана сумісність, зберігання невеликих рядків, швидка обробка
AnsiString	Близько 2^{31} символів	Від 4 байт до 2 Гбайт	8-бітні символи (ANSI), DBCS ANSI, MBCS ANSI та ін.
WideString	Близько 2^{30} символів	Від 4 байт до 2 Гбайт	Юнікод-символи — багатокористувальські сервери, мультимовні додатки

Для більшості цілей підходить тип `AnsiString`.

Розглянемо основні стандартні функції та процедури оброблення рядків.

1. Функція `Length(Str: String)` повертає кількість символів (довжину рядка). Приклад використання функції:

```
-----
var
  Str: ansistring; L: integer;
begin
  Str:='Happy New Year!';
  L:=Length(Str); {результат L = 15}
end;
-----
```

2. Процедура `SetLength(Str: String; NewLength: Integer)` може зменшувати довжину рядка. Якщо рядок містив більшу кількість символів, ніж задано в другому параметрі процедури, то «зайві» символи обрізаються. (Нагадаємо, що функція повертає обчислене значення у своєму імені й може використовуватись у виразах, а процедура передає результат списком параметрів і використовується як самостійний параметр.) Приклад використання процедури:

```
-----
var Str: ansistring;
begin
  Str:='Happy New Year!';
  SetLength(Str, 5); {результат Str = 'Happy'}
end;
-----
```

3. Функція `Pos(SubStr, Str: String)` повертає позицію підрядка в рядку. Нумерація символів починається з одиниці. У разі відсутності підрядка в рядку функція повертає значення 0. Приклад використання функції:

```
-----
var Str1, Str2: ansistring;
  P: integer;
begin
  Str1:='Happy New Year!';
  Str2:='New';
  P:=Pos(Str2, Str1); {результат P = 7}
end;
```

4. Функція `Copy(Str: String; Start, Length: Integer)` повертає підрядок (частину рядка) починаючи із символу `Start` завдовжки `Length`. Обмежень на параметр `Length` немає, однак якщо числове значення перевищує кількість символів починаючи з позиції `Start` до кінця рядка, то рядок копіюється до самого кінця. Приклад використання функції:

```
-----
var Str1, Str2: ansistring;
begin
  Str1:='Happy New Year!';
  Str2:=Copy(Str1, 7, 3); {результат Str2 = 'New'}
end;
```

5. Процедура `Insert(SubStr: String; Str: String; Pos: Integer)` вставляє в рядок `Str` підрядок `SubStr` у позицію `Pos`; починаючи з позиції вставки літери вихідного рядка розсуваються. Приклад використання процедури:

```
-----
var Str: ansistring;
begin
  Str:='Happy Year!';
  Insert('New ', Str, 7); {результат Str='Happy New Year!'}
end;
```

6. Процедура `Delete(Str: String; Start, Length: Integer)` видаляє з рядка `Str` символи починаючи з позиції `Start` завдовжки `Length` (якщо вказаний параметр виходить за межі рядка, видаляє символи до кінця рядка). Приклад використання процедури:

```
-----
var Str: ansistring;
begin
  Str:='Happy New Year!';
  Delete(Str, 6, 250); {результат Str = 'Happy'}
end;
```

7. Функції `AnsiUpperCase(Str: String)` і `AnsiLowerCase(Str: String)` переводять рядок відповідно у верхній і нижній регістри (широко відомі в літературі функції `UpperCase` і `LowerCase` виконують ті самі дії, але тільки для рядків із латинських букв). Приклад використання функції:

```
-----
var Str1, Str2, Str3: String;
begin
  Str1:='Задача';
  Str2:=AnsiUpperCase(Str1); { Str2 = «ЗАДАЧА» }
  Str3:=AnsiLowerCase(Str1); { Str3 = «задача» }
end;
```

Рядки можна порівнювати між собою звичайними операторами порівняння:

```
-----
var Str1, Str2, Str3: ansistring; logic1, logic2: boolean;
begin
  Str1:='Cat';
  Str2:='Cat';
  Str3:='Dog';
  logic1:=(Str1 = Str3); {результат logic1 = False }
  logic2:=(Str1 = Str2); {результат logic2 = True }
end;
```

Якщо рядки ідентичні, логічний вираз стане `True`.



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Дайте визначення поняття «вираз».
2. Що собою являють операнди та операції?
3. Які типи операцій використовуються в Delphi?
4. У якому порядку за пріоритетом виконуються операції в мові Delphi?
5. Наведіть приклади арифметичних операцій у Delphi.
6. Назвіть основні типи рядків Delphi.
7. Перелічіть основні стандартні функції та процедури оброблення рядків.

Практична робота № 6

Виконання математичних обчислень

Мета роботи: навчитися записувати мовою Delphi математичні вирази, і навпаки, вирази, подані мовою Delphi, записувати в математичному вигляді.

Хід роботи

Завдання 1. Записати вирази, подані мовою Delphi, в математичному вигляді.

$$1) a + (b + c) / b * (a - c) + a * b / c * d - 1 / (b + (a + c) / 2)$$

Розв'язання.

$$a + \frac{(b+c)(a-c)}{b} + \frac{abd}{c} + \frac{1}{b + \frac{a+c}{2}}$$

$$2) ((a+b)+c) / (a-2) + (a*b) / c * d - 1 / (b+a) + c / 2 / a / b$$

$$3) a + (b + c) / a - c + a * b / c * e * (d - 1 / (b + (a + c) / 2))$$

$$4) a + d * (b + 2 - c) / (a - c + a * b / c) * (d - 1 / b / 2))$$

$$5) (a + d) / (b + 2 - c) / (a * c + a * b / c) * (d - 1 / b / 2 * a))$$

$$6) a + d / (b + 2 - c) / (a * c + b / c * (d - 1 / b / 2 * a))$$

Завдання 2. Записати мовою Delphi подані нижче математичні вирази.

$$1) \frac{a+b}{c} + d - \frac{2 + \sqrt{\frac{a}{a-1}}}{b+c} \cdot \frac{cd}{ab}$$

Розв'язання.

$$(a + b) / c + d - (2 + \text{sqrt}(a / (a - 1))) * c * d / (b + c) / a / d$$

$$2) \frac{a+4}{c+2} + d^2 \frac{a}{bcd} - \frac{\sqrt{b} + \sqrt{\frac{a}{a-1}}}{c}$$

$$3) \frac{a^2}{c+d} + d \frac{c^2}{ba} + \frac{a + \sqrt{\frac{a-b}{dc-1}}}{e(b+2)}$$

$$4) \frac{a^2}{c+d} + d \frac{c}{abe} + \frac{a + \sqrt{\frac{a-b}{bc-1}}}{(b+2)c}$$

Завдання 3.

1) Розрахувати вручну, що буде виведено після таких перетворень рядка:

```
-----
var s: shortstring;
begin
  s := 'This is a nice business';
  s := copy(s, 3, 10) + copy(s, 1, 5);
  delete(s, 7, 12); s:= ansiuppercase(s);
  Label1.caption := s;
end;
-----
```

Розв'язання. «Виконуємо» програму оператор за оператором. Після першого оператора в змінній рядкового типу записано 'This is a nice business'. Другий оператор вирізає з вихідного рядка

два підрядки, з'єднує їх і записує в змінну s замість попереднього значення ('is is a ni'+ 'This '= 'is is a niThis '). Наступний оператор стирає в рядку 12 символів починаючи з 7-го. Там стільки символів немає, так що стирається все до кінця рядка. Наступний оператор переводить усі літери у верхній регістр (робить їх великими).

Відповідь: залишається рядок 'IS IS'.

- 2) Розрахувати вручну, що буде виведено після таких перетворень рядка:

```
-----
var s : shortstring;
begin
  s:='This is a fine fix!';
  s:=copy(s, 2, 7) + copy(s, 1, 5);
  delete (s,3,4); s:=ansilowercase(s);
  Label1.caption:=s;
end;
```

- 3) Розрахувати вручну, що буде виведено після таких перетворень рядка:

```
-----
var s : shortstring;
begin
  s:= 'Trubles never come singly';
  s:= copy(s, 3, 6) + copy(s, 1, 5) + copy(s, 2, 9);
  delete (s,8,12); s:=ansiuppercase(s);
  Label1.caption:=s;
end;
```



Одними з найважливіших у програмуванні є умовні оператори. Вони наявні практично в кожній програмі і здійснюють масу дуже важливих перевірок — виводити чи не виводити, використати той чи інший фрагмент алгоритму, повідомляти про помилку чи не повідомляти. Цей розділ присвячений питанням алгебри логіки й логічним операторам, побудованим на її основі.

8.1

Поняття про булеву логіку. Логічні (булеві) операції та операції відношення (порівняння)

Джордж Буль (George Boole), засновник математичної логіки, народився 2 листопада 1815 р. в місті Лінкольні (Велика Британія) у родині небагатого ремісника Джона Буля. Перші уроки математики майбутньому генієві дав його батько. Однак у юні роки Буль не виявив талантів у точних науках — його першим захопленням стала класична література. У 12 років Джон знав латину, потім оволодів грецькою, французькою, німецькою й італійською мовами. Лише в сімнадцять років Буль захопився вищою математикою. У 20 років Дж. Буль відкрив власну школу в Лінкольні. Усе своє життя він працював у школах та коледжах.

Джордж Буль — автор 50 статей на математичні теми й кількох монографій, що стали класичними. Інтереси Буля в математиці були різнобічними, але всесвітню славу йому принесла математична логіка. Можна вважати, що Буль був першим математиком, який звернувся до логічної проблематики.

У своєму дослідженні (1844), опублікованому у «Філософських працях Королівського товариства», Буль уперше пише про «обчислення висловів». 1847 р. він опублікував памфлет «Математичний аналіз логіки», у якому висловив ідею, що логіка ближча до математики, аніж до філософії. Цю роботу дуже високо поцінував шотландський математик Августус де Морган. Завдяки їй Буль у 1849 р. обійняв посаду професора математики Квінз-коледжу в графстві Корк, незважаючи на те що він навіть не мав університетської освіти.

Однак памфлет не міг вважатися серйозною науковою працею, і Буль повною мірою висловлює свої погляди на цю проблему в трактаті «Дослідження законів мислення, на яких ґрунтуються математичні теорії логіки та ймовірностей» (1854).

Одиницею Буль позначав множину всіх мислимих об'єктів, літерними символами — вибірки з неї, пов'язані зі звичайними іменниками й означеннями (так, якщо x = «рогаті», а y = «вівці», послідовна вибірка x і y з одиниці дасть клас рогатих овець). Буль показав: символи

такого роду підпорядковуються тим самим законам, що й алгебраїчні, тож будь-які об'єкти можна додавати, віднімати і множити (так, різниця $1-x$ являє собою операцію вибірки з універсуму всіх безрогих об'єктів, а добуток $(1-x)(1-y)$ — вибірку об'єктів, що не є ані рогатими, ані вівцями).

Роботи Дж. Буля 1847 і 1854 рр. започаткували алгебру логіки — згодом булеву алгебру. Учений, скориставшись двійковою системою числення, придумав свою систему позначень і правил, за допомогою яких можна закодувати будь-які вислови, а потім маніпулювати ними як звичайними числами, передбачаючи тільки два варіанти відповіді — «істина» або «хибність» (у Delphi — True і False), «одиниця» або «нуль». Булева алгебра має три основні операції — І, АБО, НЕ, які дозволяють здійснювати множення, додавання й заперечення логічних умов.

Створюючи мову Pascal, Ніклаус Вірт увів для позначення логічного типу даних зарезервоване слово Boolean на пошану творця математичної логіки. Цікаво, що в першому стандарті мови це слово належало писати з великої літери.

Отже, математична логіка (булева алгебра) оперує логічними змінними, які можуть набувати значень двох видів — true (істина) і false (хибність). У числовому кодуванні «хибність» задається цілочисловим значенням 0, а «істина» — числовим значенням 1 або більше. Формально ми можемо «спитати» в програми: «Чи ІСТИНА більша за ХИБНІСТЬ?» — і одержати ствердну відповідь.

Значення логічним змінним задаються шляхом присвоювання констант або результатів порівняння. Розглянемо приклад:

```
-----
var
  a: integer;
  x,y: Boolean;
begin
  x:=true;
  a:=7;
  y:=a > 2;
-----
```

Обидві логічні змінні (x і y) після виконання цього фрагмента програми набудуть значення true («істина»).

В операціях відношення застосовуються звичні в математиці знаки порівняння: $>$ («більше»), $<$ («менше»), $=$ («дорівнює»), $>=$ («більше або дорівнює»), $<=$ («менше або дорівнює»), $<>$ («не дорівнює»). Три останні знаки відрізняються від звичайного математичного запису. Це зумовлене історичними причинами: старі принтери й алфавітно-цифрові монітори не могли відтворювати знаки \geq , \neq і \leq , тому для них було придумано заміну з комбінацій символів, що виводяться в один рядок.

Як правило, умови порівняння в розв'язуваних задачах набагато складніші, ніж проста перевірка на рівність або нерівність. Навіть

прості задачі часто вимагають одночасного виконання відразу кількох умов. Розглянемо класичний приклад.

Відомо, що Земля обертається навколо Сонця не за 365 діб, а за 365,242199 доби. Якщо календарний рік становить 365 днів, то за 4 роки «набігає» зайва доба (точніше, «майже» доба, бо 0,242199 усе-таки менше за чверть доби). Подолати цю складність досить просто. Творці сонячних календарів установили, що кожен четвертий рік є високосним, тобто складається з 366 днів. Як уже йшлося в розділі про алгоритмічні структури, у Delphi є оператор, що може виконувати ту чи іншу дію залежно від логічної умови. Умова «високосності» року в цьому операторі записується так:

```
-----
year:=2010;
if year mod 4=0 then
  Label1.Caption:='Високосний рік'
else
  Label1.Caption:='Невисокосний рік';
-----
```

Однак такий підхід призводить до поступового накопичення помилок. Тому раз у 400 років один високосний рік прийнято вважати невисокосним. Визначається він так: це рік, що закінчується на два нулі «00», перші дві цифри якого діляться на 4. Тепер нам в умовному операторі слід написати такий логічний вираз: «номер року ділиться на 4 без остачі, і при цьому, якщо він ділиться без остачі на 100 (тобто закінчується на «00»), число його сотень не ділиться на 4». Умова занадто громіздка, трохи простіший вигляд матиме обернений варіант — умова «невисокосності» року. Записується вона так:

```
-----
year:=2010;
if (year mod 4<>0) or
  (year mod 100=0)and((year div 100) mod 4<>0) then
  Label1.Caption:='Невисокосний рік'
else
  Label1.Caption:='Високосний рік';
-----
```

Застосовані у виразах логічні операції **or** (АБО) та **and** (І) ввів у математичну логіку ще Буль.

У всіх мовах програмування реалізуються щонайменше три логічні операції:

- логічне заперечення — **not**, заміна «істини» на «хибність» і навпаки (наприклад, $a:=10$; $x:=\text{not}(a>2)$; після виконання цих операторів x дорівнюватиме false);
- логічне множення — **and**, вимагає одночасного виконання обох умов (наприклад, $a:=10$; $x:=(a>2)\text{and}(a<12)$; дає в результаті значення x , що дорівнює true; у випадку, якщо $a:=14$, x дорівнюватиме false);

- логічне додавання — **or**, вимагає виконання хоч однієї з двох умов (наприклад, $a := 5$; $x := (a < 7) \text{ or } (a > 10)$; дає в результаті значення x , що дорівнює **true**, хоч умова $a > 10$ тут не виконується);
- «виключне АБО» — **xor**, дає в результаті «істину», якщо значення операндів збігаються («істина» і «істина», «хибність» і «хибність»).

У математиці значення логічних операцій прийнято подавати у вигляді таблиць істинності, які виводять результати логічних операцій для всіх можливих значень аргументів (табл. 8.1).

Таблиця 8.1

Таблиці істинності для логічних операцій, наявних у Delphi

Змінна x	False	False	True	True
Змінна y	False	True	False	True
not x	True	True	False	False
not y	True	False	True	False
x or y	False	True	True	True
x and y	False	False	False	True
x xor y	True	False	False	True



ЗАВДАННЯ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

Внести в табл. 8.2 значення True або False згідно зі зразком (як це зроблено в першому рядку).

Таблиця 8.2

Результати логічних операцій

Значення змінної a	1	3	5	7	9
Значення змінної b	10	-1	2	20	7
$(a \bmod 2 = 0) \text{ or } (b < 10)$	False	True	True	False	True
$((a \bmod 2) = (a \div 2)) \text{ or } (b > 0)$					
$(\text{not } (a > 2)) \text{ and } (b < 11)$					
$(a \leq 2) \text{ or } (a > 7) \text{ or } (b > a)$					
$(a \geq 5) \text{ and } (b \bmod 6 > 2)$					
$(a * b > 10) \text{ and } (a > b)$					
$(a > 2 * b) \text{ or } (b > a * 2)$					




ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. У чому полягають досягнення математика Дж. Буля?
2. Якими змінними оперує математична логіка?
3. Перелічіть можливі операції відношення (порівняння).
4. Назвіть чотири основні логічні операції.
5. Що більше — True чи False?

8.2

Формування умов

Умови для виконання тієї чи іншої дії можуть формуватися шляхом порівняння обраних значень, як результат обчислення логічних виразів або шляхом опитування відповідних відеокомпонентів. Найбільш простим і поширеним компонентом для введення умов одно- або двоальтернативного розгалуження є компонент CheckBox , розташований на закладці Standard.

На формі цей компонент має вигляд квадратної комірки з підписом; клацанням мишею в цю комірку можна помістити відповідну позначку (за замовчуванням — «галочку»). У процесі виконання програми наявність або відсутність позначки в елементі CheckBox перевіряється (а в разі потреби встановлюється) властивістю Checked.

Розглянемо простий приклад.

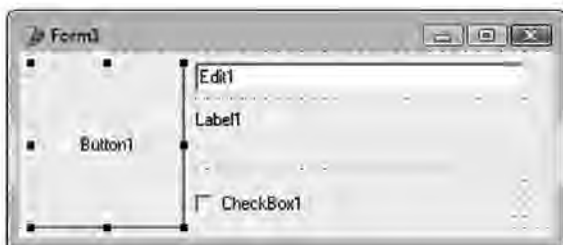


Рис. 8.1. Вигляд форми, одержаний у процесі візуального проектування

Приклад 5. На формі в Edit вводиться довжина сторони квадрата. Натисканням кнопки в Label виводиться його площа. Якщо в розташованому на формі компоненті CheckBox стоїть позначка, до числового значення в Label приписується напис «кв. м» (рис. 8.1).

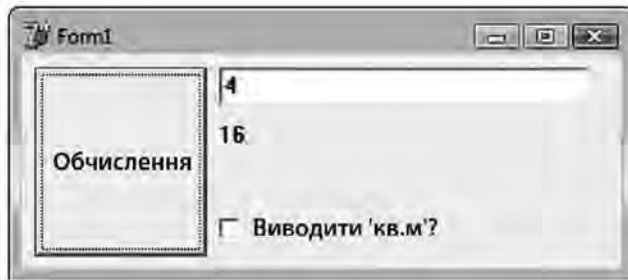
У процесі готування завдання настроїмо відеокомпоненти: замінимо рядки Caption на кнопки й компоненті CheckBox на значущі написи. У Label рядок Caption приберемо, у компоненті Edit очистимо властивість Text (рядок, що вводиться, — за замовчуванням там назва компонента).

Подвійним клацанням на кнопці перейдемо в режим введення програмного коду в метод TForm1.Button1Click, тобто запишемо фрагмент коду, що буде виконуватися в разі натискання кнопки:

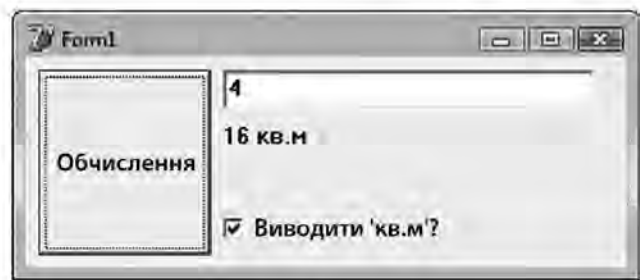
```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Caption:=floattostr(sqr(strtfloat(edit1.Text)));
    if CheckBox1.Checked then
        Label1.Caption:=Label1.Caption+'кв.м';
end;
-----
```

Коли запустити програму на виконання, вона працюватиме так, як показано на рис. 8.2.

Зверніть увагу: для зручності читання написи в усіх компонентів виконані напівжирним шрифтом (властивість Font.Bold має значення



а



б

Рис. 8.2. Робота програми з непозначеним (а) і позначеним (б) CheckBox

True). Програму можна вдосконалити, додавши ще один CheckBox для перемикання звичайного і напівжирного шрифтів (це можна зробити за допомогою окремої кнопки, а можна додати програмний код безпосередньо в кнопку «Обчислення»):

```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    Label1.Caption:=floattostr(sqr(strtfloat(edit1.Text)));
    if CheckBox1.Checked then
        Label1.Caption:=Label1.Caption+' кв. м';
    if CheckBox2.Checked then
        begin
            Button1.Font.Style:=[fsBold];
            Label1.Font.Style:=[fsBold];
            Edit1.Font.Style:=[fsBold];
            CheckBox1.Font.Style:=[fsBold];
            CheckBox2.Font.Style:=[fsBold];
        end
    else
        begin
            Button1.Font.Style:=[];
            Label1.Font.Style:=[];
            Edit1.Font.Style:=[];
            CheckBox1.Font.Style:=[];
            CheckBox2.Font.Style:=[];
        end;
    end;
end;
-----
```

Примітка. І перша, і друга програми не здійснюють перевірки на відсутність числового значення в компоненті Edit. Якщо, на ваш погляд, таку перевірку треба передбачити, напишіть цей оператор IF самі.

Коли запустити програму на виконання, вона тепер працюватиме, як показано на рис. 8.3.

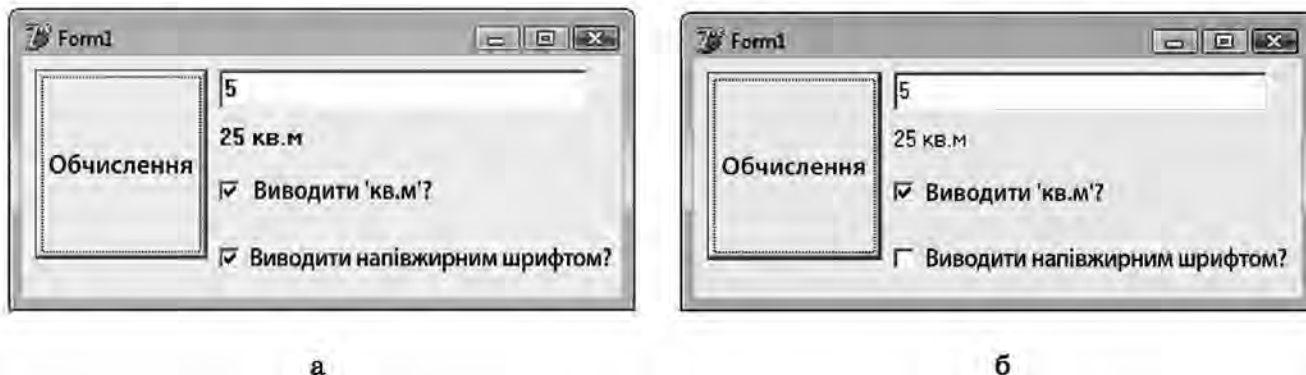


Рис. 8.3. Виконання програми з позначкою в CheckBox «Виводити напівжирним шрифтом» (а) і без позначки (б).



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Як можуть формуватись умови для виконання якої-небудь дії?
2. Для чого призначений компонент CheckBox?
3. Назвіть основну властивість компонента CheckBox. Як вона використовується?
4. Назвіть властивість компонента CheckBox, що змінюється в разі позначення елемента хрестиком. Яке значення (True чи False) буде записане в цю властивість?
5. Яка властивість компонента CheckBox відповідає за виведення тестового запитання поряд із діалоговим елементом?
6. Опишіть, які властивості слід настроїти, щоб написи в Label і CheckBox виконувалися більшим напівжирним шрифтом.

8.3

Алгоритмічні конструкції одно-, дво- і багатоальтернативних розгалужень

У низці випадків одноальтернативне розгалуження («робити — не робити») може виявитися недостатнім: в алгоритмі може бути поставлене завдання вибору однієї з двох альтернатив. Подібні алгоритмічні структури реалізуються двоальтернативним оператором IF, що має таку структуру:

```

-----
if логічний вираз then
    оператор
else
    оператор;
-----

```

Тут логічний вираз — це булева змінна, константа або логічний вираз, який являє собою операцію або кілька операцій порівняння, об'єднаних операторами булевої алгебри.

Оператори, що стоять після ключових слів **THEN** і **ELSE**, можуть бути простими операторами або складеними (тобто групою операторів, об'єднаною операторними дужками **begin** і **end**).

Як приклад наведемо фрагмент програми, що обчислює корінь квадратного рівняння після перевірки значення дискримінанта (змінні a , b , c — коефіцієнти квадратного рівняння, x_1 і x_2 — його корені; усі використані змінні дійсного типу, вони описані на початку процедури або в розділі інтерфейсних змінних).

```

-----
  if  $b*b - 4*a*c >= 0$  then
  begin
    Label1.caption:=FloatToStr((-b+ sqrt( $b*b - 4*a*c$ ))/2/a);
    Label2.caption:=FloatToStr((-b+ sqrt( $b*b - 4*a*c$ ))/2/a);
  end
  else
  begin
    Label1.caption := 'Немає дійсних коренів';
    Label2.caption := '';
  end;
-----

```

Іноді в алгоритмах недостатньо й двох альтернатив, тоді використовується оператор багатоальтернативного розгалуження **CASE**, який має таку структуру:

```

-----
  case перемикач of
    значення 1 перемикача: оператор;
    значення 2 перемикача: оператор;
    ...
  end;
-----
  або
-----
  case перемикач of
    значення 1 перемикача: оператор;
    значення 2 перемикача: оператор;
    ...
  else
    оператор
  end;
-----

```

Вираз-перемикач у цьому операторі може бути змінною або виразом перелічного (порядкового) типу, наприклад цілого, літерного. Тип перемикача має збігатися з типом його значень, які стоять перед двокрапкою в альтернативах оператора **CASE**. Частина **ELSE** не обов'язкова — вона вписується в оператор у випадку, коли значення виразу-перемикача не збігається з жодним із зарезервованих значень. Таким чином, якщо вираз-перемикач не дорівнює жодному з цих значень і є частина **ELSE**,

то виконується оператор, що йде за ELSE. Якщо ж частину ELSE в оператор CASE не включено, то CASE не виконує ніяких дій.

Оператори, що стоять після двокрапки в альтернативах CASE (так само як і в операторі IF), можуть бути простими або складеними (тобто групою операторів, об'єднаних операторними дужками **begin i end**).

Розглянемо простий приклад. На формі містяться кнопка, компонент для введення тексту Edit1 і рядок виведення Label1. Натисканням кнопки потрібно ввести номер місяця невисокосного року й вивести в Label1 кількість днів у цьому місяці.

Метод натискання кнопки матиме такий вигляд:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
var
    mesyac: integer;
begin
    mesyac:=strtoint(edit1.Text);
    case mesyac of
        1,3,5,7,8,10,12: label1.caption:='31';
        4,6,9,11: label1.caption:='30';
        2:label1.caption:='28';
    else
        label1.caption:='Введено неможливий номер місяця'
    end;
end;
-----
```

У цьому прикладі певним значенням змінної-перемикача *mesyac* відповідають однакові дії. Щоб не повторювати однакові оператори при різних альтернативах, у деяких рядках значення перемикача просто перераховані через кому. Замість зазначених операторів можуть стояти інші оператори розгалуження. Наприклад, модифікуємо наведений вище фрагмент для будь-якого року — як високосного, так і невисокосного. Нагадаємо, що високосним є рік, номер якого без остачі ділиться на 4 і не має в числовому записі двох останніх нулів.

Тепер на формі в нас з'явиться ще один компонент для введення тексту Edit2 (для введення номера року), а метод обробника кнопки зміниться так:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
var
    mesyac, god: integer;
begin
    mesyac:=strtoint(edit1.Text);
    god:=strtoint(edit2.Text);
    case mesyac of
        1,3,5,7,8,10,12: label1.caption:='31';
        4,6,9,11: label1.caption:='30';
    end;
end;
-----
```

```

2: if (god mod 4 <> 0) or (god mod 100 = 0) then
    label1.caption:='28'
else
    label1.caption:='29';
else
    label1.caption:='Введено неможливий номер місяця'
end;
end;

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Якими є особливості використання оператора IF?
2. Як вставити в оператор IF більш ніж один оператор?
3. Яке призначення має оператор багатоальтернативного розгалуження CASE? У яких випадках його використовують?
4. Нарисуйте структурну схему для реалізації трьох альтернатив.
5. Наведіть приклади використання чотирьох альтернатив за допомогою операторів IF і CASE.

8.4

Виконання програм із розгалуженнями в покроковому режимі. Вкладені оператори розгалуження

Розглянемо виконання умовних операторів у програмуванні тесту на фреймах.

Приклад 6. Створюємо основну форму і два фрейми (створення фрейму New → Frame) — їх наведено на рис. 8.4.

Виносимо на форму компонент Frames, у діалоговому вікні обираємо Frame2 і Frame3 (у них наскрізна нумерація з формами, тому якби в нас були дві форми, то фрейми одержали б номери 3 і 4). Потрапивши на форму, фрейми змінюють імена: вони стають Frame21 і Frame31 (тобто 2-й фрейм на 1-й формі і 3-й фрейм на 1-й формі).

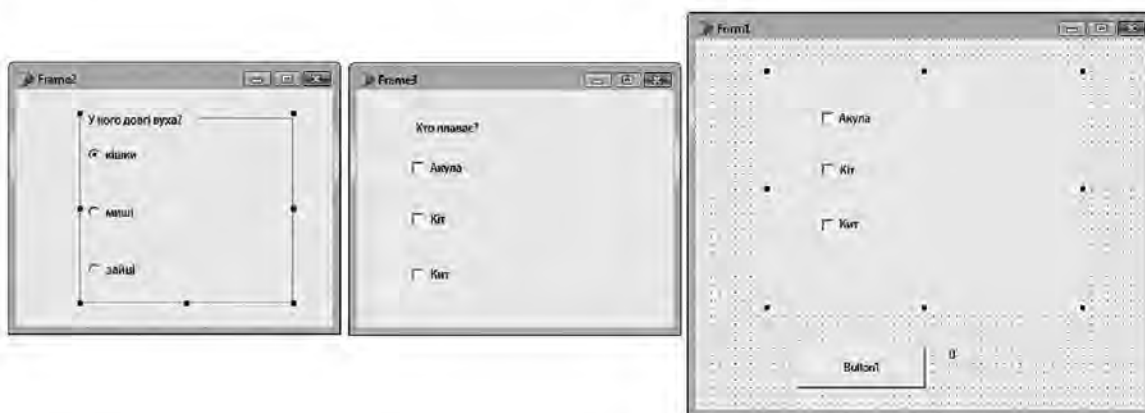


Рис. 8.4. Два фрейми та основна форма з фреймами

Тепер програмуємо тест у методі натискання кнопки:

```

-----
var
  Form1: TForm1;
  n: integer=0;

implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
begin
  if frame21.Visible then
  begin
    if frame21.radiogroup1.itemindex=2 then
      n:=n+1;
    frame21.Visible:=false;
    frame31.Visible:=true;
  end
  else
  begin
    if frame31.checkbox1.checked and
frame31.checkbox3.checked then
      n:=n+1;
    frame31.Visible:=false;
  end;
  Label1.Caption:=inttostr(n);
end;
-----

```

Необхідно звернути увагу на присвоювання значення типізованим константам. Для того щоб це було дозволено, потрібно поставити «галочку» в настройках Project → Options → Assignable typed constant. Ще одна особливість програми — компоненти фреймів із форми «не видно», їх слід іменувати на повне ім'я, наприклад: frame21.radiogroup1.itemindex.

Розглянемо приклад використання фреймів та умовних операторів.

Приклад 7. Готуємо фрейм, поміщуємо в нього Image. Виносимо на форму компонент frames, вставляємо в нього підготований фрейм (рис. 8.5). Оскільки він менший за форму, автоматично з'являються скролери (властивість AutoScroll можна вимкнути — false).

Зображення у віконці можна переміщувати програмно. Наприклад, розташуємо на формі кнопки «Ліворуч», «Праворуч», «Угору» й «Униз». Так, кнопка «Ліворуч» зсуває рисунок на 5 пікселів:



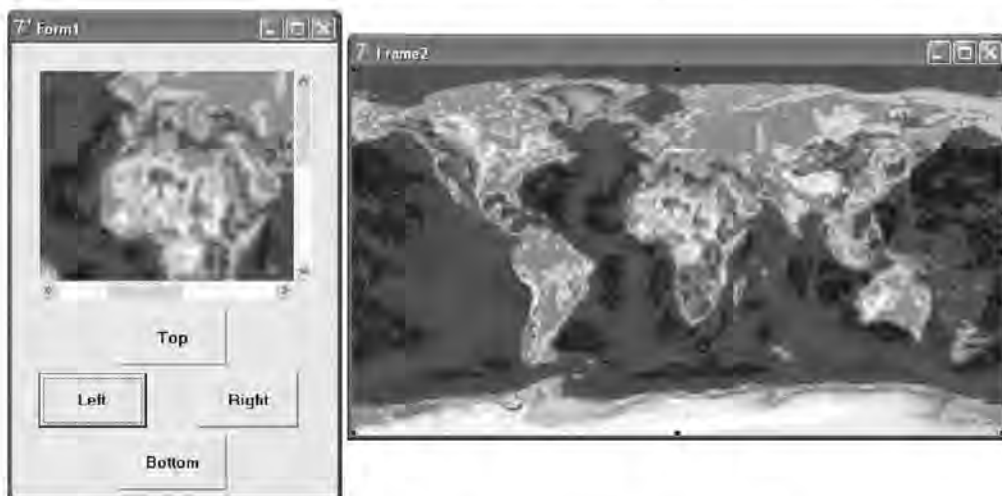


Рис. 8.5. Перегляд малюнка за допомогою фрейму «у віконці»

```

-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    frame21.image1.Left:=frame21.image1.Left-5;
end;
-----

```

Використовуючи подібний підхід, можна реалізувати переміщення і в інших напрямках. Така програма має певні недоліки: натискаючи кнопки, можна «засунути» рисунок за край вікна. Це можна виправити кількома умовними операторами, встановлюючи «неприступність» кнопок. Наприклад, під час руху малюнка праворуч виконується така програма:

```

-----
    frame21.image1.Left:=frame21.image1.Left+5;
    if frame21.image1.Left>0 then
        Button2.Enabled:=false;
        Button1.Enabled:=true;
-----

```

Однак тепер не забудьте «ввімкнути» потрібну кнопку, коли рисунок відсувається від краю,— «ліва» кнопка вмикає «праву», і навпаки. У наведеному фрагменті програми кнопка Button2 робить доступною Button1 (рядок Button1.Enabled:=true;).

Повні умови зміщення малюнка в трьох інших кнопках вам належить записати самостійно (відмінності там будуть у властивостях top і left та в напрямках руху).



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. У чому полягає особливість використання frame?
2. Які дії необхідно виконати для можливості присвоїти значення типізованих констант?
3. Як змістити рисунок праворуч на 5 пікселів?
4. Як змістити рисунок угору на 7 пікселів?

Практична робота № 7

Складання програм з одноальтернативними розгалуженнями

Мета роботи: навчитися визначати області допустимих значень, заданих програмно або за допомогою структурних схем алгоритмів.

Хід роботи

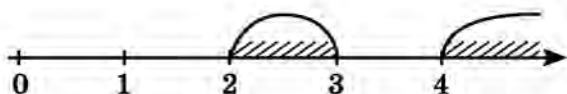


Рис. 8.6. Область допустимих значень змінної x

Завдання 1. На рис. 8.6 подано заштриховану область значень, яких може набувати змінна x на числовій осі. Для цієї області записати умовний оператор мовою Delphi.

Розв'язання. Запустимо середовище Delphi. Введемо значення x . В обробник кнопки OnClick впишемо такий код:

```
-----
if ((x>=2) and (x<=3)) or (x>=4) then
    Label1.Caption := 'Належить'
else
    Label1.Caption := 'Не належить';
-----
```

Залежно від поточного значення x введемо в Label1 відповідь, чи належить x до області допустимих значень. Умова $(x \geq 2)$ and $(x \leq 3)$ — відрізок на осі від 2 до 3, включаючи межі (самі числа 2 і 3). Частина виразу or $(x \geq 4)$ додає до множини значень частину осі правіше від точки $x = 4$.

Завдання 2. Для областей, показаних на рис. 8.7, самостійно записати умови порівняння мовою Delphi.

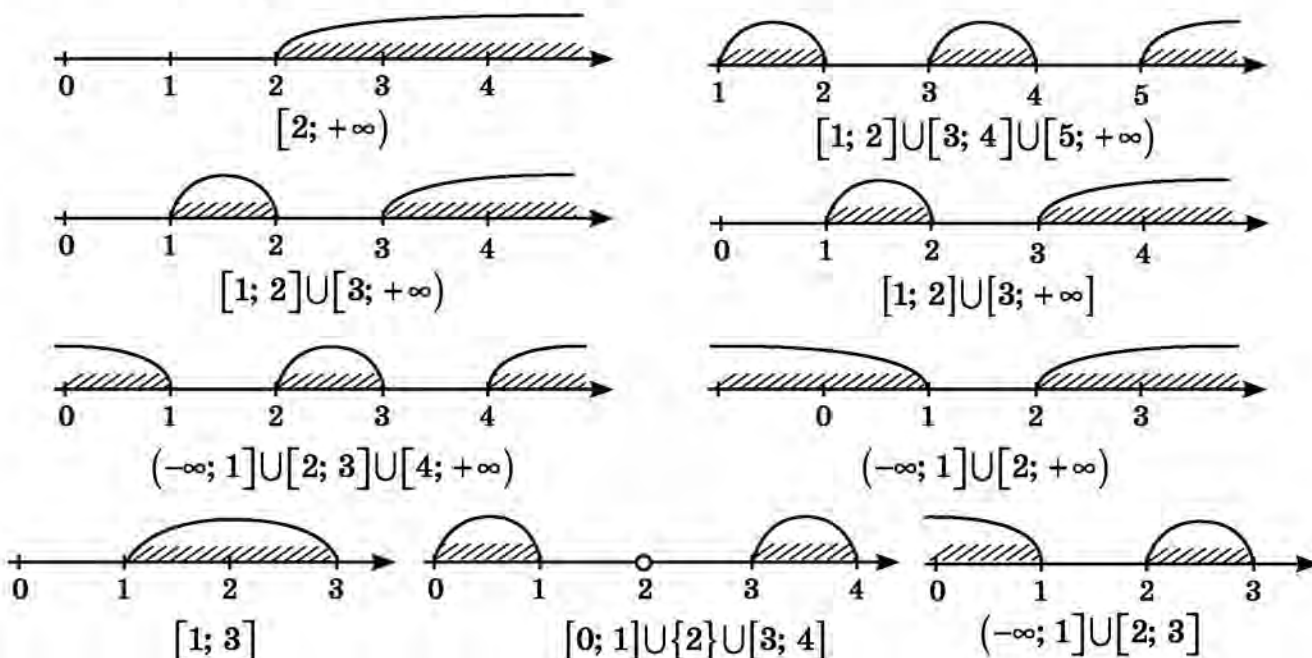


Рис. 8.7. Області обмежень

Завдання 3. Зобразити на числовій осі область обмежень, що відповідає логічним виразам в операторі IF:

`if (x>=2) and (x<=7) or (x=8) then ...`

Розв'язок задачі подано на рис. 8.8.



Рис. 8.8. Область обмежень за даним оператором IF

Завдання 4. Для подання логічних виразів в операторі IF самостійно зобразити області допустимих значень:

- 1) `if (x>=2) and ((x<=6) and (x<=7) then ...`
- 2) `if ((x>=2) and ((x<=6) or ((x>=3) and (x<=7))) then ...`
- 3) `if (x=2) or (x=4) then ...`
- 4) `if ((x>=2) and (x<=3)) or ((x>=4) and (x<=5)) then ...`
- 5) `if (x>=2) and (x<=3) and (x>=3) and (x<=4) then ...`
- 6) `if (x<=4) and (x>=2) and (x<=3) then ...`

Завдання 5. Передбачається, що в програмі є чотири змінні: *a*, *b*, *c*, *d*. З них три рівні між собою, а одна менша за решту. Визначити, яка змінна відрізняється від інших.

Розв'язання. Наведений фрагмент виводить порядковий номер меншої змінної:

```

-----
if (a+b) < (c+d) then
    if a<b then
        Label1.Caption := '1'
    else
        Label1.Caption := '2'
else
    if c<d then
        Label1.Caption := '3'
    else
        Label1.Caption := '4';
-----

```

У наведеному фрагменті немає перевірки на коректність введення чисел, тому цілком закономірно виникає питання: а що виведе цей фрагмент програми, якщо вихідні дані будуть такими:

- | | |
|--|--|
| 1) усі чотири числа однакові; | 3) <i>a</i> =1, <i>b</i> =2, <i>c</i> =3, <i>d</i> =4; |
| 2) <i>a</i> =1, <i>b</i> =1, <i>c</i> =2, <i>d</i> =2; | 4) <i>a</i> =1, <i>b</i> =3, <i>c</i> =2, <i>d</i> =2? |

Завдання 6. Записати логічний вираз, який дає значення «істина» (true) для таких умов: «усі парні значення, які більші або дорівнюють 6».

Розв'язання. `if (x mod 2 = 0) and (x>=6) then ...`

Зберегти на диску проекти, що відповідають областям обмежень. Заштрихувати в зошиті області обмежень для програмних умов.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

- Записати умови, що визначають значення «істина» для:
 - непарних значень x , які не дорівнюють 5;
 - значень x , що більші або дорівнюють 20 і діляться без остачі на 7;
 - значень x , запис яких закінчується на «3»;
 - парних невід'ємних значень x ;
 - значень x , які менші або дорівнюють 2, або x , що дорівнюють 7;
 - парних значень x , що більші або дорівнюють останній цифрі числового запису x .

- Чи відрізняються результати виконання двох поданих нижче фрагментів програм?

```

1) x:=4;
   if x>3 then
     x:=x + 3
   else
     begin
       x:=x*2;
       x:=x - 7;
     end;

```

```

2) x:=4;
   if x>3 then
     x:=x + 3
   else
     x:=x*2;
     x:=x - 7;

```

- Чи відрізнятимуться результати виконання двох поданих нижче фрагментів програм, якщо $x=2$?

```

1) if x>3 then
     x:=x*2
   else
     x:=x+1;

```

```

2) if x>3 then
     x:=x*2
     x:=x+1;

```

Проведіть аналогічне порівняння для $x=4$.

- Чи однаковий результат виведуть два подані нижче фрагменти програм для різних значень x ?

```

1) if x>2 then
     begin
       if x<4 then
         y:=1
       else
         y:=2;
     end;

```

```

2) if x>2 then
     begin
       if x<4 then
         y:=1
       end
     else
       y:=2;

```

- Записати оператор IF, що відповідає трьом фрагментам 1–3 структурних схем алгоритму, наведеним на рис 8.9.

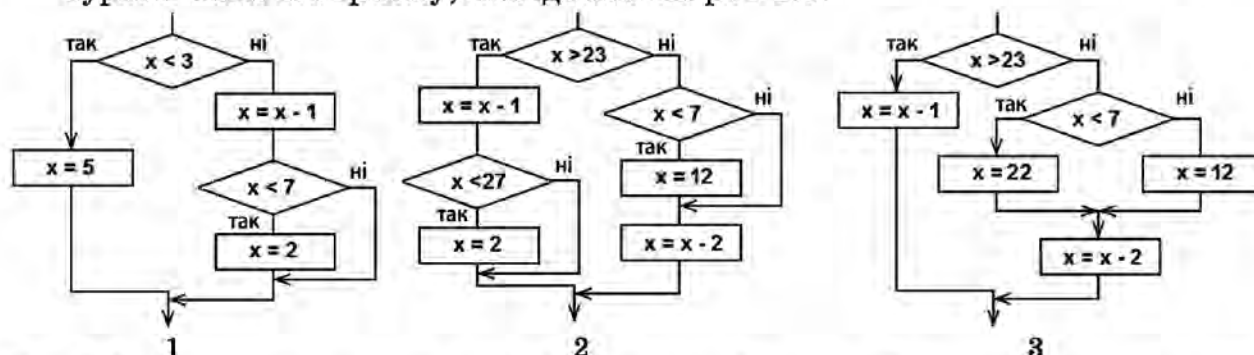
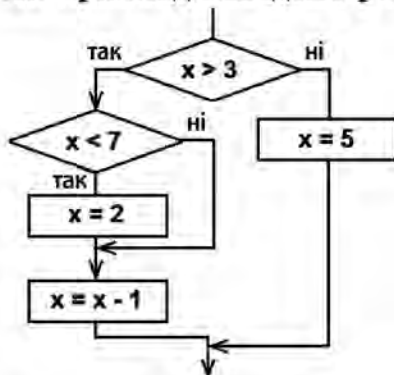


Рис. 8.9. Фрагменти алгоритму для їх програмної реалізації за допомогою оператора IF

Як приклад наведемо розв'язання аналогічної задачі (рис. 8.10).



```

if x > 3 then
begin
  if x < 7 then
    x := 2;
    x := x - 1;
  end
else
  x := 5;
end
  
```

Рис. 8.10. Фрагмент алгоритма

6. Яка область відповідає істинному значенню логічного виразу в записаному операторі IF (рис. 8.11)?

$\text{if } (x * x < 4) \text{ and } (\text{sqr}(y - 1) < 4) \text{ then...}$

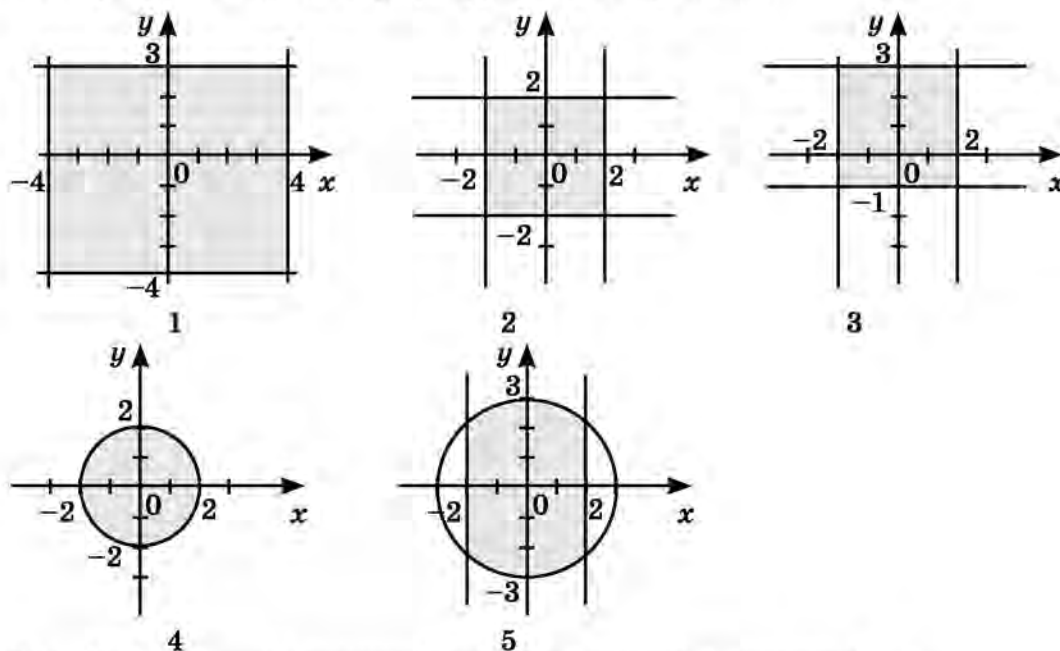


Рис. 8.11. Области істинності логічного виразу в операторі IF

Практична робота № 8

Складання програм з багатоальтернативним розгалуженням

Мета роботи: навчитися складати програми з багатоальтернативними умовами.

Завдання. На формі розміщені компонент для введення тексту Edit1, кнопка й рядок виведення Label1. В Edit1 вводиться додатне ціле число в інтервалі від 1 до 10. Передбачено, що це можлива кількість птахів. Потрібно вивести в Label1 введене число з рядком «птах», «птахи» або «птахів» (залежно від введеного значення).

Хід роботи

1. Продумаємо алгоритм. Згідно з мовними правилами для числа 1 слід написати слово «птах», для 2, 3 і 4 — «птахи», для 5, 6, 7, 8, 9, 10 — «птахів».

2. Запустимо Delphi. Створимо новий додаток.
3. Розмістимо на формі компоненти Edit, Label і Button.
4. В обробник події OnClick кнопки (Button) запишемо код:

```

-----
procedure TForm1.Button1Click(Sender: TObject);
var
    k:integer;
begin
    k:= strtoint(edit1.Text);
    case k of
        1: label1.caption:= inttostr(k)+' птах';
        2,3,4: label1.caption := inttostr(k)+' птахи';
        5,6,7,8,9,10: label1.caption := inttostr(k)+' птахів';
    end;
end;
-----

```

5. Задача стає набагато цікавішою, якщо зняти обмеження на можливу кількість птахів (хай тільки введене число вкладається в цілочисловий тип даних). У такому випадку алгоритм складнішає. Розглянемо, як змінюються відмінкові закінчення слова «птах» зі зростанням числового значення. Наприклад, якщо взяти за основу останню цифру в числах 1, 21, 31, 41 і т. д. (остачу від ділення на 10), то скрізь слід написати слово «птах». Для чисел же 11, 111, 211, 311 слід писати «птахів». Аналогічна неузгодженість існує й для чисел, що закінчуються на 12, 13 і 14.

Із цієї причини програма в кнопці матиме такий вигляд:

```

-----
procedure TForm1.Button1Click(Sender: TObject);
var
    k:integer;
begin
    k:= strtoint(edit1.Text);
    if (k mod 100>=11) and (k mod 100<=14) then
        label1.caption := inttostr(k)+' птахів'
    else
        case k mod 10 of
            1: label1.caption:= inttostr(k)+' птах';
            2,3,4: label1.caption := inttostr(k)+' птахи';
            5,6,7,8,9,10: label1.caption := inttostr(k)+
                ' птахів';
        end;
    end;
-----

```

6. Збережемо проект у своїй папці на диску.



Основною перевагою програмувальної обчислювальної техніки є можливість організовувати повторювані розрахунки для різних значень вхідних аргументів. Для організації багаторазових обчислень у мовах програмування використовуються оператори циклів. У цьому розділі ви дізнаєтеся, яких видів бувають оператори циклів, як за їх допомогою знайти суму або добуток послідовності чисел, як визначити числовим методом значення максимуму або мінімуму.

9.1

Алгоритмічна конструкція повторення та її різновиди: визначені та невизначені цикли, цикли з передумовою і з постумовою. Оператори циклів у мові програмування

Запис послідовних команд не завжди може бути ефективним способом розв'язування поставленого завдання. Дуже часто в програмі потрібно виконати одну й ту саму послідовність дій кілька разів. Наприклад, потрібно вивести на формі числа від 1 до 10. Один зі способів — написати десять рядків коду. Однак таке розв'язання є найгіршим. Звичайно ж, краще використати цикли. *Цикл* — це спеціальна конструкція мови, що дозволяє запрограмувати багаторазове виконання певного блоку команд. Кожний «прохід» циклу називається ітерацією. У Delphi існують три типи циклів: *цикл із параметром* (цикл зі змінною, значення якої збільшується або зменшується задане число разів); *цикл із передумовою* (умова передуватиме виконуваним операторам, і якщо вона не виконується, то цикл завершується), *цикл із постумовою* (цикл виконується принаймні один раз, але може й більше; після кожного виконання перевіряється умова виходу з циклу).

➔ **Цикл із параметром**

Цикл із параметром дозволяє виконати набір команд фіксоване число разів, тобто число ітерацій має бути відоме до початку виконання циклу. Особливістю такого циклу є те, що вводиться спеціальна змінна-лічильник, яка послідовно проходить указаний діапазон значень. Значення цієї змінної може бути використане в блоці коду, що міститься в циклі; вона має бути локальною (описаною всередині модуля процедури або функції, у якій написано цикл) і не може змінюватися за допомогою операторів присвоювання всередині циклу.

Цикл із параметром описується зарезервованим словом FOR (англ. «для»). Загальний вигляд цієї конструкції такий:

FOR лічильник:=початк_значення [TO/DOWNTO] кінц_значення DO
 {дії}

Лічильник — оголошена вище змінна перелічуваного типу (здебільшого ціле число).

Початкове значення і кінцеве значення — межі діапазону, який послідовно проходить змінна-лічильник. Ці значення, природно, належать до того самого типу даних, що й змінна-лічильник.

Залежно від співвідношення початкового і кінцевого значень використовується або ключове слово **TO**, або **DOWNTO**. Слово **TO** застосовується тоді, коли кінцеве значення більше за початкове, а **DOWNTO** — у протилежному випадку (цикл має спадати).

Як дії зазвичай вказується якась команда або набір команд. Якщо команд кілька, їх, як заведено, слід брати в операторні дужки **begin ... end**.

Розглянемо на прикладі використання циклу з параметром.

Приклад 8. Вивести на формі фрагмент таблиці множення для значення, що вводиться. Для розв'язання завдання використати цикл **FOR**.

Розв'язання. Розмістимо на формі компоненти **Edit**, **Мемо** та **Button**. В **Edit** введемо числове значення в інтервалі від 1 до 10, потім натисканням кнопки послідовно помножимо введене число на 1, 2, 3 ... 10. На рис. 9.1 наведено інтерфейс цього завдання. Програма матиме такий вигляд:



Рис. 9.1. Приклад використання циклу **FOR**



```
unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    Memo1: TMemo;
  procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

```

var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  x, i: integer;
begin
  x:=strtoint(Edit1.Text);
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Таблиця множення на '+inttostr(x));
  for i:=1 to 10 do
    Memo1.Lines.Add(inttostr(x)+'*'+inttostr(i)+'='+
      inttostr(x*i));
  end;
end.
-----

```

Зверніть увагу на одну часту помилку — використання значення змінної-лічильника після завершення циклу. Після виконання циклу значення змінної-лічильника не визначене! Тому, якщо ви хочете використовувати далі значення змінної, їй необхідно присвоїти нове значення явно.

➔ Цикл із передумовою

Цикл **WHILE** (англ. «доки») — цикл, у якому умова розташована перед тілом циклу, а сам цикл виконується доти, доки умова не стане хибною.

Загальний вигляд цієї конструкції такий:

```

WHILE {умова} DO
{дії}

```

Як умова задається логічний вираз. Ті операції, які будуть виконуватись у циклі, називаються тілом циклу.

Особливістю циклу з передумовою є те, що він може не виконатися жодного разу — у тому випадку, якщо зазначена умова від початку буде хибною. При цьому цикл може стати й «вічним» — якщо умова ніколи не набере значення **False**. Саме тому слід стежити за тим, щоб завжди було задано умови для завершення роботи циклу.

Розглянемо на прикладі використання циклу з передумовою.

Завдання. Вивести на формі фрагмент таблиці множення для значення, що вводиться. Для розв'язання завдання використати цикл **WHILE**.

Розв'язання. Оскільки умова завдання еквівалентна умові попереднього завдання, наведемо тільки фрагмент коду, що відповідає обробковій події **OnClick** кнопки:

```

var
  x,i:integer;
begin
  x:=strtoint(Edit1.Text);
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Таблиця множення на '+inttostr(x));
  i:=1;
  while i<=10 do
  begin
    Memo1.Lines.Add(inttostr(x)+'*'+inttostr(i)+'='+
      inttostr(x*i));
    i:=i+1;
  end;
end;

```

➔ Цикл із постумовою

Третій, завершальний, цикл — цикл із постумовою REPEAT (англ. «повтор»). Прикметно, що цього циклу в багатьох мовах програмування немає — є тільки FOR і WHILE. Тим часом цикл із постумовою дуже зручний.

Працює цикл точно так само, як і WHILE, з однією лише відмінністю, що впливає з його назви: умова циклу розташована після тіла циклу, а не до нього.

Загальний вигляд цієї конструкції такий:

```

REPEAT
{дії}
UNTIL {умова виходу з циклу};

```

Є кілька моментів, на які слід звернути увагу. По-перше, задається умова виходу з циклу, у той час як у циклі WHILE задається умова продовження циклу. По-друге, у разі наявності кількох команд, які розміщуються в тілі циклу, брати їх у блок BEGIN ... END не потрібно — зарезервовані слова REPEAT ... UNTIL самі становлять аналогічний блок.

Цикл із постумовою, на відміну від циклу з передумовою, завжди виконується хоча б один раз! Але так само, як і цикл WHILE, у разі неправильно написаної умови він може стати «вічним».

Розглянемо на прикладі використання циклу з постумовою.

Завдання. Вивести на формі фрагмент таблиці множення для значення, що вводиться. Для розв'язання завдання використати цикл REPEAT ... UNTIL.

Розв'язання. Оскільки умова завдання еквівалентна умові попередніх завдань, наведемо тільки фрагмент коду, що відповідає обробникові події OnClick кнопки:

```

var
  x, i: integer;
begin
  x:=strtoint(Edit1.Text);
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Таблиця множення на '+inttostr(x));
  i:=1;
  repeat
    Memo1.Lines.Add(inttostr(x)+'*'+inttostr(i)+'='+
      inttostr(x*i));
    i:=i+1;
  until i>10;
end;

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть причини використання циклів.
2. У чому відмінність циклу **FOR**, що використовує опцію **TO** і **DOWNTO**?
3. Скільки разів виконається цикл **FOR i:=2 TO 8 DO**?
4. Скільки разів виконається цикл **FOR i:=10 TO 4 DO**?
5. У чому відмінність циклів **WHILE** і **REPEAT ... UNTIL**?
6. Який цикл може не виконатися жодного разу?
7. Який цикл обов'язково виконається хоча б один раз?



ПРАКТИЧНІ ЗАВДАННЯ

1. Ввести на формі два цілих додатних числа M і N . Роздрукувати всі парні числа в інтервалі від M до N .
2. Вивести на формі всі двозначні числа, що складаються з однакових цифр (11, 22, ...).
3. Вивести на формі всі тризначні числа, які починаються і закінчуються на одну й ту саму цифру.
4. Ввести на формі ціле додатне число. Роздрукувати всі прості дільники введеного числа.
5. Ввести на формі два цілих додатних числа. Роздрукувати найбільший спільний дільник для введених чисел.
6. Ввести на формі два цілих додатних числа. Роздрукувати найменше спільне кратне для введених чисел.

9.2

Розв'язування задач, у яких використовуються обчислення за ітераційними формулами. Вкладені цикли**➤ Розв'язування задач, у яких використовуються обчислення за ітераційними формулами**

Дуже часто в задачах програмування необхідно одержати доступ до цифр цілого числа. Це може знадобитися для перетворення чисел з однієї системи числення в іншу, для знайдення суми цифр числа, для відшукування найбільшої і найменшої цифр у записі числа й багато для чого ще. Розглянемо на прикладі подібні перетворення.

Приклад 9. Дано чотиризначне ціле додатне число. Необхідно одержати суму цифр, що складають запис цього числа. Наприклад, дано число 1234; відповідь: $1+2+3+4=10$.

Розв'язання. Спробуємо написати алгоритм перетворення — розбиття числа на цифри за розрядами.

$1234 \bmod 10 = 4$. Ми виділили розряд одиниць.

$1234 \div 10 = 123$. Число стало в 10 разів меншим, а розряд десятків опинився останнім.

$123 \bmod 10 = 3$. Виділили розряд десятків.

$123 \div 10 = 12$. Наше число стало в 10 разів меншим, а розряд сотень опинився останнім.

$12 \bmod 10 = 2$. Виділили розряд сотень.

$12 \div 10 = 1$. Число стало в 10 разів меншим, а розряд тисяч опинився останнім.

$1 \bmod 10 = 1$. Ми виділили розряд тисяч.

Тепер якщо ми зможемо підсумувати остачі від ділення, то одержимо потрібний результат. При цьому простежується циклічна послідовність дій. На рис. 9.2 подано інтерфейс завдання. Програма матиме такий вигляд:

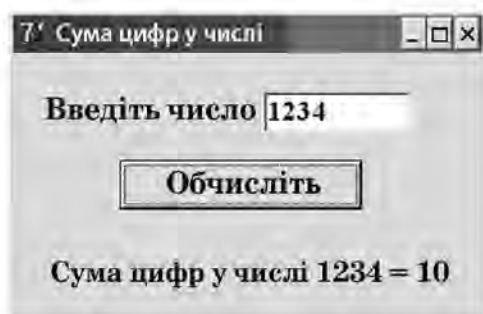


Рис. 9.2. Визначення суми цифр у записі числа

```

unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes,
  Graphics, Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label2: TLabel;
  end;

```

```

    procedure Button1Click(Sender: TObject);
    private
        { Private declarations }
    public
        { Public declarations }
    end;

var
    Form1: TForm1;
implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
    x,i,s,ost,y: integer;
begin
    x:=strtoint(Edit1.Text);
    s:=0;
    y:=x;
    for i:=1 to 4 do
    begin
        ost:=x mod 10;
        x:=x div 10;
        s:=s+ost;
    end;
    x:=y;
    label2.Caption:='Сума цифр у числі ' +inttostr(x) +'='
    +inttostr(s);
end;
end.

```

Використання циклу FOR для розв'язування подібних завдань виправдане лише у випадку, якщо відома кількість цифр у записі числа. Якщо ж кількість цифр не відома, то зручно скористатися циклом WHILE. Для розв'язання цього завдання в загальному вигляді перепишемо цикл:

```

var
    x,i,s,ost,y: integer;
begin
    x:=strtoint(Edit1.Text);
    s:=0;
    y:=x;

```

```

while x>0 do
begin
    ost:=x mod 10;
    x:=x div 10;
    s:=s+ost;
end;
x:=y;
label2.Caption:='Сума цифр у числі ' +inttostr(x) +'='
+inttostr(s);
end;

```

➔ Вкладені цикли

Вкладений цикл — це частина алгоритму або програми, що неодноразово виконується і перебуває в тілі іншого, зовнішнього, циклу.

Для кожного значення змінної-лічильника зовнішнього циклу лічильник вкладеного циклу встигає пробігти всі свої значення.

Ніяких спеціальних конструкцій для вкладених циклів немає. Усе працює так, як і у випадку використання звичайних циклів. Застосовуються вони здебільшого для оброблення таблиць (двовимірних масивів).

Розглянемо найпростіший приклад застосування вкладеного циклу.

Приклад 10.

Вивести таблицю множення.

Розв'язання. Необхідно створити цикл, що послідовно надає множнику значення від 1 до 9, формуючи стовпчик добутків. А щоб вивести таблицю (перемноживши числа від 1 до 10), потрібен ще один такий самий цикл. На рис. 9.3 наведено інтерфейс цього завдання. Програма матиме такий вигляд:



Рис. 9.3. Приклад використання вкладеного циклу

```

unit Unit1;
interface
uses
    Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms, Dialogs, StdCtrls;
type
    TForm1 = class(TForm)
        Button1: TButton;
        Memo1: TMemo;

```

```

    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  i, j: integer;
begin
  Memo1.Lines.Clear;
  for i:=1 to 9 do
  begin
    Memo1.Lines.Add('Таблиця множення на '+inttostr(i));
    for j:=1 to 10 do
      Memo1.Lines.Add(inttostr(i)+'*'+inttostr(j)+'='+
        inttostr(i*j));
    end;
  end;

end.

```

➔ Переривання й продовження циклу

Виконання фіксованого числа ітерацій не завжди приводить до потрібного результату. Іноді можуть виникнути ситуації, коли було б логічним перервати цикл, не виконуючи його до кінця, тобто просто виключити всі дальші ітерації. Така можливість існує — для цього необхідно скористатися командою `Break`. Ця команда завершує цикл, що виконується в цей момент, і продовжує далі виконання програми. При цьому поточна ітерація до кінця не виконується — переривання відбувається саме в тому рядку, на якому записана команда `Break`.

Якщо цикл, виконання якого переривається командою `Break`, вкладений в інший цикл, то зовнішній цикл продовжить виконуватися, тобто команда `Break` зупиняє тільки один цикл, а не всі наявні.

Команда `Break` — це вихід із циклу. Іноді ж потрібно просто пропустити поточну ітерацію й перейти до наступної. Вручну це можна зробити, узявши всі команди в блок умовного оператора, однак такий спосіб не дуже зручний. Саме тому існує команда продовження циклу, вона називається `Continue`. Ця команда змушує цикл відразу перейти до наступної ітерації, не продовжуючи виконання поточної.

**ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ**

1. Поясніть на прикладах використання операцій `div` і `mod`.
2. Опишіть алгоритм переведення цілого додатного числа, заданого в десятковій системі числення, у двійкову систему.
3. У чому особливість використання циклів `WHILE` і `FOR` у переведенні чисел у двійкову систему числення?
4. Що таке вкладені цикли?
5. Яким оператором здійснюється переривання циклу? У яких випадках таке переривання необхідне?

**ПРАКТИЧНІ ЗАВДАННЯ**

1. Ввести на форму ціле чотирирозрядне додатне число. Перевірити правильність твердження: сума перших двох цифр дорівнює сумі двох останніх. Наприклад: $4581 \rightarrow 4+5=8+1$.
2. Ввести на форму ціле чотирирозрядне додатне число. Перевірити правильність твердження: задане число симетричне (числа, запис яких читається однаково справа наліво і зліва направо, називаються паліндромами). Наприклад: 1551, 8778 і т. ін.
3. Ввести на форму ціле чотирирозрядне додатне число. Перевірити, чи містить задане число три однакові цифри. Наприклад: 6676, 4544, 6000.
4. Ввести на форму ціле чотирирозрядне додатне число. Перевірити, чи містить задане число чотири різні цифри. Наприклад: 1234, 8976, 3461.
5. Ввести на форму ціле чотирирозрядне додатне число. Показати, на яких позиціях (розряди) у заданому числі стоять цифри, кратні числу 3.
6. Ввести на форму ціле чотирирозрядне додатне число. Перевірити, чи всі цифри заданого числа парні. Наприклад: 6842 — «так», 6354 — «ні».

Для завдань вищої складності розв'язати завдання 1–6, використовуючи числа довільної розрядності.

Практична робота № 9. Використання циклів

Мета роботи: навчитися використовувати цикли в ході розроблення додатків.

Завдання. Ввести кількість членів ряду й розрахувати значення π .

Теоретичні відомості

На початку XVIII ст. англійський математик Брук Тейлор запропонував низку формул для розкладання функцій у степеневі ряди. Використання цього відкриття відіграло важливу роль у реалізації на обчислювальних машинах різних функцій. Замість зберігання величезних таблиць значень функцій для різних аргументів у бібліотеці мови програмування може бути записаний лише алгоритм визначення результуючого значення шляхом обчислення нескладної формули — суми обмеженого числа елементів ряду (числової послідовності).

Найвідоміша з цих формул, яка дістала назву «ряд Тейлора», призначена для обчислення функції $\arctg(x)$:

$$\arctg(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \dots$$

Якщо у формулу підставити $x=1$, то $\arctg(1) = \frac{\pi}{4}$.

Тоді формула набуває вигляду:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots$$

або

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \right).$$

Багато поколінь програмістів користувалися цим рядом для обчислення значення числа π із заданою точністю.

Хід роботи

1. Запустимо середовище Delphi та створимо новий додаток. На форму (рис. 9.4) помістимо три відеокомпоненти: кнопку, Edit та Label. В Edit вводиться кількість членів ряду, залучена до підсумовування. Натисканням кнопки в Label виводиться обчислене значення π .
2. Уся програма для розрахунку π міститься в методі (процедурі), що викликається натисканням однієї кнопки на формі, і має такий вигляд:

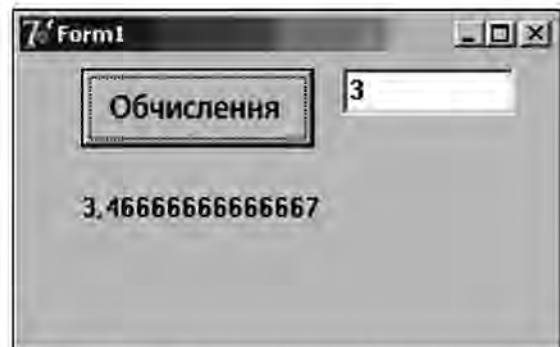


Рис. 9.4. Вигляд форми для обчислення числа π за допомогою ряду Тейлора

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i,k,zn:integer;
    s:real;
begin
    k:=strtoint(Edit1.Text);
    s:=0; {сума членів ряду}
    zn:=1; {знак чергового доданка, чергується +1, -1, +1,...}
    for i:=1 to k do

```

```

begin
    s:=s + zn*(1/(2*i-1));
    zn:=zn*(-1);
end;
label1.caption:=floattostr(4*s);
end;

```

3. Наведемо деякі пояснення до запропонованої програми. Зверніть увагу на типи даних: кількість членів ряду, параметр циклу й знак доданка — цілочислові змінні; сума ряду, що накопичує знакозмінні значення простих дробів, — дійсна величина. Знак чергового доданка визначається окремою змінною, хоча можна було б запропонувати і який-небудь арифметичний вираз, наприклад:

$$2 * (i \bmod 2) - 1$$

Цей вираз дає значення «-1» для парних i та «+1» — для непарних. Оскільки в циклі після оператора FOR виконуються відразу два оператори, в оформленні циклу застосовано операторні дужки `begin ... end`, що слугують для об'єднання кількох операторів (у цьому випадку їх два) в один складений оператор.

4. Якщо запустити програму і спробувати вводити різні числа, то одержимо, що ряд з одного елемента дає значення π , яке дорівнює 4, із двох елементів — 2,666..., із трьох — 3,4666... і т. д. Зі збільшенням числа доданків точність зростатиме.

5. Збережемо проект на диску.

6. Створимо новий додаток, потрібний для розв'язування наступної задачі.

Ще одним популярним алгоритмічним застосуванням циклів є перевірка на подільність: досліджуване ціле число послідовно ділять на ряд натуральних чисел (зазвичай від 2 до $n-1$, хоч у дійсності слід перевіряти на подільність від 2 до \sqrt{n}), після чого й перевіряють остачу від ділення — якщо ніде не вийде значення 0, то число просте. Розглянемо кілька прикладів.

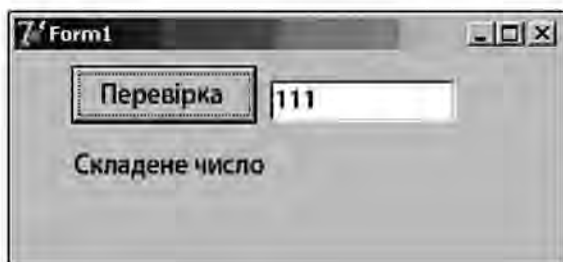


Рис. 9.5. Форма програми для визначення простих чисел

Завдання 1. Ввести ціле число й вивести в Label рядок «Просте число» або «Складене число» після перевірки його подільності на натуральний ряд чисел від 2 до $n-1$.

Розв'язання. Спроекуємо форму, як показано на рис. 9.5.

Текст програми, набраний у методі OnClick — єдиної кнопки на формі, — має такий вигляд:

```

-----
procedure TForm1.Button1Click(Sender: TObject);
var
    i,k:integer;
begin
    k:=strtoint(Edit1.text);
    Label1.caption:='Просте число';
    for i:=2 to k - 1 do
        if k mod i = 0 then
            Label1.caption:='Складене число';
    end;
-----

```

Зауважимо, що зазвичай цю задачу розв'язують за допомогою додаткової логічної змінної (прапорця), яка міняє своє вихідне значення, якщо число ділиться хоч на один дільник. У наведеному фрагменті роль змінної-прапорця відіграє рядок виведення в компонент Label (властивість Caption). У програмуванні прапорцем (flag) прийнято називати змінну, яка набуває одного з двох значень: true або false, 0 або 1 і под. У цьому випадку таке використання Caption цілком допустиме, бо вся задача будувалася для того, щоб вивести на екран це повідомлення. Протягом іншого використання одержаного результату не йшлося.

Завдання 2. Визначити, на який мінімальний множник потрібно помножити число 9, щоб запис результату складався тільки з цифр 2.

Розв'язання. За кодом програми задача така сама проста, як і попередня, однак алгоритм її розв'язування слід обдумати.

Розв'язувати задачу будемо методом від супротивного: припустімо, ми знайшли число, що складається тільки з цифр 2 і ділиться на 9. Тоді з усіх можливих таких чисел нам треба вибрати мінімальне. Реалізуємо це програмним шляхом. Виникає єдине питання: як зібрати числа із самих двійок (2, 22, 222, 2222 і т. д.)? Робиться це досить просто, однак необхідно стежити за межами (діапазонами) цілочислових типів даних — числа можуть сягати величезних значень. Тому скористаємося типом даних longint (cardinal).

У методі OnClick (єдиної кнопки на формі) записано такий код:

```

-----
procedure TForm1.Button1Click(Sender: TObject);
var
    chislo,stepin:longint;
begin
    chislo:=2;
    stepin:=10;
    while chislo mod 9 <>0 do

```

```

begin
  chislo:=chislo+2*stepin;
  stepin:=stepin*10;
end;
Label1.caption:=inttostr(chislo div 9);
end;

```

Ця задача має елегантне аналітичне розв'язання. Ознакою подільності на 9 є той факт, що сума цифр числа ділиться на 9. Очевидно, що мінімальним із таких чисел є 222 222 222. Звідси випливає відповідь задачі: 222 222 222/9.

Примітка. Три запропоновані додатки дуже схожі один на одного й можуть бути одержані один з одного шляхом часткового змінення коду. Зберігати їх треба в різні папки.



ЗАДАЧІ ДЛЯ САМОСТІЙНОГО РОЗВ'ЯЗУВАННЯ

1. За допомогою функціонального ряду для введеного x обчислити значення $\arctg(x)$ з n першими числами послідовності:

$$\arctg(x) = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \dots$$

Примітка. У цьому випадку на формі буде два Edit для введення n і x . Степінь x у чисельнику доведеться обчислювати за допомогою окремої змінної так само, як це робилося зі знаком доданка в розглянутому вище прикладі (див. рис. 9.4).

2. За допомогою функціонального ряду для введеного x обчислити значення $\cos(x)$ з n першими числами послідовності:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \dots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Примітка. У цьому випадку на формі буде два Edit для введення n і x . Степінь x у чисельнику доведеться обчислювати за допомогою окремої змінної, ще одна змінна буде потрібна для обчислення факторіала в знаменнику.

3. За допомогою функціонального ряду для введеного x обчислити значення $\sin(x)$ з n першими числами послідовності:

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \dots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!}.$$

Примітка. У цьому випадку на формі буде два Edit для введення n і x . Степінь x у чисельнику доведеться обчислювати за допомогою окремої змінної, ще одна змінна буде потрібна для обчислення факторіала в знаменнику.

Практична робота № 10

Програмування обчислень за ітераційними формулами



Мета роботи: опанувати програмування ітераційних алгоритмів.

Завдання. Написати програму, що здійснює переведення цілого додатного числа з десяткової системи числення у двійкову.

Теоретичні відомості

Системи числення — це способи кодування числової інформації, тобто способи записування чисел за допомогою певного алфавіту, символи якого прийнято називати цифрами.

У двійковій системі числення використовуються всього дві цифри: 0 і 1. Саме тому вона лежить в основі роботи комп'ютера, бо в комп'ютері існують два стійкі стани: низька або висока напруга; є струм чи немає струму; намагнічено чи не намагнічено тощо. Одному стану відповідає значення 1, другому — 0.

Загальне правило переведення з десяткової системи таке: щоб перевести число з десяткової системи числення в будь-яку іншу, потрібно ділити число на основу нової системи числення доти, доки частка від ділення не буде меншою за основу системи числення, при цьому необхідно фіксувати всі остачі від ділення. Потім потрібно записати частку від ділення та всі остачі, починаючи з останньої, у зворотній послідовності. Таким чином, вийде: частка — найстарший розряд, а перша остача — наймолодший розряд.

Приклад переведення числа у двійкову систему подано на рис. 9.6.

Хід роботи

1. Запустимо середовище Delphi та створимо новий додаток.
2. Винесемо на форму необхідні компоненти. Інтерфейс програми показано на рис. 9.7.

$$\begin{array}{r}
 391 \overline{) 2} \\
 390 \overline{) 195} \overline{) 2} \\
 1 \overline{) 194} \overline{) 97} \overline{) 2} \\
 1 \overline{) 96} \overline{) 48} \overline{) 2} \\
 1 \overline{) 48} \overline{) 24} \overline{) 2} \\
 0 \overline{) 24} \overline{) 12} \overline{) 2} \\
 0 \overline{) 12} \overline{) 6} \overline{) 2} \\
 0 \overline{) 6} \overline{) 3} \overline{) 2} \\
 0 \overline{) 2} \overline{) 1} \\
 1
 \end{array}
 \quad \text{Відповідь: } 391_{10} = 110000111_2$$

Рис. 9.6. Переведення числа 391 у двійкову систему

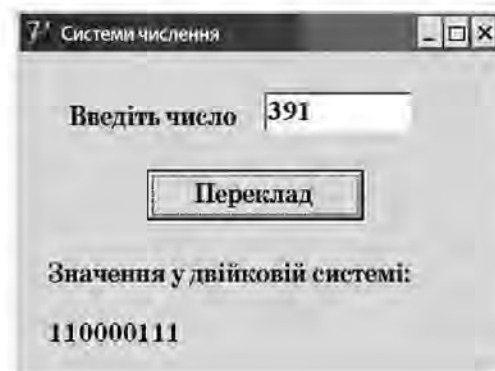


Рис. 9.7. Інтерфейс програми переведення числа у двійкову систему

3. В обробник кнопки OnClick помістимо код:

```
-----  
procedure TForm1.Button1Click(Sender: TObject);  
var  
    x,y:integer;  
begin  
    x:=strtoint(Edit1.Text);  
    label3.Caption:=' ';  
    while x>0 do  
    begin  
        y:=x mod 2;  
        x:=x div 2;  
        if y=0 then  
            label3.Caption:='0'+label3.Caption  
        else  
            label3.Caption:='1'+label3.Caption;  
        end;  
    end;  
end.
```

4. Збережемо проект і запустимо програму.



Під час вивчення цього розділу ми навчимося використовувати різноманіття мовних засобів у роботі з підпрограмами. Розглянемо ситуації, що вимагають у ході написання програмного коду використання процедур і функцій, познайомимося зі вбудованими підпрограмами для роботи з рядковими типами даних. Велику увагу буде приділено роботі з графічними можливостями Delphi.

10.1

Поняття підпрограми. Оголошення підпрограми, її тіло та оператор її виклику. Поняття функції. Створення й використання власних функцій

➔ **Поняття підпрограми. Оголошення підпрограми, її тіло та оператор її виклику**

Уявімо таку ситуацію: у нашій програмі одна й та сама послідовність дій повторюється і прописана в різних її частинах. У чому недоліки такого дублювання? Насамперед більш громіздким стає код. Але це не головне. Річ у тім, що пряме копіювання фрагментів коду викликає труднощі в подальшій модифікації коду програми — необхідно вносити зміни в різні частини програми. Наприклад, наша програма розраховує якісь дані за заданою формулою. Крім розрахунків, програма будує графіки, зберігає інформацію і т. д. І в якийсь момент з'ясовується, що формулу розрахунку потрібно змінити. А в нас вона по всій програмі задана в явному вигляді. Доведеться її виправляти скрізь — інакше виникнуть серйозні проблеми: половина розрахунків стануть хибними, не ті дані будуть збережені та ін. Було б логічно задати формулу розрахунку один раз і в який-небудь спосіб її викликати. Отут і допомагають підпрограми.

Підпрограма являє собою набір операторів і команд, оформлених спеціальним чином. Підпрограму можна викликати з основної програми, причому необмежену кількість разів. Виносячи якийсь код у підпрограму, ми виключаємо його дублювання в програмі і, природно, зменшуємо загальний обсяг коду програми. Використання підпрограм надає додатку більш структурованої форми. Якоюсь мірою підпрограми спрощують і читання коду іншими користувачами. Понад те, використання підпрограм дозволяє організувати поділ праці в ході роботи над великим проектом: кожен працює над своїм завданням, а всі результати швидко й просто підключаються до основної програми.

У мові Delphi підпрограми оформлюються у вигляді процедур і функцій. Ім'я процедури й функції задається згідно з правилами задавання імен ідентифікаторів (змінних). Воно може складатися тільки

з латинських літер, цифр і знака підкреслення, при цьому не може починатися з цифри. Імена мають бути унікальними — не можуть існувати підпрограми з однаковими іменами. Це правило має винятки, але їх перелік виходить за межі нашого посібника.

Отже, підпрограмою називається іменована, логічно закінчена група операторів, яку можна викликати за ім'ям (тобто виконати) будь-яку кількість разів із різних частин програми. За структурою підпрограма практично ідентична самій програмі: вона містить заголовок, блок описів, блок реалізації. У загальному випадку робота з підпрограмою здійснюється у два етапи. Спочатку необхідно описати процедуру (функцію), інакше основна програма її просто не знайде. Після того як процедуру (функцію) описано, її можна викликати з основної програми. При цьому, звичайно, ніщо не заважає нам редагувати підпрограму і програму паралельно. Таким чином, ми можемо спочатку просто описати підпрограму й «повісити» на неї найпростішу дію на зразок виведення віконця з повідомленням, потім прописати її виклик у всіх потрібних місцях, протестувати працездатність програми, після чого продовжити написання підпрограми.

Відмінності у використанні процедур і функцій мають принциповий характер. Функції використовують у випадку, коли підпрограма має повернути, обчисливши, лише один результат скалярного типу. При цьому функція може використовуватись у виразах як операнд, на місце якого підставляється результат роботи цієї функції. Якщо нічого не потрібно повертати або ж необхідно повернути більш ніж один параметр, використовують процедури.

⇒ Поняття функції. Створення й використання власних функцій

Розгляд підпрограм почнемо на прикладі функції. У загальному вигляді опис функції є таким:

```
-----
function <ім'я функції> (<список формальних параметрів>):
<тип результату>;
const ...;
type ...;
var ...;
begin
    <оператори>
end;
-----
```

Напишемо функцію, що визначає суму двох дійсних чисел. Продемонструємо два способи повернення результату функції. Наведемо приклад функцій, ідентичних за своїм вмістом:

```

-----
function sum (a,b:real): real;
begin
    sum:=a+b;
end;
-----

```

```

-----
function sum (a,b:real): real;
begin
    Result:=a+b;
end;
-----

```

Правила опису та використання функцій:

1. Заголовок функції починається зі службового слова **function**, за яким зазначається ім'я функції. У наведеному прикладі — **sum**.

2. Слідом за ім'ям у круглих дужках перелічуються вхідні параметри та їхні типи (у цьому випадку два параметри **a, b** типу **real**).

3. Якщо функція не має формальних параметрів, то круглі дужки в заголовку не ставляться.

4. Після дужок, у які взято вхідні параметри, через двокрапку вказують тип значення результату, що повертається. Функція слугує для обчислення тільки одного значення, яке передається в програму через ім'я функції. При цьому тип значення результату, що повертається, — це будь-який скалярний тип даних (числовий, символічний, булів, рядковий тип або покажчик).

5. Усередині функції перед виконуваною її частиною можуть бути оголошені змінні, які називаються локальними. Ці змінні поза функцією не існують, бо під них відводиться пам'ять під час кожного виклику функції, а в разі виходу з функції ці змінні знищуються.

6. Усі змінні, описані в програмі до оголошення функції, також доступні всередині функції. Ці змінні називаються глобальними. Щоб уникнути помилок, доступні змінні всередині функції використовувати не рекомендовано.

7. Тіло функції розташовується між **begin ... end** і являє собою набір команд.

8. Для повернення в програму результату функції ім'я функції у виконуваний частині має одержати своє значення. Із цією метою можна також використовувати внутрішню змінну **Result**. Перевагою використання цієї змінної є те, що вона може брати участь у виразах як операнд. Наприклад, для функції знаходження суми можна використовувати присвоювання **Result:=Result+a**, але неможливе присвоювання **sum:=sum+a**.

Функція викликається всередині основної програми (або іншої функції). Наприклад, для записаної вище функції **sum** можливий такий виклик:

```

var
  x, y, S: real;
begin
  // введення змінних x і y
  S:=sum(x,y);
  // виведення змінної S
end;

```

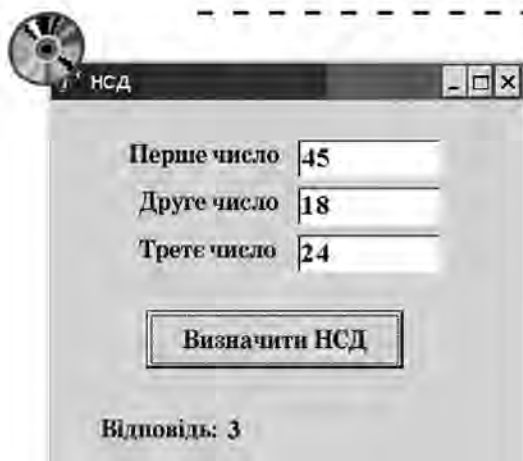


Рис. 10.1. Пошук НСД трьох цілих додатних чисел

Приклад 11. Реалізувати як функцію пошук НСД (найбільшого спільного дільника) двох цілих додатних чисел. В основній програмі визначити НСД для трьох чисел, враховуючи, що $\text{НСД}(a, b, c) = \text{НСД}(\text{НСД}(a, b), c)$.

Розв'язання. Як алгоритм пошуку найбільшого спільного дільника візьмемо алгоритм Евкліда. Приклад візуальної форми для розв'язання задачі наведено на рис. 10.1. Слід звернути увагу на використання типу даних *cardinal*, що являє собою ціле чотирибайтне число без знака.

Програма матиме такий вигляд:

```

unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Button1: TButton;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
  end;

var
  Form1: TForm1;
  a,b,c: cardinal;

```

```

implementation
{$R *.dfm}      // форма пошуку найбільшого спільного дільника
function NOD(x,y:cardinal):cardinal;
begin
  while x<>y do // доки два числа не стануть рівними
    if x>y then // від більшого числа
      x:=x - y // віднімаємо менше число
    else
      y:=y - x;
  Result:=x;
end;
// введення трьох значень і виклик функції НСД
procedure TForm1.Button1Click(Sender: TObject);
begin
  a:=strtoint(edit1.Text); // зчитування першого значення
  b:=strtoint(edit2.Text); // зчитування другого значення
  c:=strtoint(edit3.Text); // зчитування третього значення
  label4.Caption:='Відповідь: '+inttostr(NOD(NOD(a,b),c));
end;
end.

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Чим зумовлена необхідність використовувати підпрограми?
2. У чому відмінності в застосуванні процедур і функцій?
3. Які типи даних може повернути функція?
4. Опишіть, який вигляд має заголовок функції.
5. Яким чином можна записати результат усередині тіла функції?



ПРАКТИЧНІ ЗАВДАННЯ

1. Написати власну функцію, що може визначати мінімальне значення з двох дійсних чисел. На формі організувати введення чотирьох дійсних чисел і пошук за допомогою функції мінімального значення серед введених.
2. Написати функцію, яка для трьох дійсних чисел визначає, чи можуть вони бути сторонами трикутника (сума будь-яких двох сторін має бути більшою, ніж третя сторона). На формі організувати введення трьох чисел. За допомогою функції визначити, чи є три введених числа сторонами трикутника.
3. Написати функцію, яка для цілого числа визначає кількість десятків у його записі (наприклад, 2347 → 4 десятки). На формі організувати введення трьох цілих чисел. За допомогою підпрограми визначити суму десятків у трьох введених числах.
4. Написати функцію, яка визначає суму цифр у цілому додатному числі. На формі організувати введення трьох цілих чисел. За допомогою функції визначити, яке з введених чисел має найбільшу суму цифр.

10.2

Поняття процедури. Створення й виклик процедур. Підпрограми з параметрами. Поняття локальної та глобальної змінних

➔ Поняття процедури. Створення й виклик процедур

За своєю структурою процедури нагадують програму: вони, як і програма, складаються із заголовка й тіла. Заголовок процедури містить у собі зарезервоване слово **procedure**, ім'я процедури та список формальних параметрів, які в разі їх використання беруться в круглі дужки. *Ім'я процедури* — це ідентифікатор, унікальний у межах програми. *Формальні параметри* — це дані, які передаються в процедуру для оброблення, і дані, що їх процедура повертає. Якщо процедура не одержує даних іззовні і нічого не повертає, формальні параметри (у тому числі й круглі дужки) не записуються. Тіло процедури являє собою локальний блок, за структурою аналогічний програмі:

```
-----
procedure <ім'я процедури>(<список формальних параметрів>);
const ...;
type ...;
var ...;
begin
    <оператори>
end;
-----
```

Описи констант, типів даних і змінних дійсні тільки в межах даної процедури. У тілі процедури можна використовувати будь-які глобальні константи та змінні, а також викликати будь-які підпрограми (процедури та функції).

З підпрограмами, реалізованими у вигляді процедур, ми вже мали справу багато разів: будь-який обробник будь-якої події є підпрограмою. Коли ми створюємо обробник натискання кнопки, з'являється заготовка коду, де є заголовок з ім'ям, наприклад `Button1Click`, нижче йде блок **begin ... end** — блок реалізації, а вище від нього ми вписуємо свої змінні, константи й інші необхідні елементи.

Виклик процедури для виконання здійснюється на її ім'я, за яким іде взятий у круглі дужки список фактичних параметрів, тобто даних, що передаються в процедуру:

```
<ім'я процедури> (<список фактичних параметрів>);
```

Якщо процедура не приймає даних, то список фактичних параметрів (у тому числі й круглі дужки) не вказується.

Поняття процедури є надзвичайно важливим, бо саме воно лежить в основі однієї з найпопулярніших технологій розв'язування задач мовою Delphi. Зовні технологія ця проста: задача розбивається на кілька логічно відокремлених підзадач, і розв'язання кожної з них оформлюється

у вигляді окремої процедури. Будь-яка процедура може містити в собі інші процедури, кількість яких обмежена тільки обсягом пам'яті вашого комп'ютера.

Виникає питання: у якому місці в тілі процедури необхідно розміщувати іншу підпрограму? По-перше, її необхідно писати в такому місці, де програма зможе її знайти. Підпрограми можна описувати в розділі описів, де описуються змінні, тільки вгорі, над ними. Наприклад, якщо наша підпрограма використовується тільки в обробнику натискання кнопки і більше ніде, її можна помістити в розділ описів цього обробника. Тоді з інших обробників підпрограму видно не буде. Друге правило: будь-яка підпрограма має бути описана вище від того місця, де вона викликається. Викликати підпрограму, розташовану нижче, не можна, тому що компілятор на цей момент часу не знатиме про підпрограму. І третє правило: щоб підпрограму було видно з будь-якого місця поточного модуля (поточної форми), її слід описати в розділі **implementation** вище за всі обробники подій.

✦ Підпрограми з параметрами. Параметри процедур і функцій

Параметри слугують для передавання вихідних даних у підпрограми та приймання результатів роботи цих підпрограм.

Вихідні дані передаються в підпрограму за допомогою вхідних параметрів, а результати роботи підпрограми повертаються через вихідні параметри. Параметри можуть також бути вхідними й вихідними одночасно.

Вхідні параметри оголошуються за допомогою ключового слова **const**; їхні значення не можуть бути змінені всередині підпрограми. Розглянемо як приклад функцію пошуку найменшого значення з двох цілих чисел:

```
-----
function min(const a, b: Integer): Integer;
begin
    if a < b then
        Result := a
    else
        Result:=b;
end;
```

Для оголошення вихідних параметрів слугує ключове слово **out**:

```
-----
procedure SizeForm (out width, height:integer);
begin
    width:=Form1.ClientWidth;
    height:=Form1.ClientHeight
end;
```

Присвоювання значень вихідних параметрів усередині підпрограми приводить до присвоювання значень змінних, переданих як аргументи:

```
-----  
procedure TForm1.Button1Click(Sender: TObject);  
var  
    w,h:integer;  
begin  
    SizeForm(w,h);  
    caption:=inttostr(w)+' '+inttostr(h);  
end;  
-----
```

Після виклику процедури `SizeForm` змінні `w` і `h` міститимуть значення, які було присвоєно формальним параметрам `width` і `height` відповідно.

Якщо параметр є одночасно і вхідним, і вихідним, то він описується з ключовим словом **var**:

```
-----  
procedure Exchange(var a, b: Integer);  
var  
    c: Integer;  
begin  
    c := a;  
    a := b;  
    b := c;  
end;  
-----
```

Зміна значень **var**-параметрів усередині підпрограми приводить до зміни значень змінних, переданих як аргументи:

```
-----  
var  
    x, y: Integer;  
begin  
    x := 7;  
    y := 13;  
    ...  
    Exchange(x, y);  
    // Тепер x = 13, y = 7  
    ...  
end;  
-----
```

Викликаючи підпрограми, на місце **out**- і **var**-параметрів можна підставляти тільки змінні, але не константи й не вирази.

Якщо в описі параметра немає жодного з ключових слів: **const**, **out** або **var**, то параметр вважається вхідним. Його можна змінювати, при цьому не впливаючи на фактичний аргумент, оскільки всі зміни виконуються з копією аргументу, створюваною на час роботи підпрограми. Викликаючи підпрограму, як такий параметр можна використовувати константи й вирази. Різні способи передавання параметрів: з використанням **const**, **out**, **var** і без них (табл. 10.1) — можна сполучати в одній підпрограмі.

Таблиця 10.1

Способи передавання параметрів

Ключове слово	Призначення параметра	Спосіб передавання
<відсутнє>	Вхідний	Передається копія значення
const	Вхідний	Передається копія значення або посилання на значення залежно від типу даних
out	Вихідний	Передається посилання на значення
var	Вхідний і вихідний	Передається посилання на значення

Якщо передається значення, то підпрограма маніпулює копією аргументу. Такий спосіб називається передаванням параметрів за значеннями (англ. *parameter passing by value*).

Якщо передається посилання на значення, то підпрограма маніпулює безпосередньо аргументом, звертаючись до нього через передану адресу. Цей спосіб називається передаванням параметрів за посиланням (англ. *parameter passing by reference*).

Приклад 12. Створимо процедуру Average, що набуває чотирьох параметрів. Перші два параметри (x і y) є вхідними й служать для передавання вихідних даних. Другі два параметри є вихідними й служать для приймання в програмі виклику результатів обчислення середнього арифметичного (sa) та середнього геометричного (sg) від значень x і y .

Розв'язання. Скористаємося формою, поданою на рис. 10.2. У результаті маємо:

```

unit Unit1;
interface
uses Windows, Messages, SysUtils,
  Variants, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls;

```

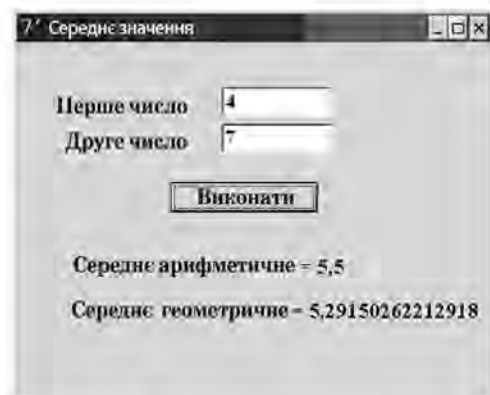


Рис. 10.2. Пошук середнього арифметичного й середнього геометричного значень



type

```

TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Label3: TLabel;
    Label4: TLabel;
    procedure Button1Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

```

var

```

Form1: TForm1;
a, b, s1, s2: Real;

```

implementation

```

{$R *.dfm}

```

```

// процедура пошуку середнього арифметичного й середнього
// геометричного

```

```

procedure Average(const x, y: Real; out sa, sg: Real);

```

begin

```

    sa:= (x + y) / 2; // обчислення середнього арифметичного
    sg:= Sqrt(x * y); // обчислення середнього геометричного

```

end;

```

// введення даних і виклик процедури

```

```

procedure TForm1.Button1Click(Sender: TObject);

```

begin

```

    a:=StrToFloat(Edit1.Text); // прочитати перше число
    b:=StrToFloat(Edit2.Text); // прочитати друге число
    Average(a, b, s1, s2); // виклик процедури Average
    label3.Caption:='Середнє арифметичне = '
+ FloatToStr(s1);
    label4.Caption:='Середнє геометричне = '
+ FloatToStr(s2);
end;

```

end.

➔ Прив'язка підпрограм до форми

Якщо написані нами підпрограми займаються діями, не пов'язаними з візуальними компонентами форми (це, наприклад, які-небудь арифметичні обчислення), то вони можуть просто розміщуватися в коді. Однак спроби звернутися з підпрограми до форми або до її компонентів нічого не дадуть: підпрограма «не знає» про нашу форму. Для того щоб у процедурі або функції можна було працювати з компонентами форми, необхідно виконати певні дії — оформити їх у вигляді методу форми.

Розглянемо конкретний приклад. Припустимо, у нас на формі є три кнопки: `Button1`, `Button2` і `Button3`. Нам потрібно в різних частинах програми по черзі вмикати й вимикати ці кнопки (використовуючи властивість `Enabled`). Щоб не копіювати один і той самий код, краще написати підпрограму, яка робитиме все необхідне. Запишемо нашу процедуру:

```
-----
procedure EnableButtons(Enabl: Boolean);
begin
    Button1.Enabled:=Enabl;
    Button2.Enabled:=Enabl;
    Button3.Enabled:=Enabl
end;
```

Однак такий код програми видасть помилку компіляції. Це пов'язане з тим, що розроблена процедура «не бачить» кнопку `Button1` (як «не бачить» і інші дві).

Щоб «прив'язати» процедуру до форми, нам потрібно виконати такі дії:

1. Додати заголовок методу в опис самої форми. Для цього потрібно скопіювати заголовок нашої процедури в секцію **private** або **public**, у розділ **type**, де описано нашу форму. Вибір секції — **private** або **public** — у кожному випадку слід робити окремо. Усе, що описане в **private**, доступне тільки в рамках даної форми. А все, що описане в **public**, може бути доступне з інших форм і модулів.

2. У розділі **implementation** у заголовку процедури перед назвою додати ім'я класу форми. Ім'я класу форми в Delphi автоматично формується з літери «Т» й імені самої форми. Наприклад, для форми `Form1` ім'я класу `TForm1`. Між ім'ям класу форми й ім'ям підпрограми має стояти крапка. У результаті заголовок процедури матиме такий вигляд:

```
procedure TForm1.EnableButtons(Enabl: Boolean);
```

Якщо виклик створеної нами процедури здійснюється із самої `Form1`, то можна звертатися до процедури просто за ім'ям. Наприклад, помістимо на форму прапорець типу `TCheckBox` і в обробнику його події `OnClick` напишемо:

```

procedure TForm1.CheckBox1Click(Sender: TObject);
begin
    EnableButtons(checkbox1.Checked)
end;

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Назвіть причини використання процедур.
2. У яких випадках у передаванні параметрів використовується ключове слово **const**?
3. У яких випадках у передаванні параметрів використовується ключове слово **out**?
4. У яких випадках у передаванні параметрів використовується ключове слово **var**?
5. Що собою являють способи передавання параметрів за значеннями і за посиланням?



ПРАКТИЧНІ ЗАВДАННЯ

1. Що виведе поданий нижче фрагмент програми?

```

var
    x, y, z: integer;
procedure p (a: integer; var b, c: integer);
begin
    b:=b - 1;
    b:=a + c - 2;
    c:=- a;
    a:=b + c;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    x:=1;
    y:=9;
    z:=-2;
    p(x,y,z);
    label1.Caption:=IntToStr(x)+' '+IntToStr(y)+' '
+IntToStr(z);
end;
end.

```

2. Що виведе поданий нижче фрагмент програми?

```

var
    x, y, z : integer;
procedure p (a:integer; const b: integer; out c: integer);
begin
    a:=a + c;
    c:=a - b - 2;

```

```

c:=a - 8;
a:=c - 1;
end;

```

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    x:=2;
    y:=1;
    z:=7;
    p(x,y,z);
    label1.Caption:=IntToStr(x)+' '+IntToStr(y)+' '
+IntToStr(z);
end;
end.

```

3. Що виведе поданий нижче фрагмент програми?

```

var
    x, y, z: integer;
procedure p (var a: integer; b, c: integer);
begin
    a:=b + c;
    b:=a + c - 2;
    c:=13 - a;
    b:=b - 1;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    x:=2;
    y:=-1;
    z:=4;
    p(x,y,z);
    label1.Caption:=IntToStr(x)+' '+IntToStr(y)+' '
+IntToStr(z);
end;
end.

```

4. Написати процедуру, яка в цілому додатному числі визначає суму та добуток його цифр. На формі організувати введення трьох цілих чисел і виведення для кожного з них суми та добутку його цифр.
5. Написати процедуру, що перевіряє, чи є запис цілого додатного числа паліндромом. На формі організувати введення п'яти цілих чисел і пошук суми чисел, які є паліндромами.

10.3**Вбудовані процедури та функції: рядкові, перетворення типів даних, генератор псевдовипадкових чисел****➔ Вбудовані процедури та функції**

Усі процедури та функції мови Delphi поділяються на дві групи: вбудовані і визначені програмістом. Вбудовані процедури та функції (табл. 10.2–10.6) є частиною мови й можуть викликатися за ім'ям без попереднього опису.

Таблиця 10.2

Арифметичні функції

Назва	Призначення	Тип результату
Abs(X)	Повертає абсолютне значення аргументу X	Integer або Real
Exp(X)	Повертає значення експоненти в степені X	Real
Ln(X)	Повертає натуральний логарифм аргументу X	Real
Pi	Повертає значення числа π	Extended
Sqr(X)	Повертає квадрат аргументу X	Integer або Extended
Sqrt(X)	Повертає квадратний корінь аргументу X	Extended

Приклади:

Вираз	Результат
Abs(-7)	7
Exp(1)	2.71828182845905
Ln(Exp(1))	1
Pi	3.14159265358979
Sqr(7)	49
Sqrt(64)	8

Таблиця 10.3

Тригонометричні функції

Назва	Призначення	Тип результату
ArcTan(X)	Повертає кут, тангенс якого дорівнює X (результат у радіанах)	Extended
Cos(X)	Повертає косинус аргументу X (X задається в радіанах)	Extended
Sin(X)	Повертає синус аргументу X (X задається в радіанах)	Extended

Приклади:

Вираз	Результат
ArcTan(Sqrt(3))	1.04719755119660
Cos(Pi/3)	0.5
Sin(Pi/6)	0.5

Слід зазначити, що до середовища Delphi входить стандартний модуль Math, який містить чимало додаткових підпрограм для тригонометричних, логарифмічних, статистичних та фінансових обчислень.

Таблиця 10.4

Функції виділення цілої або дробової частини

Назва	Призначення	Тип результату
Frac(X)	Повертає дробову частину аргументу X	Extended
Int(X)	Повертає цілу частину дійсного числа X	Extended
Round(X)	Округлює дійсне число X до цілого	Int64
Trunc(X)	Повертає цілу частину дійсного числа X. Результат належить до цілого типу	Int64

Приклади:

Вираз	Результат
Frac(2.5)	0.5
Int(2.5)	2.0
Round(2.7)	3
Trunc(2.5)	2

Порядковими змінними називаються величини типів integer, char, boolean, а також похідні від цих типів. Для всіх них передбачено операції > і <, що визначають, яке значення є попереднім, яке — наступним.

Таблиця 10.5

Підпрограми для роботи з порядковими величинами

Назва	Призначення	Тип результату
Chr(X)	Повертає символ, порядковий номер якого дорівнює X	Char
Dec(X, [N])	Зменшує цілу змінну X на 1 або на задане число N	Ordinal type або Pointer
Inc(X, [N])	Збільшує цілу змінну X на 1 або на задане число N	Ordinal type або Pointer
Odd(X)	Повертає True, якщо аргумент X є непарним числом	Boolean
Ord(X)	Повертає порядковий номер аргументу X у своєму діапазоні значень	Ordinal type
Pred(X)	Повертає значення, що передуює значенню аргументу X в діапазоні цього типу	Ordinal type
Succ(X)	Повертає значення, що йде за значенням аргументу X в діапазоні цього типу	Ordinal type

Приклади:

Вираз	Результат
Chr(65)	'A'
Odd(3)	TRUE
Ord('A')	65
Pred('B')	'A'
Succ('A')	'B'

Усі розглянуті вище підпрограми реалізовані у вигляді функцій, що повертають результат залежно від значення аргументу. У стандарті мови є також процедури керування, призначені для змінення порядку виконання операторів керування.

Таблиця 10.6

Процедури передачі керування

Назва	Призначення
Break	Перериває виконання циклу
Continue	Починає нове повторення циклу
Exit	Перериває виконання поточного блоку
Halt	Зупиняє виконання програми й повертає керування операційній системі

⇒ **Стандартні процедури та функції для роботи з рядками**

Оскільки оброблення рядків виконується практично в кожній програмі, яка працює з візуальним середовищем, то стандартний модуль System має набір процедур і функцій (табл. 10.7), які значно полегшують цей процес. Усі вони можуть застосовуватись як до коротких, так і до довгих рядків.

Таблиця 10.7

Процедури та функції для роботи з рядками

Назва	Призначення
Concat(S1, S2, ..., Sn): string	Повертає рядок, одержаний у результаті зчеплення рядків S1, S2, ..., Sn. За принципом роботи функція Concat аналогічна операції зчеплення рядків (+)
Copy(S: string , Index, Count: Integer): string	Копіює з рядка S підрядок Count символів починаючи з позиції Index
Delete(var S: string , Index, Count: Integer)	Видаляє з рядка S Count символів починаючи з позиції Index
Insert(Source: string ; var S: string , Index: Integer)	Вставляє рядок Source у рядок S починаючи з позиції Index
Length(S: string): Integer	Повертає реальну довжину рядка S у символах
SetLength(var S: string ; NewLength: Integer)	Установлює для рядка S нову довжину NewLength

Закінчення табл. 10.7

Назва	Призначення
Pos(Substr, S: string): Integer	Виявляє першу появу підрядка Substr у рядку S. Повертає номер тієї позиції, де є перший символ підрядка Substr. Якщо в рядку S не знайдено підрядок Substr, результат дорівнює 0
AnsiLowerCase(const S: string): string	Перетворює заголовні літери рядка S на малі з урахуванням мови, встановленої в середовищі Windows
AnsiUpperCase(const S: string): string	Перетворює малі літери рядка S на заголовні з урахуванням мови, встановленої в середовищі Windows

Приклади:

Вираз	Значення S
S:=Concat('Object ', 'Pascal');	'Object Pascal'
S:=Copy('Програмування',3,3);	'огр'
S := 'Програмування в'; Delete(S, 2, 3);	'Прамування в'
S := 'Програмування в'; Insert('Delphi ', S, 15)	'Програмування Delphi в'

Приклади:

Вираз	Результат
Pos('pa', 'Програмування в')	5
Length ('Програмування в')	15

➡ **Перетворення типів даних**

Використання візуальних компонентів передбачає необхідність перетворювати велику кількість типів даних. Наведемо список процедур і функцій у середовищі Delphi, за допомогою яких виконуються ці перетворення (табл. 10.8).

Таблиця 10.8

Процедури і функції перетворення типів даних

Назва	Призначення
IntToStr(Value: Integer): string	Перетворює ціле число Value на рядок
StrToInt(const S: string): Integer	Перетворює рядок на ціле число

Назва	Призначення
<code>FloatToStr(Value: Extended): string;</code>	Перетворює дійсне число <code>Value</code> на рядок
<code>StrToFloat(const S: string): Extended;</code>	Перетворює рядок на дійсне число
<code>Str(X [:Width [:Decimals]], var S: string)</code>	Перетворює числове значення величини <code>X</code> на рядок <code>S</code> . Необов'язкові параметри <code>Width</code> і <code>Decimals</code> є цілочисловими виразами. Значення <code>Width</code> задає ширину поля результуючого рядка. Для дійсних чисел <code>Decimals</code> задає кількість символів у дробовій частині
<code>Val(S: string, var V; var Code: Integer)</code>	Перетворює рядок <code>S</code> на величину цілого або дійсного типу й поміщує результат у змінну <code>V</code> . Якщо під час операції перетворення помилки не виявлено, значення змінної <code>Code</code> дорівнює нулю; якщо помилку виявлено (у рядку є недопустимі символи), <code>Code</code> містить номер позиції першого помилкового символу, а значення <code>V</code> не визначене
<code>StrToBool(const S: string): Boolean</code>	Перетворює рядок на булеве значення
<code>BoolToStr(B: Boolean; UseBoolStrs: Boolean = False): string</code>	Перетворює булеве значення на рядок
<code>DateTimeToStr(const DateTime: TDateTime): string</code>	Перетворює значення дати й часу на рядок
<code>StrToDateTime(const S: string): TDateTime;</code>	Перетворює рядок на значення дати й часу

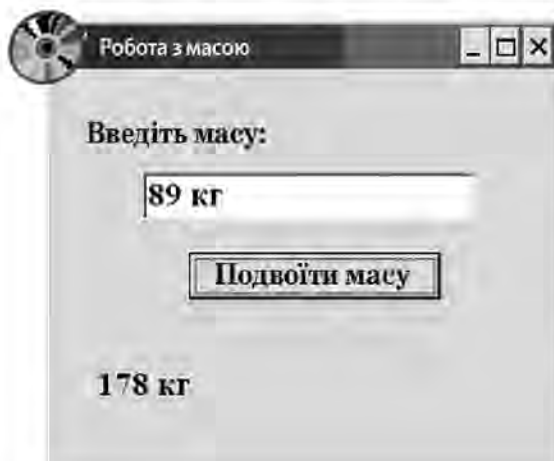


Рис. 10.3. Подвоєння маси

Розглянемо приклади використання функцій, призначених для роботи з рядками.

Приклад 13. В `Edit` на формі ввести масу у вигляді рядка у форматі «числове значення», пробіл і «кг», наприклад: «89 кг». Натисканням на кнопку в `Label` вивести подвоєну масу, наприклад: «178 кг».

Розв'язання. Розробимо форму відповідно до рис. 10.3.

Після записування програмного коду в метод `Button1Click` програма набуде такого вигляду:

```

unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;

implementation
{$R *.dfm}

procedure TForm1.Button1Click(Sender: TObject);
var
  s, st :string;
  ves:real;
begin
  s:=Edit1.Text;           //прочитати рядок
  st:=copy(s, 1, pos(' ',s) - 1); //копіюємо масу
  delete(s, 1, pos(' ',s) - 1);   //видаляємо її в рядку
  ves:=StrToFloat(st); //перетворюємо масу на дійсне числове
                        //значення
  ves:=2 * ves; //збільшуємо у два рази
  s:=FloatToStr(ves) + s;   //перетворюємо масу на рядок
  label2.Caption:=s;
end;
end.

```

Приклад 14. Ввести на форму довільний рядок символів (слова можна розділяти будь-якою кількістю пробілів). Вивести всі слова рядка, які є паліндромами. (Паліндромом називається слово або фраза, що мають однаковий зміст, якщо читати їх зліва направо і справа наліво, наприклад: «потоп», «кіт утік».)



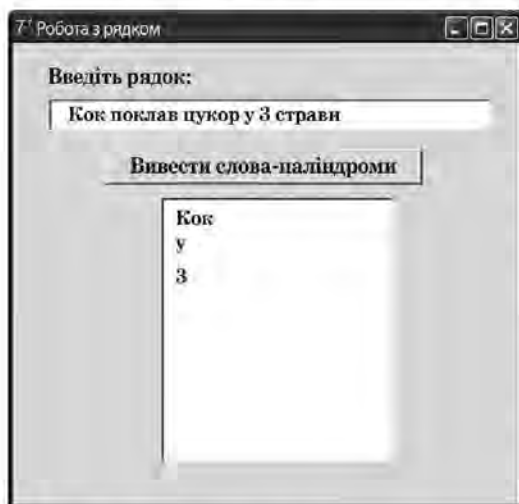


Рис. 10.4. Пошук у заданому рядку слів-паліндромів

Розв'язання. Інтерфейс задачі подано на рис. 10.4. Програма матиме такий вигляд:

```

unit Unit1;
interface
uses Windows, Messages, SysUtils,
Variants, Classes, Graphics,
Controls, Forms, Dialogs,
StdCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Button1: TButton;
    Memo1: TMemo;
    Label1: TLabel;
  end;
var
  Form1: TForm1;
implementation
{$R *.dfm}

function Udal_probел (st:string):string;
// функція – видалення зайвих пробілів
begin
  st:=st + ' '; // додати в кінець рядка пробіл
  while pos(' ', st) > 0 do // якщо є подвійні пробіли
    delete(st,pos(' ', st), 1); // видалити один із них
  if st[1]=' ' then // якщо перший символ – пробіл
    delete(st, 1, 1); // видалити його
  Result:=st; // повернути результат
end;

function Palindrom (st:string):boolean;
// функція – перевірка слова-паліндрома
var
  i: integer;

```

```

begin
    Result:=true;                                // слово є паліндромом
    for i:=1 to length(st) div 2 do              //перевіряємо пари символів
        if AnsiUpperCase(st[i])<>AnsiUpperCase(st[length(st) - i + 1]) then
            Result:=false;                       // якщо символи нерівні
end;

procedure TForm1.Button1Click(Sender: TObject);
var
    s, slovo: string;
    i: integer;
begin
    s:=Edit1.Text;                               // прочитати рядок з Edit1
    s:=Udal_probел(s);                           // видалити зайві пробіли
    while s<>' ' do                               // доки в рядку є слова
    begin
        slovo:= copy(s, 1, pos(' ',s) - 1); // копіюємо слово
        delete(s, 1, pos(' ', s));           // видаляємо слово разом із пробілом
        if Palindrom(slovo) then             // якщо слово-паліндром
            memo1.Lines.Add(slovo);          // вивести паліндром у Мемо
    end;
end;
end.

```

Запропонований алгоритм виділення слів у рядку можна використовувати щоразу, коли необхідно здобути доступ до окремих слів оброблюваного рядка.

Розглянемо інший підхід у роботі з текстовою інформацією.

Приклад 15. Створити програму, що змушує послідовно «осипатись» рядок, розташований у верхній частині форми. Падіння літер здійснити за допомогою компонента TTimer, робота якого закінчується із закінченням рядка.

Розв'язання. На рис. 10.5 наведено приклад падіння символів рядка. Щоб рядок зорово не скорочувався, доцільно встановити в Label шрифт Courier New.

Програма матиме такий вигляд:

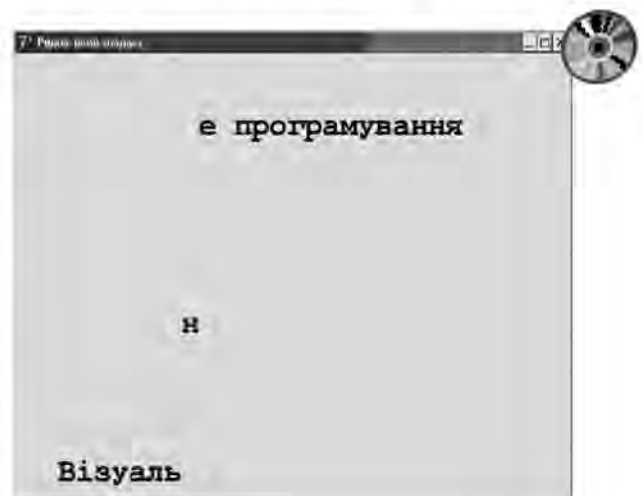


Рис. 10.5. Рядок, який «падає»

```

unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, ExtCtrls, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Timer1: TTimer;
    procedure Timer1Timer(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  f: boolean=true;
  k: integer=1;
  t,l: integer;
implementation
{$R *.dfm}

procedure TForm1.Timer1Timer(Sender: TObject);
var
  c:char; // символ, який падає
begin
  if f then
  begin
    c:=label1.caption[k]; // скопіювати 1 символ і замінити
                        // його пробілом
    label1.caption:=copy(label1.caption,1,k-1)+' ' +
copy(label1.caption,k+1,255);
    label2.Caption:=c; // записати в другий label
    if label1.Caption='' then // якщо нічого не залишилося –
                        // закінчити
      Timer1.Enabled:=false;
    f:=false;
    k:=k+1;
    label2.top:=t; //відновити позиції
    label2.Left:=l;
  end
  else

```

```

begin
    label2.Top:=label2.Top+5;           // посунути вниз
    if label2.top>=label3.Top then      // якщо досяг низу
    begin
        label3.Caption:=label3.Caption+label2.Caption;
        // перекинути символ
        l:=l+label2.Width;
        f:=true;
    end;
end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    doublebuffered:=true; // увімкнути подвійну буферизацію
    t:=label2.top;        // запам'ятати позиції другого label
    l:=label2.left;
end;

end.

```

➔ Генератор псевдовипадкових чисел

Усі сучасні ігри й додатки для створення різних подій часто використовують механізм випадкових чисел. Це може бути задавання вихідних даних, щоб не вводити їх вручну, або випадкової генерації лабіринту; вибір ігрового персонажа або хаотичного руху частинок чи мікроорганізмів. Прикладів таких подій доволі багато, і всі вони об'єднані одним — тим, що ці події мають випадковий характер. Для реалізації подібних подій використовується генератор випадкових чисел, хоча насправді генерація абсолютно випадкових чисел неможлива, тому буде точнішим називати ці числа псевдовипадковими. Отже, генератор псевдовипадкових чисел (ГПВЧ, англ. *Pseudorandom number generator*) — це алгоритм, що генерує послідовність чисел, елементи якої майже незалежні один від одного й підпорядковуються заданому розподілу (зазвичай рівномірному).

Історія генераторів випадкових чисел почалася з досліджень Джона фон Неймана. 1946 р. він запропонував алгоритм генерації випадкових чисел. Суть його така: необхідно взяти будь-яке N -значне число й піднести його до квадрата, отримавши $2N$ -значне число (у разі потреби його слід доповнити зліва нулями до $2N$ -значного числа), далі із запису цього числа взяти серединну послідовність із N цифр. У такий спосіб одержується наступне число в послідовності. Наприклад, якщо взяти як початкове число значення 5928 ($N=4$), одержимо таку послідовність чисел: 1411 ($5928^2 = 35\,141\,184$, серединні чотири цифри 1411), 9909 ($1411^2 = 1\,990\,921$, доповнюємо число зліва одним нулем і беремо чотири

серединні цифри 9909), 1882, 5419... Описаний алгоритм одержання послідовності випадкових чисел називається методом середніх квадратів. Цей метод має один суттєвий недолік: якщо введена послідовність випадково міститиме нульові значення, генератор перебудується в режим генерації нулів.

Наступний відомий алгоритм генерації випадкових чисел розроблений Д. Лемером (1949 р.), він має назву «лінійний конгруентний метод». Його ідея така: обираються три початкових числа a , c і m , а також певне початкове число послідовності X_0 . Далі для генерації послідовності використовується формула $X_{n+1} = (aX_n + c) \bmod m$.

Наприклад, стандартний генератор випадкових чисел у Delphi використовує такі значення: $a=134\,775\,813$, $c=1$, $m=2^{32}$; початкове значення X_0 може задавати користувач за допомогою процедури Randomize на основі показань системного годинника (табл. 10.9).

Таблиця 10.9

Процедури і функції для роботи з псевдовипадковими числами

Назва	Призначення
Random	Повертає випадкове дійсне число в діапазоні $0 \leq X < 1$
Random(N)	Повертає випадкове ціле число в діапазоні $0 \leq X < N$
Randomize	Заново ініціалізує вбудований генератор випадкових чисел новим значенням, одержаним від системного таймера

Розглянемо роботу генератора псевдовипадкових чисел на прикладі такої програми: на формі розташовані кнопка й таймер (невидимий компонент TTimer, закладка System). Ця кнопка щосекунди ($\text{Timer1.Interval}=1000$) змінює своє положення (але не виходить за межі форми).

Для розв'язання цієї задачі помістимо код, наведений нижче, в обробник події OnTimer компонента Timer1:

```
-----
procedure TForm1.Timer1Timer(Sender: TObject);
begin
    Button1.Left:=Random(ClientWidth-Button1.Width);
    Button1.Top:=Random(ClientHeight-Button1.Height)
end;
-----
```

Кнопка дійсно «стрибає» з місця на місце. Однак якщо запустити програму кілька разів і поспостерігати за переміщенням кнопки, то можна помітити: вона кожного разу потрапляє на одні й ті самі точки.

Щоб розв'язати цю проблему, потрібно помістити в обробник OnCreate форми виклик процедури Randomize:

```
-----
procedure TForm1.FormCreate(Sender: TObject);
begin
    Randomize
end;
-----
```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Перелічіть арифметичні функції, що використовуються в роботі з числовими змінними. Наведіть приклади.
2. Перелічіть тригонометричні функції. Наведіть приклади.
3. Перелічіть функції виділення цілої та дробової частин. Наведіть приклади.
4. Які функції Delphi призначені для оброблення рядкових даних?
5. Перелічіть процедури, що використовуються в роботі з рядковими даними.
6. Перелічіть функції перетворення типів даних.
7. За допомогою методу середніх квадратів одержте шість значень випадкових чисел за умови, що початкове число дорівнює 1234.
8. За допомогою лінійного конгруентного методу одержте п'ять значень випадкових чисел за умови, що $a=109$, $c=1$, $m=225$, а початкове значення $X_0=16$.



ПРАКТИЧНІ ЗАВДАННЯ

1. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів і розділових знаків. Усі символи, які йдуть за знаками «.», «!», «?», зробити заголовними. Вивести одержаний рядок.
2. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів і розділових знаків. Замінити в рядку всі коми на крапки, а крапки — на коми й вивести одержаний рядок.
3. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів і розділових знаків. Видалити в рядку всі літери на позначення голосних і вивести одержаний рядок.
4. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів. Визначити в рядку найдовше й найкоротше слова та вивести їх.
5. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів. Визначити й вивести з рядка слова, запис яких являє собою ціле число без знака.
6. Організувати на формі введення рядка, що складається зі слів, розділених довільною кількістю пробілів. Сформувати новий рядок, у якому слова йдуть у зворотному порядку. Вивести одержаний рядок.



Практична робота № 11

Створення й застосування користувацьких підпрограм, а також вбудованих процедур і функцій у графіці

Мета роботи: освоїти можливості Delphi в рисуванні найпростіших графічних зображень.

Завдання. Побудувати на формі зображення координатних осей, конверта і піраміди, графіка функції, використовуючи графічні можливості Delphi.

Теоретичні відомості

У Windows для позначення канви, на якій виконується рисунок, використовується *термін контекст* пристрою (DC). Взаємодія з контекстами пристроїв на рівні викликів функцій API може виявитися доволі складною. Тому спочатку потрібно одержати в системі Windows дескриптор контексту пристрою (посилання на формальний опис можливостей пристрою), після чого вибрати в нього потрібні вам об'єкти (пера, пензлі або шрифти). Тільки потім можна на ньому щось рисувати. Після закінчення рисування, перш ніж видалити контекст пристрою, потрібно звільнити обрані в нього об'єкти. В іншому випадку пам'ять, яка використовується у вашому додатку, ніколи не буде повернута системі.

Наявний клас TCanvas із бібліотеки візуальних компонентів значно полегшує роботу з контекстами пристроїв. Як властивість (*property*) він включений у багато компонентів.

Розглянемо основні властивості та методи класу TCanvas (табл. 10.10 і 10.11).

Таблиця 10.10

Основні властивості класу TCanvas

Властивість	Опис
Brush	Задає колір заливки та шаблон у ході зафарбовування фігур
ClipRect	Являє собою поточну прямокутну область відсікання канви. Усі операції графічного виведення обмежені заданим прямокутником. Властивість тільки для читання
CopyMode	Визначає, як відбуватиметься копіювання (метод CopyRect) на дану канву зображення з іншого місця: один до одного, з інверсією зображення тощо
Font	Містить шрифт, який клас TCanvas використовує у виведенні тексту (метод TextOut)
Handle	Дескриптор (HDC) канви. Ця властивість дає можливість прямо викликати функції API Windows
Pen	Визначає стиль і колір проведених ліній
PenPos	Задає координати x і y для чергової операції рисування
Pixels	Являє собою двовимірний масив пікселів об'єкта Canvas

Таблиця 10.11

Основні методи класу TCanvas

Властивість	Опис
Arc	Рисує на канві поточним пером сегмент еліпса
BrushCopy	Рисує бітову матрицю з прозорим тлом
CopyRect	Копіює на канву частину зображення
Draw	Копіює зображення з пам'яті на канву
Ellipse	Рисує поточним пером еліпс і зафарбовує його поточним пензлем
FloodFill	Здійснює заливку області канви поточним пензлем
LineTo	Проводить поточним пером лінію з даної точки в точку з координатами x і y
MoveTo	Встановлює поточне положення пера
Pie	Рисує на канві сектор еліпса
Polygon	За допомогою масиву об'єктів типу TPoint будує багатокутник і зафарбовує його поточним пензлем
Polyline	За допомогою масиву об'єктів типу TPoint рисує поточним пером ламану лінію. Автоматичного замикання контуру при цьому не відбувається
Rectangle	Рисує на канві поточним пером прямокутник і зафарбовує його поточним пензлем
RoundRect	Рисує зафарбований прямокутник із закругленими кутами
StretchDraw	Копіює бітову матрицю з пам'яті на канву. При цьому відповідно до цільової прямокутної області відбувається масштабування (розтягнення або стиснення) растрових зображень
TextExtent	Повертає висоту й ширину рядка Text у пікселях. Ураховується поточний шрифт канви
TextHight	Повертає висоту рядка Text у пікселях. Ураховується поточний шрифт канви
TextOut	Виводить рядок Text за допомогою поточного шрифту в заданому місці канви
TextRect	Виводить текст із відсіканням. При цьому частина тексту, що потрапила за межі прямокутної області відсікання, стає невидимою

Перелічені властивості й методи становлять лише малу частину можливостей контекстів пристроїв Windows. Але навіть ця мала частина може задовольнити 80 % потреб, що виникають зазвичай у ході роботи з графікою.

В інтерфейсі графічних пристроїв (GDI) Windows є велика кількість типів об'єктів, які впливають на роботу контекстів пристроїв. Найчастіше з об'єктів Graphics Device Interface використовуються пера, пензлі та шрифти. Менш популярні графічні об'єкти — палітри, бітові матриці й області відсікання.

Розглянемо можливості вкладеного об'єкта «перо» (*pen*).

За допомогою пер можна рисувати різноманітні лінії: як окремі, що сполучають одну точку з іншою, так і ті, що обмежують різні геометричні фігури: прямокутники, еліпси й многокутники.

Для доступу до пера використовується властивість *Pen* класу *TCanvas*, що являє собою об'єкт типу *TPen*. У табл. 10.12 перелічено властивості класу *TPen*. Методів і подій, які заслуговують на згадування, у цьому класі немає.

Таблиця 10.12

Властивості класу *TPen*

Властивість	Опис
<i>Color</i>	Задавання кольору лінії
<i>Handle</i>	Являє собою дескриптор пера (HPEN). Використовується у випадках прямих викликів відповідних функцій API
<i>Mode</i>	Визначає спосіб прорисовки ліній (нормальний, інверсний, з виключним або (<i>xor</i>) тощо)
<i>Style</i>	Визначає стиль лінії (суцільний, пунктирний, штриховий, штрихпунктирний, прозорий і т. д.)
<i>Width</i>	Задавання товщини лінії в пікселях

Здебільшого використання цих властивостей не викликає ніяких труднощів.

Хід роботи

1. Запустимо середовище Delphi, створимо новий додаток, розмістимо на формі компонент «кнопка». Напишемо невелику програму, що виводить на екрані дві лінії — горизонтальну й вертикальну. При цьому лінії мають бути пунктирними (стилі *psDash* і *psDot*), а їхній колір — синім. Позначимо координати початку й кінця ліній (рис. 10.6).

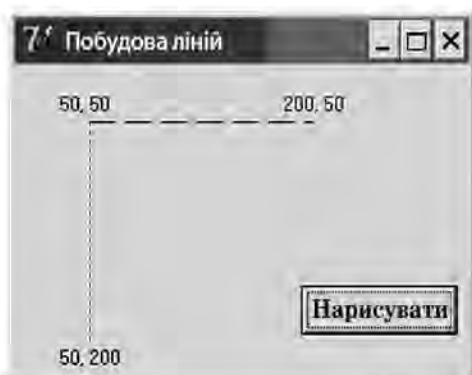


Рис. 10.6. Виведення ліній на формі

2. Використовуючи роботу з об'єктом Canvas, задамо стилі, колір ліній і проведемо їх. Оформимо виведення значень координат.
3. В обробник події OnClick помістимо код:

```
-----
procedure TForm1.Button1Click(Sender: TObject);
begin
    Canvas.Pen.Color := clBlue;
    Canvas.Pen.Style := psDash;           // вибір стилю лінії
    Canvas.MoveTo(50, 50);                // перехід робочої точки
    Canvas.LineTo(200, 50);               // провести лінію в точку
                                           // (200,50)

    Canvas.Pen.Style := psDot;            // вибір стилю лінії
    Canvas.MoveTo(50, 50);                // перехід робочої точки
    Canvas.LineTo(50, 200);               // провести лінію в точку
                                           // (50, 200)

    canvas.TextOut(30,30,'50, 50');       // виведення тексту
                                           //в зазначених координатах
    canvas.TextOut(180, 30,'200, 50');
    canvas.TextOut(30, 200,'50, 200');
end;
-----
```

У попередніх завданнях ми виводили текст і рисунок у вікні програми. Однак якщо нам необхідно сховати вікно, а потім знову його активізувати, ми виявимо, що рисунок на формі зник. Це відбувається тому, що наш рисунок поки що має тимчасовий характер. Щоб зробити його постійним, слід помістити наведений вище код в обробник події OnPaint форми. Тепер у разі потреби перерисувати вікно генеруватиметься подія OnPaint, і рисунок буде відновлено.

Штриховий і пунктирний стилі можна використовувати тільки в роботі з пером завтовшки 1 піксел. За допомогою стилю psClear ви можете усунути лінії, що їх Windows рисує по зовнішній межі таких об'єктів, як, наприклад, прямокутники, еліпси й зафарбовані многокутники.

4. Збережемо проект на диску.
5. Створимо новий додаток для роботи з пензлями (brush).

Теоретичні відомості

Пензлі визначають спосіб зафарбовування фігур. За допомогою поточного пензля заповнюється внутрішня область кожної з нарисованих вами фігур: еліпсів, прямокутників, многокутників тощо. Пензель може

рисувати штрихові лінії або бітову матрицю. Його зовнішній вигляд можна контролювати за допомогою властивості `Brush` класу `TCanvas`, що є, у свою чергу, об'єктом класу `TBrush`. Як і `TPen`, `TBrush` не містить методів і подій для програмування.

Властивості класу `TBrush` перелічено в табл. 10.13.

Таблиця 10.13

Властивості класу `TBrush`

Властивість	Опис
<code>Bitmap</code>	Ідентифікує бітову матрицю, що використовується як тло пензля. У Windows 95 бітова матриця повинна була мати розміри 8×8
<code>Color</code>	Встановлює колір пензля
<code>Handle</code>	Являє собою дескриптор пензля (<code>HBRUSH</code>). Використовується у випадках прямих викликів функцій API
<code>Style</code>	Визначає стиль пензля (суцільний, прозорий або одна з можливих штриховок)

Властивість `Style` за замовчуванням має значення `bsSolid` (суцільна заливка). Якщо в зафарбовуванні геометричних фігур ви хочете використати штриховку, то властивості `Style` слід присвоїти один із таких стилів: `bsHorizontal`, `bsVertical`, `bsFDiagonal`, `bsBDiagonal`, `bsCross` або `bsDiagCross`.

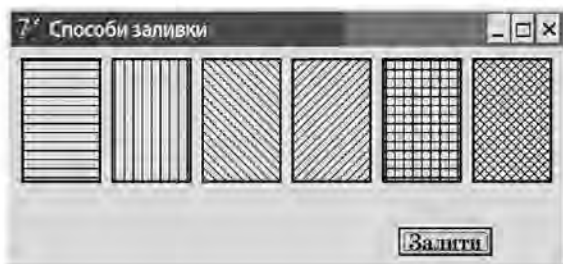


Рис. 10.7. Способи заливки

6. Продемонструємо перелічені способи заливки (у зазначеному порядку) за допомогою невеликої програми, що рисує прямокутники. Внесемо на форму кнопку. (Інтерфейс завдання подано на рис. 10.7.)
7. В обробник події `OnClick` цієї кнопки впишемо програмний код:

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    canvas.pen.Width:=2;
    canvas.Brush.Color:=clblack;
    canvas.Brush.Style:=bsHorizontal;
    canvas.rectangle(10,10,80,120);
    canvas.Brush.Style:=bsVertical;
    canvas.rectangle(90,10,160,120);
    canvas.Brush.Style:=bsFDiagonal;
    canvas.rectangle(170,10,240,120);

```

```

    canvas.Brush.Style:=bsBDiagonal;
    canvas.rectangle(250,10,320,120);
    canvas.Brush.Style:=bsCross;
    canvas.rectangle(330,10,400,120);
    canvas.Brush.Style:=bsDiagCross;
    canvas.rectangle(410,10,480,120);
end;

```

Коли ви використовуєте пензель-штриховку, його властивість `Color` визначає колір штрихових ліній. У процесі зафарбовування графічних об'єктів за допомогою такого пензля VCL автоматично встановлює режим фону в `Transparent` (наприклад прозорий). Це означає, що колір фону пензля буде збігатися з кольором вікна, на якому нарисований графічний об'єкт.

8. Збережемо проект на диску.

9. Створимо новий додаток. Трохи ускладнимо завдання. Намалюємо на формі конверт і піраміду. Приклад того, як мають виглядати конверт і піраміда у вікні чинної програми, показано на рис 10.8.



10. За аналогією з попереднім завданням впишемо в кнопку обробник події:

Рис. 10.8. Побудова конверта й піраміди

```

procedure TForm1.Button1Click(Sender: TObject);
var
    i:integer;
begin
    Canvas.Pen.width := 2;
    Canvas.Rectangle(30,30,180,130);
    Canvas.MoveTo(30,30);
    Canvas.LineTo(105,80);
    Canvas.LineTo(180,30);
    for i:=0 to 4 do
        Canvas.Rectangle(300-20*i,30+20*i,400+20*i,50+20*i);
    end;

```

11. Збережемо проект на диску.

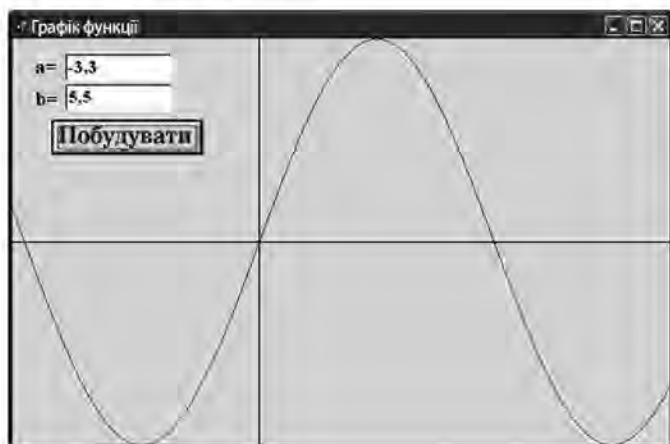


Рис. 10.9. Побудова графіка функції $y = \sin(x)$

12. Створимо новий додаток. Ще раз ускладнимо завдання. Побудуємо на формі графік функції $y = \sin(x)$.

У ході побудови потрібно ввести ліву і праву межі інтервалу (дійсні числа a і b), виконати масштабування графіка функції по осях Ox і Oy (рис. 10.9).

Програма матиме такий вигляд:

```
-----
unit Unit1; {Графік функції}
interface
uses Windows, Messages, SysUtils, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Edit1: TEdit;
    Edit2: TEdit;
    Button1: TButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
implementation
{$R *.DFM}

function f(x:real):real; // функція побудови
begin
  result:=sin(x);
end;
```

```

procedure TForm1.Button1Click(Sender: TObject);
var
  x1,x2,max,min,shagx,shagy:real;
  i,sh:integer;
begin
  x1:=strtofloat(edit1.text);           // ліва межа
  x2:=strtofloat(edit2.text);           // права межа

  shagx:=(x2-x1)/form1.ClientWidth;     // масштабування
                                         // по осі Oх
  min:=f(x1);                           // початкові значення для мінімального
  max:=f(x1);                           // і максимального значень функції
  for i:=1 to form1.ClientWidth do
  begin
    if f(x1+i*shagx)<min then           // пошук мінімального
      min:=f(x1+i*shagx);              // і максимального
                                         // значень функції

    if f(x1+i*shagx)>max then
      max:=f(x1+i*shagx);
  end;
  shagy:=form1.clientheight/(max-min);   // масштабування по осі Oy

  if x1*x2 < 0 then                   // малювання осі Oy
  begin
    canvas.moveto(round(abs(x1)*form1.clientwidth/
      (x2-x1)),0);
    canvas.lineto(round(abs(x1)*form1.clientwidth/
      (x2-x1)),form1.clientheight);
  end;
  if min*max<0 then                 // малювання осі Oх
  begin
    canvas.moveto(0,form1.clientheight -
      round(abs(min)*form1.clientheight/(max-min)));
    canvas.lineto(form1.clientwidth,form1.clientheight -
      round(abs(min)*form1.clientheight/(max-min)));
  end;

  canvas.pen.Color:=rgb(255,0,0);       // колір графіка функції
  canvas.moveto(1,round(form1.clientheight-(f(x1+1*shagx)-
    min)*shagy) );
  for i:=2 to form1.ClientWidth do   // побудова графіка
    canvas.lineto(i,round(form1.clientheight-
      (f(x1+i*shagx)-min)*shagy) );
end;
end.

```

☐ ПРАКТИЧНЕ ЗАВДАННЯ

Нарисувати на формі один із запропонованих прапорців (рис. 10.10). Кольори елементів прапорця узгодити з викладачем. У ході рисування використати умовні оператори та цикли, оскільки надалі можлива модифікація прапорця — збільшення кількості його елементів (кружків, квадратиків тощо).

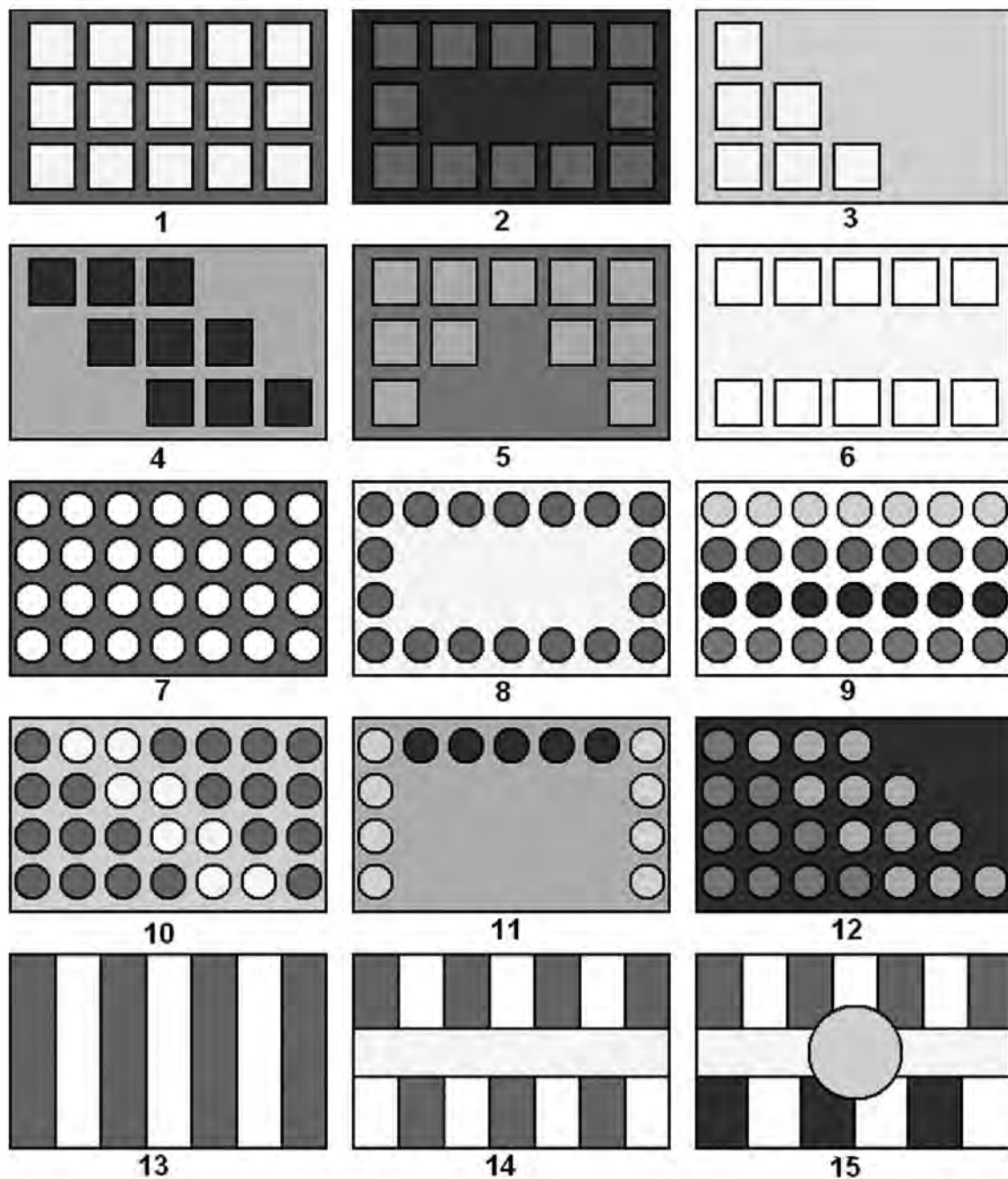


Рис. 10.10. Зразки прапорців для рисування за допомогою об'єкта Canvas



Список прізвищ у класному журналі та сторінка з оцінками, таблиця результатів чемпіонату з футболу й інформаційна панель, де показано час прибуття й відправлення електричок,— усе це масиви, тобто структури даних, у яких потрібний рядок, стовпчик або окремий елемент можна знайти за їхнім порядковим номером.

У цьому розділі розповідається, як працювати з масивами в мові Delphi, починаючи з введення-виведення відносно простих одновимірних масивів. Показано, як розраховувати для них такі базові характеристики, як сума, добуток, найменше і найбільше значення елементів. Розглянуто питання пошуку елементів у масивах та їх сортування.

11.1

Поняття масиву. Оголошення одновимірного масиву. Індексація елементів. Введення даних у масив і відображення його вмісту

⇒ Поняття масиву. Оголошення одновимірного масиву. Індексація елементів

Масивом називається впорядкована індексована сукупність однотипних елементів, що мають спільне ім'я. Елементами масиву можуть бути дані різних типів, включно зі структурованими. Кожен елемент одновимірного масиву однозначно визначається ім'ям та індексом (для багатовимірного масиву — індексами).

Масиви зручно використовувати там, де доводиться працювати з великою кількістю однотипних даних, при цьому передбачено кількразове їх оброблення.

Лінійний (одновимірний) масив — масив, у якого в описі задано тільки один індекс. Одновимірні масиви часто називають векторами, бо вони являють собою скінченну послідовність пронумерованих елементів.

Загальний вигляд опису лінійного (одновимірного) масиву:

```
array[<тип_індексів>] of <тип_елемента>;
```

Найчастіше опис здійснюється в такий спосіб:

```
array[<нижня_межа>..<верхня_межа>] of <тип_елемента>;
```

Наприклад:

```
var
```

```
  a: array [byte] of real;
```

```
  b: array [1..5] of integer;
```

```
  c: array [0..9] of boolean;
```

Зверніть увагу на те, що нижня і верхня межі індексів у масиві зазначаються довільно. Межі діапазону можуть бути навіть від'ємними. Наприклад, можливий такий запис: `[-3..5]`. Головне, щоб права межа діапазону була більшою за ліву. На практиці зазвичай доводиться нумерувати елементи з 1 або з 0. Програмістові-початківцю краще вести нумерацію з одиниці, бо в лічбі ми використовуємо саме таку послідовність значень. Однак, із погляду зберігання даних і деяких алгоритмів, варто починати нумерацію з нуля.

Нумерувати елементи масиву можна не тільки цілими числами. Будь-який порядковий тип даних (перелічуваний, інтервальний, символний, логічний, а також довільний, створений на їхній основі) може виконувати роль індексу. При цьому сукупний розмір масиву не може перевищувати 2 Гбайт. Таким чином, допускаються такі описи масиву:

```
-----
var
d: array [char] of real;
e: array ['a'..'z'] of integer;
f: array [boolean] of char;
-----
```

Наведемо хибні описи масиву:

```
-----
var
g: array [integer] of integer;
h: array [real] of byte;
-----
```

Масиви належать до структур прямого доступу. При цьому вміст масиву зберігається в неперервній ділянці пам'яті. Це означає, що можна прямо (не перебираючи всіх попередніх) звернутися до будь-якого елемента масиву, що цікавить нас. Для цього необхідно вказати ім'я масиву й індекс (індекси), узятий у квадратні дужки:

```
a[3] := 5.4;
b[2] := -2;
c[0] := true;
d['1'] := 1.567;
e['b'] := 7;
f[true] := 'Y';
```

Подібний масив можна сприймати як таблицю з одним рядком і кількома комірками (табл. 11.1).

Таблиця 11.1

Розташування масиву *b* цілих чисел у пам'яті

<i>b</i> [1]	<i>b</i> [2]	<i>b</i> [3]	<i>b</i> [4]	<i>b</i> [5]
Базова адреса	Базова адреса + 4 байти	Базова адреса + 8 байт	Базова адреса + 12 байт	Базова адреса + 16 байт

Усього під масив, наведений у табл. 11.1, у пам'яті буде виділено 20 байт — у наведеному прикладі оголошено масив *b* цілих чисел типу *integer*, кожній комірці (елементу) масиву буде виділено по 4 байти пам'яті. Усі ці елементи відповідають звичайним змінним такого ж типу.

⇒ Введення даних у масив і відображення його вмісту

Програмісти, що мають справу зі старими мовами програмування або працюють у консольному додатку, введення даних у масив зазвичай оформлюють у вигляді нескладного циклу з параметром *FOR*, наприклад:

```

var
  a: array [1..10] of integer;
  i: integer;
begin
  for i:= 1 to 10 do
  begin
    write('Ввести елемент a[' , i , ']=');
    readln(a[i]);
  end;
end.

```

Такий підхід для візуального програмування, в основі якого лежить подієве керування, неприйнятний. Простіше кажучи, у нашій програмі замість циклу *FOR* має використовуватися послідовність дій (введення даних; натискання кнопки) самого користувача.

Приклад 16. Розробити програму, що реалізовує заповнення на формі цілочислового масиву, який складається з 10 елементів.

Розв'язання. Розробимо інтерфейс програми відповідно до рис. 11.1.

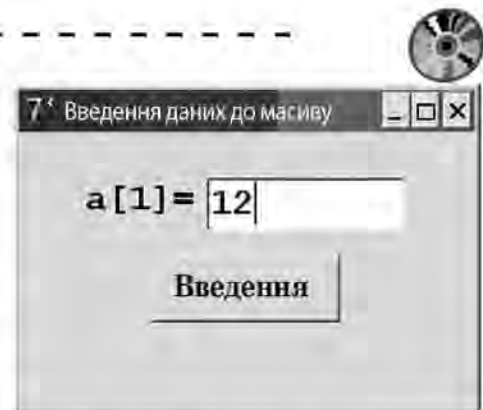


Рис. 11.1. Введення елементів масиву за допомогою компонентів *Label*, *Edit* та *Button*

Програма матиме такий вигляд:

```
-----  
unit Unit1;  
interface  
uses  
    Windows, Messages, SysUtils, Variants, Classes, Graphics,  
    Controls, Forms, Dialogs, StdCtrls;  
type  
    TForm1 = class(TForm)  
        Edit1: TEdit;  
        Button1: TButton;  
        Label1: TLabel;  
  
        procedure Button1Click(Sender: TObject);  
private  
    { Private declarations }  
public  
    { Public declarations }  
end;  
var  
    Form1: TForm1;  
    a: array [1..10] of integer;  
        // опис масиву з 10 цілих чисел  
    i: integer=1;        // поточний індекс масиву  
  
implementation  
{$R *.dfm}  
  
procedure TForm1.Button1Click(Sender: TObject);  
begin  
    a[i]:=strtoint(edit1.text); //записування значення з поля  
    label1.Caption:='a['+inttostr(i+1)+']=';  
    i:=i+1;                    // збільшення поточного індексу  
    edit1.Clear;               // очистити поле введення  
    edit1.SetFocus;            // перевести фокус  
    if i=11 then               // якщо індекс=11, то закінчити введення  
    begin  
        label1.visible:=false;  
        edit1.enabled:=false;  
        button1.Enabled:=false;  
    end;  
end;  
end.  
-----
```

У консольному додатку виведення вмісту масиву можна організувати так:

```
-----
var
  a: array [1..10] of integer;
  i: integer;
begin
    // введення та оброблення масиву...
    for i:= 1 to 10 do // цикл для виведення масиву
        writeln('Елемент a[' , i , ']= ' , a[i]);
    end.
-----
```



Приклад 17. Організувати виведення одновимірного масиву за допомогою візуальних компонентів.

Розв'язання. Розмістимо на формі компоненти Button, Label та Memo, як показано на рис. 11.2. Програма матиме такий вигляд:

```
-----
unit Unit1;
interface
uses Windows, Messages, SysUtils,
  Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Label1: TLabel;
    Memo1: TMemo;

    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a: array [1..10] of integer=(0,3,86,20,27,67,31,16,37,42);
  i: integer;
```

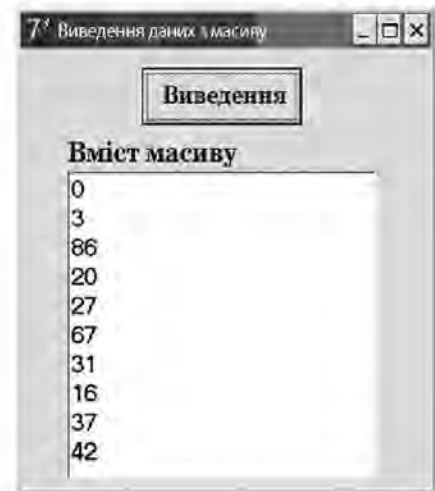


Рис. 11.2. Виведення одновимірного масиву за допомогою компонентів Button, Label та Memo

```

implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
begin
    for i:=1 to 10 do
        memo1.Lines.Add(inttostr(a[i]));    //виведення в поле Мемо
    end;
end.

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Наведіть приклади задач, які можна розв'язати за допомогою одновимірних масивів.
2. Дайте опис одновимірного масиву в загальному вигляді й у конкретних прикладах.
3. Які типи даних можуть використовуватися як індекси масиву? Наведіть кілька прикладів.
4. Поясніть, чи можливий такий опис масиву:
a: array [10..5] of integer;
5. Яка конструкція використовується для введення масиву?
6. Є масива: array [0..10] of real. Скільки байтів він займає в пам'яті?



Приклад розв'язання практичного завдання

Приклад 18. Дано масив із N ($N \leq 50$) цілих чисел. Потрібно заповнити його випадковими числами в діапазоні від -10 до 10 . Вивести спочатку всі елементи, розташовані на парних позиціях, а потім — усі елементи на непарних позиціях. При цьому якщо перший елемент масиву більший за останній, то виведення слід здійснювати від початку до кінця масиву, у протилежному випадку — від кінця до початку.

Розв'язання. У ході розв'язування цього завдання необхідно звернути увагу на розмірність масиву. Завдання передбачає необхідність ввести число N (кількість елементів), при цьому його значення не може бути більшим за 50 . Отже, нас цілком улаштує статичний масив (пам'ять для якого виділяється на етапі компіляції). У процесі роботи програми, можливо, будуть задіяні не всі елементи масиву. Так буде, якщо користувач введе N , менше за 50 . Роботу зі статичним масивом демонструє така програма:

```

unit Unit1;
interface

uses
    Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms, Dialogs, StdCtrls;

```

```

type
    TForm1 = class(TForm)
    Edit1: TEdit;
    Label1: TLabel;
    Button1: TButton;
    Memo1: TMemo;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;
    a: array [1..50] of integer;
    i, N: integer;

implementation
{$R *.dfm}

procedure TForm1.FormCreate(Sender: TObject);
begin
    randomize;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
    N:=strtoint(edit1.Text); // введення числа елементів масиву
    for i:=1 to N do
        a[i]:=random(21)-10; // заповнення випадковими значеннями
    if a[1]>a[N] then // якщо перший елемент більший за останній
        begin // виведення від 1 до останнього
            for i:=1 to N do
                if i mod 2 = 0 then // перевірка на парність позиції
                    memo1.Lines.Add(inttostr(a[i]));
                for i:=1 to N do
                    if i mod 2 <> 0 then // перевірка на непарність позиції
                        memo1.Lines.Add(inttostr(a[i]));
            end
        end
    else // якщо перший елемент не більший за останній

```

```

begin                // виведення від останнього до 1
  for i:=N downto 1 do
    if i mod 2 = 0 then
      memo1.Lines.Add(inttostr(a[i]));
      for i:=N downto 1 do
        if i mod 2 <> 0 then
          memo1.Lines.Add(inttostr(a[i]));
        end
      end;
    end;
  end.

```



ПРАКТИЧНІ ЗАВДАННЯ

1. Організувати на формі введення масиву, що складається з 10 дійсних чисел. Організувати його виведення в порядку, оберненому порівняно зі введенням.
2. Організувати на формі введення масиву, що складається з 10 цілих чисел. В елементів масиву, значення яких перевищують 100, змінити знак на протилежний. Вивести парні елементи масиву.
3. Описано масив із N ($N \leq 50$) цілих чисел. Заповнити його випадковими числами в діапазоні від 10 до 16. Визначити, скільки в ньому елементів, які відрізняються від останнього елемента. Вивести знайдену кількість елементів.
4. Описано масив із N ($N \leq 40$) цілих чисел. Заповнити його випадковими числами в діапазоні від -10 до 10. Якщо друге число більше за передостаннє, то вивести на екран спочатку всі додатні елементи, у протилежному випадку — всі від'ємні.
5. Описано масив із N ($N \leq 40$) дійсних чисел. Заповнити його випадковими числами в діапазоні від -100 до 200. Вивести спочатку елементи від середини масиву до кінця, а потім — від середини до початку.

11.2

Обчислення підсумкових показників для числового масиву. Пошук даних у масиві

⇒ Обчислення підсумкових показників для числового масиву

Дуже велика частина завдань, що вимагають роботи з масивами чисел, передбачає обчислення показників: суми, добутку й середнього арифметичного елементів масиву.

Процес нагромадження суми елементів масиву є досить простим і фактично нічим не відрізняється від підсумовування значень певної

числової послідовності. Змінній, у якій зберігається значення суми, присвоюється значення, що дорівнює нулю, потім у циклі послідовно підсумовуються елементи масиву. Зазвичай у роботі з масивом використовується цикл FOR — це пов'язане з тим, що в такій структурі число ітерацій заздалегідь відоме. Тип змінної для зберігання суми має бути таким самим, як і тип елементів масиву, або поглинати його (у деяких випадках розмір може виявитися недостатнім). Наведемо фрагмент програми для пошуку суми елементів масиву:

```

-----
var
  a: array [1..10] of integer;
  i, S: integer;
begin
  ...
  S:=0;                      // початкове значення суми
  for i:=1 to 10 do
    S:=S+a[i];               // накопичування суми елементів
  ...
end;
-----

```

Пошук добутку елементів масиву здійснюється аналогічно. Відмінність полягає лише в задаванні початкового значення: якщо початкове значення суми дорівнює 0, то для добутку необхідно брати 1 (множення на одиницю не змінює значення). Наведемо фрагмент програми для пошуку добутку елементів масиву:

```

-----
var
  a: array [1..10] of integer;
  i, P: integer;
begin
  ...
  P:=1;                      // початкове значення добутку
  for i:=1 to 10 do
    P:=P*a[i];               // накопичування добутку елементів
  ...
end;
-----

```

Для пошуку середнього арифметичного значення всіх елементів масиву необхідно визначити суму, а потім поділити знайдене значення на кількість елементів масиву. Слід звернути увагу на те, що ділення треба виконувати за межами циклу, при цьому значення змінної має бути дійсного типу. Наведемо фрагмент програми для пошуку середнього арифметичного значення елементів масиву:

```

var
  a: array [1..10] of integer;
  i: integer;
  Sr: real;
begin
  ...
  Sr:=0; //спочатку Sr використовується як сума
  for i:=1 to 10 do
    Sr:=Sr+a[i];
  Sr:=Sr/10; //потім Sr стає середнім арифметичним
  ...
end;

```

У задачах, де використовуються масиви, часто постає необхідність визначити кількість елементів, що задовольняють певні критерії. Умови обмеження можуть бути найрізноманітнішими — наприклад, підрахунок кількості додатних, від’ємних, парних, непарних тощо елементів масиву. Наведемо фрагмент програми для знаходження кількості додатних елементів масиву:

```

var
  a: array [1..10] of integer;
  i, k: integer;
begin
  ...
  k:=0; // початкове значення кількості
  for i:=1 to 10 do
    if a[i]>0 then // умова вибору
      k:=k+1; // збільшити кількість на 1
  ...
end;

```

➔ Пошук даних у масиві

Пошук — одна з тих дій, які часто зустрічаються в програмуванні. Вхідними даними для алгоритму є послідовність елементів $a[1], a[2], a[3], \dots, a[N]$ (у нашому випадку це масив) і певний елемент x . Завдання полягає в тому, щоб одержати на виході номер елемента масиву, який дорівнює x , або з’ясувати, що такого елемента в послідовності немає.

Приклад 19. Навести фрагмент програми, що реалізовує послідовний пошук елемента в одновимірному масиві.



Розв'язання. За допомогою методу послідовного (лінійного) пошуку по черзі порівнюємо елементи масиву із заданим значенням. Виявивши збіг, повертаємо індекс знайденого елемента. Якщо після закінчення порівняння заданий елемент не знайдений, то зберігаємо інформацію про це (рис. 11.3).

Програма матиме такий вигляд:

```

unit Unit1;
interface
uses Windows, Messages, SysUtils, Variants,
Classes, Graphics, Controls, Forms, Dialogs,
StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label2: TLabel;
    Edit2: TEdit;
    Button2: TButton;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a: array[1..50] of integer;
  N, x, p: integer;
implementation
{$R *.dfm}
procedure TForm1.FormCreate(Sender: TObject);
begin
  randomize;
end;
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;

```

Рис. 11.3. Знаходження елемента масиву методом послідовного пошуку

```

begin
    N:=strtoint(Edit1.Text);    // задавання розміру масиву
    for i:=1 to N do
        a[i]:=random(10);
    end;
    procedure TForm1.Button2Click(Sender: TObject);
    var
        i:integer;
    begin
        x:=strtoint(Edit2.Text);    // введення елемента для пошуку
        p:=-1;                      // початкове значення позиції
        for i:=1 to N do           // у циклі перебираємо елементи
            if a[i]=x then         // якщо знайшли заданий елемент
                begin
                    p:=i;          // зберегти позицію
                    break;         // припинити пошук
                end;
        if p<>-1 then              // якщо був заданий елемент
            label3.Caption:='Позиція '+inttostr(p)
        else                       // якщо елемент відсутній
            label3.Caption:='Такого елемента немає'
        end;
    end.

```

Запропонований алгоритм має один суттєвий недолік: у процесі пошуку доводиться перевіряти в середньому половину масиву (у разі наявності шуканого елемента в масиві) або й весь масив (якщо шуканого елемента немає). У випадку великих розмірів це дуже неекономно.

Кількість операцій можна суттєво зменшити за допомогою бінарного (двійкового) пошуку. Це класичний алгоритм пошуку елемента у відсортованому масиві. Суть алгоритму полягає в зменшенні (у 2 рази) на кожному кроці області пошуку.

Розглянемо на прикладі послідовність дій для цього виду пошуку.

Приклад 20. Реалізувати бінарний спосіб пошуку елементів у масиві.

Розв'язання. Беремо середній елемент. Якщо він дорівнює шуканому, то пошук успішно завершений. Якщо середній елемент більший за шуканий, то шуканий елемент перебуває в першій половині масиву (якщо він є в масиві). Якщо ж середній елемент менший за шуканий, то шуканий елемент міститься в другій половині масиву. Таким чином, ми розбили масив на дві частини. Використовуємо той самий алгоритм в обраній частині масиву. Продовжуємо дії доти, доки не буде знайдено шуканий елемент або доведено, що він у масиві відсутній (рис. 11.4).

Програма матиме такий вигляд:



```

unit Unit1;
interface
uses Windows, Messages, SysUtils,
Variants, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls;
type
  TForm1 = class(TForm)
    Label1: TLabel;
    Edit1: TEdit;
    Button1: TButton;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a: array[1..20] of integer = (1,4,7,8,9,12,13,17,19,21,24,
32,36,44,45,54,55,63,66,70);
  x: integer;
  l, p, middle: integer;
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
begin
  x:=strtoint(Edit1.Text);
  l:=1; // початкове значення лівої позиції
  p:=20; // початкове значення правої позиції
  while l<=p do
  begin
    middle:=round((p+1)/2); // обчислення середини
    if x=a[middle] then
      // порівняння шуканого і середнього елементів
    begin
      label2.Caption:='Позиція '+inttostr(middle);
      exit;
    end;
  end;
end;

```

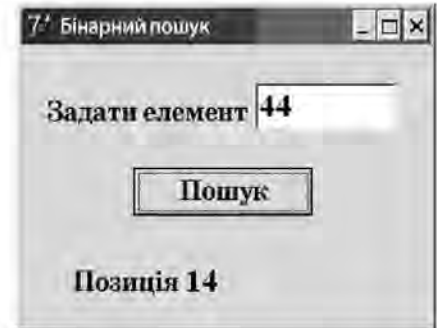


Рис. 11.4. Знаходження елемента масиву методом бінарного пошуку

```

    end
  else
    if x < a[middle] then // шуканий елемент у лівій частині
      p := middle - 1    // перенести праву межу
    else                 // шуканий елемент у правій частині
      l := middle + 1;    // перенести ліву межу
    end;
    label2.Caption := 'Такого елемента немає'
                      // якщо елемент відсутній

  end;
end.

```

У наведеній програмі слід звернути увагу на вміст масиву. Ми задали впорядкований у момент опису масив. Це пов'язане з тим, що способи сортування елементів масиву буде розглянуто в темі 11.5.

? ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Чим відрізняється послідовність дій у ході пошуку суми і добутку елементів масиву?
2. Дано фрагмент коду для обчислення суми елементів цілочислового масиву. Чи правильне значення суми буде одержано в результаті обчислень?

```

var
  a: array [1..10] of integer;
  i, S: integer;
begin
  S := a[1];
  for i := 2 to 10 do
    S := S + a[i];
  end;

```

3. Описано масив `a: array [1..500] of byte`; Якого типу має бути змінна, що зберігає значення суми?
4. Дано масив зі 100 цілих чисел. Скільки (у середньому) буде потрібно ітерацій для знайдення елемента за допомогою послідовного пошуку (за умови, що шуканий елемент наявний)?
5. Дано масив зі 100 цілих чисел. Скільки (у середньому) буде потрібно ітерацій для знайдення елемента за допомогою бінарного пошуку (за умови, що шуканий елемент наявний)?



ПРАКТИЧНІ ЗАВДАННЯ

1. Організувати на формі введення масиву, що складається з 10 цілих чисел. Підрахувати суму додатних чисел і суму (за модулем) від'ємних чисел. Організувати виведення спочатку елементів, сума яких більша, а потім елементів, сума яких менша.
2. Організувати на формі введення масиву, що складається з 12 цілих чисел. Підрахувати добуток парних чисел. Вивести числа масиву, які більші за знайдений добуток.
3. Організувати на формі введення масиву, що складається з 14 цілих чисел. Підрахувати середнє арифметичне чисел, що потрапили в діапазон від 10 до 100. Вивести числа масиву, менші за знайдене середнє арифметичне. Передбачити дії в ситуації, коли числа від 10 до 100 відсутні.
4. Описано масив із N ($N \leq 30$) дійсних чисел. Заповнити його випадковими числами в діапазоні від 10 до 100. Підрахувати середнє арифметичне чисел першої половини масиву й окремо — другої половини масиву. Вивести у зворотному порядку ту половину масиву, у якій середнє арифметичне менше.
5. Описано масив із N ($N \leq 50$) цілих чисел. Заповнити його випадковими числами в діапазоні від 100 до 1000. Ввести шукане число. За допомогою методу послідовного пошуку визначити позицію шуканого числа в масиві. (Попередньо масив має бути відсортований.) Для розв'язання задачі скористатися циклом WHILE.

Практична робота № 12
Обробка одновимірних масивів: введення
й виведення даних, пошук



Мета роботи: навчитися програмної обробки масивів.

Завдання. Організувати на формі введення рядка символів і визначити, з яких символів він складається; підрахувати кількість заданих символів.

Теоретичні відомості

Дані типу рядок (string) являють собою одновимірний масив, кожен елемент якого є символом (char). У Delphi існує кілька видів рядків, зокрема короткі (shortstring) і довгі (string).

Для роботи з рядком як із масивом необхідно знати кількість елементів (символів), із яких він складений. Це пов'язане з тим, що рядки є динамічним типом: їхня фактична довжина визначається в операції присвоювання. Кількість символів у рядку можна дізнатися за допомогою функції Length (рядок).

Природно, існує цілий арсенал функцій і процедур для роботи з рядковим масивом.

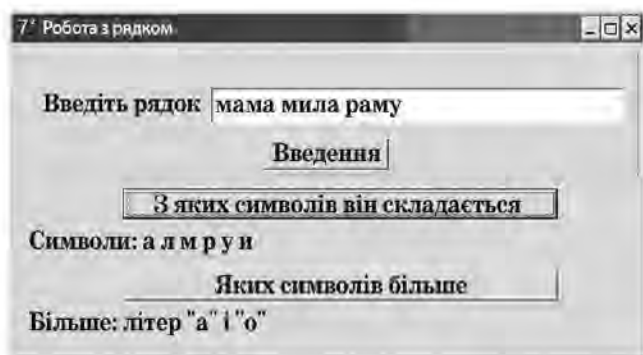
Хід роботи

Рис. 11.5. Зовнішній вигляд програми в роботі з рядковим масивом

1. Запустимо середовище Delphi. Створимо новий додаток. Для розв'язання задачі спроектуємо форму, показану на рис. 11.5.
2. В інтерфейсній частині проекту опишемо необхідні масиви:

var

```
Form1: TForm1;
s: string;
a: array [char] of integer;
```

implementation

3. Введемо програмний код в обробник події OnClick першої кнопки:
-

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i: integer;
  c: char;
begin
  s:=edit1.Text;
  panel1.Enabled:=false;
  for i:=1 to length(s) do
  begin
    c:=s[i];
    a[c]:=a[c]+1;
  end;
end;
```

4. Введемо програмний код в обробник події OnClick другої кнопки:
-

```
procedure TForm1.Button2Click(Sender: TObject);
var
  c: char;
begin
  for c:=#0 to #255 do
    if a[c]>0 then
      label2.Caption:=label2.Caption+c+' ';
  end;
```

5. Введемо програмний код в обробник події OnClick третьої кнопки:

```
-----
procedure TForm1.Button3Click(Sender: TObject);
var
    sum1,sum2:integer;
begin
    sum1:=a['a']+a['o'];
    sum2:=a['e']+a['и'];
    if sum1>sum2 then
        label3.Caption := label3.Caption+' літер «а» і «о»'
    else
        if sum2>sum1 then
            label3.Caption := label3.Caption+' літер «е» і «и»'
        else
            label3.Caption := label3.Caption+' літер однакова кількість'
    end;
```

6. Збережемо проект на диску.

У цьому завданні окремого пояснення потребує використання обробника події OnClick компонента Button1. Перше, на що слід звернути увагу,— це розташування поля введення Edit1 і кнопки Button1 на панелі Panel1, що дозволяє заборонити доступ (після закінчення введення) до всіх компонентів введення шляхом присвоювання panel1.Enabled:=false. Друге — заповнення масиву а. Цілочисловий масив, описаний у «верхньому» розділі **var**, є глобальним — отже, всі його елементи за замовчуванням дорівнюють нулю. Індексами масиву виступають символи (їх 256 значень, нумерованих від 0 до 255). Таким чином, у циклі **FOR** ми послідовно одержуємо доступ до всіх символів введеного рядка й залежно від того, який символ розглядається, збільшуємо відповідне значення масиву (лічильник) на одиницю.

Обробник події OnClick компонента Button2 передбачає перебір усіх можливих символів і виведення в label2 тих, які зустрілися принаймні один раз.

Обробник події OnClick компонента Button3 визначає дві суми — sum1 і sum2. Далі відбувається їх порівняння. Зверніть увагу на механізм порівняння двох змінних. Можливі три результати: перше значення більше, друге значення більше або ж вони однакові. Запропоноване розв'язання з використанням вкладених операторів **IF ... THEN ... ELSE** з погляду оптимізації коду є найкращим.

11.3

Обчислення підсумкових характеристик елементів, що задовольняють певні властивості

У людини завжди викликали інтерес числа, що мають якісь «незвичайні» властивості. До таких чисел можна віднести: прості, досконалі, дружні, числа ряду Фібоначчі, числа Армстронга та ін. До виникнення обчислювальної техніки визначення на належність чисел до цих категорій було вельми працемісткою справою. Це було пов'язане з великим обсягом обчислювальних операцій. Зараз для нас не становить особливої складності виконати кількост тисяч операцій для таких розрахунків.

Розглянемо, що собою являють перелічені вище числа.

1. *Просте число* — це натуральне число, яке має рівно два натуральні дільники (тільки 1 і самого себе). Усі інші числа (крім одиниці) називаються складеними. Послідовність простих чисел починається так: 2, 3, 5, 7, 11, 13, 17, 19, 23 ...

2. *Досконале число* — натуральне число, яке дорівнює сумі всіх власних дільників (тобто всіх додатних дільників, відмінних від цього числа). Перше досконале число — 6 ($1+2+3=6$), наступне — 28 ($1+2+4+7+14=28$). У міру того як натуральні числа зростають, досконалі числа зустрічаються все рідше.

3. *Числа ряду Фібоначчі* — елементи числової послідовності 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144 ... У цій послідовності кожне наступне число дорівнює сумі двох попередніх. Назву числа отримали за ім'ям середньовічного математика Леонардо Пізанського, відомого як Фібоначчі.

4. *Число Армстронга* — натуральне число, що дорівнює сумі своїх цифр, піднесених до степеня, який дорівнює кількості його цифр. Наприклад, десяткове число 153 — число Армстронга, оскільки $1^3 + 5^3 + 3^3 = 153$.

5. *Дружні числа* — два різних натуральних числа, для яких сума всіх власних дільників першого числа дорівнює другому числу, а сума всіх власних дільників другого числа дорівнює першому числу. Числа 220 і 284 визначив Піфагор (близько 500 р. до н. е.), числа 1184 і 1210 — Паганіні (1860 р.).

Розглянемо на прикладі роботу з числами.



Приклад 21. Організувати на формі введення цілих додатних чисел (їх кількість не може перевищувати 50). Вивести введені значення в компонент Мемо1. Є два набори перемикачів, що визначають, які числа перенести в компонент Мемо2 і які підсумкові показники необхідно підрахувати для перенесених значень.

Розв'язання. Для розв'язання цього завдання спроектуємо форму (вона показана на рис. 11.6).

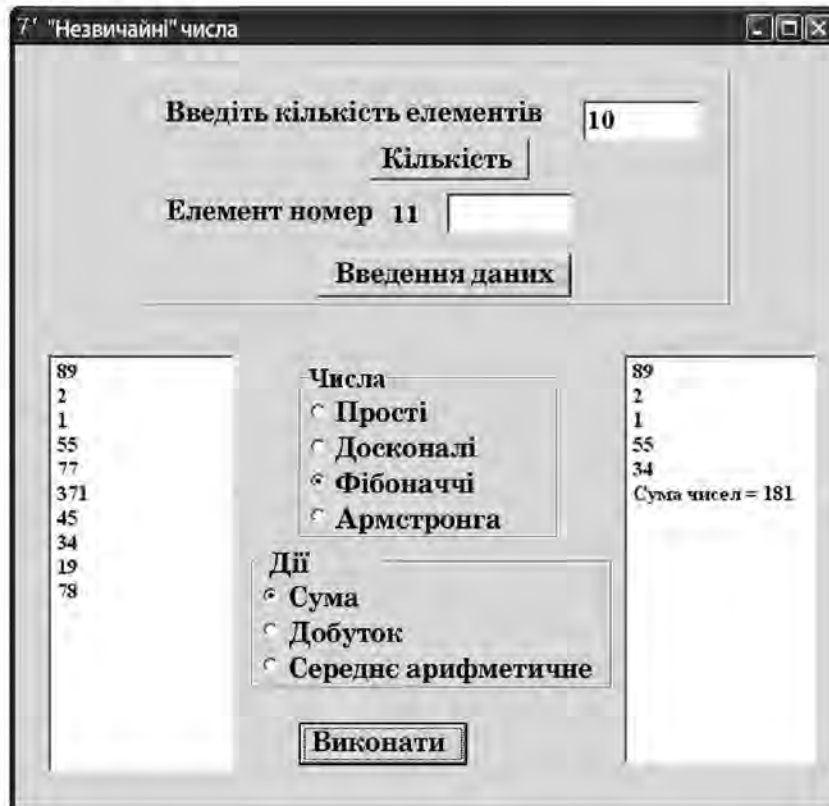


Рис. 11.6. Обчислення суми, добутку й середнього арифметичного значення для елементів, що задовольняють певні властивості

Програма матиме такий вигляд:

```
unit Unit1;
interface
uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    Edit1: TEdit;
    Label2: TLabel;
    Button1: TButton;
    Edit2: TEdit;
    Button2: TButton;
    Memo1: TMemo;
    Memo2: TMemo;
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
```

```

    Button3: TButton;
    Label3: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
private
    { Private declarations }
public
    { Public declarations }
end;

var
    Form1: TForm1;
    a: array [1..50] of integer;
    n: integer;
    i: integer=1;
implementation
{$R *.dfm}

function prost(x:integer):boolean; // функція перевірки на
                                   // належність до простих чисел
var
    i: integer;
begin
    Result:=true;                // вважаємо, що число просте
    for i:=2 to x-1 do           // перебираємо всі дільники (можна
                                   // до x div 2)
        if x mod i =0 then       // якщо ділиться, то число не просте
            Result:=false;
    if x=1 then                  // число один не просте
        Result:=false;
end;

function sov(x:integer):boolean; // функція перевірки на
                                   // належність до досконалих чисел
var
    i,sum: integer;
begin
    sum:=0;                      // початкове значення суми дільників
    for i:=1 to x-1 do           // перебираємо всі дільники
        if x mod i =0 then       // якщо i є дільником, то
            sum:=sum+i;          // додаємо його до суми
    if sum=x then
        Result:=true
    else
        Result:=false;
end;

```

```

function fib(x:integer):boolean; // функція перевірки
                                // на належність до чисел Фібоначчі
var
  a, b, c: integer;
begin
  a:=1;                        // перше число ряду
  b:=1;                        // друге число ряду
  while a < x do              // поки число, що перевіряється, більше за a
  begin
    c:=a+b;                    // обчислюємо третє число
    a:=b;                      // здійснюємо зсув по ряду
    b:=c;
  end;
  if a=x then
    Result:=true
  else
    Result:=false;
end;

```

```

function arm(x:integer):boolean; // функція перевірки
                                // на належність до чисел Армстронга
var
  st: string;
  i, j, sum, buf, y, p, k : integer;
begin
  st:=inttostr(x);              // рядкове подання числа
  k:=length(st);                // кількість цифр у числі
  buf:=x;                       // запам'ятовуємо число
  sum:=0;
  while x>0 do                 // поки ще є цифри
  begin
    y:=x mod 10;                // визначаємо останню цифру
    x:=x div 10;                // знищуємо її в числі
    p:=1;
    for j:=1 to k do           // підносимо цифру до степеня k
      p:=p*y;
    sum:=sum+p;                 // збільшуємо суму
  end;
  if sum=buf then              // якщо сума дорівнює числу
    Result:=true
  else
    Result:=false;
end;

```

```

procedure TForm1.Button1Click(Sender: TObject);
begin
    n:=strtoint(Edit1.Text);           // введення кількості елементів
end;

procedure TForm1.Button2Click(Sender: TObject);
var
    j: integer;
begin
    a[i]:=strtoint(Edit2.Text);         //запис значення з поля
                                         //введення

    label3.Caption:=inttostr(i+1);
    i:=i+1;                             // збільшення поточного індексу
    Edit2.Clear;                         // очистити поле введення
    Edit2.SetFocus;                      // перевести фокус
    if i=n+1 then                        // якщо індекс=n+1, то
                                         // закінчити введення

    begin
        Panel1.Enabled:=false;          // зробити недоступною панель
        for j:=1 to n do                // вивести в мемо1 введені числа
            memo1.Lines.Add(inttostr(a[j]));
    end;
end;

procedure TForm1.Button3Click(Sender: TObject);
var
    sum, pr, k, i: integer;
    sr: real;
begin
    memo2.Clear;
    sum:=0;                             // початкове значення суми
    pr:=1;                               // початкове значення
                                         // добутку
    k:=0;                                // початкове значення кількості
    case RadioGroup1.ItemIndex of
        0:                               // якщо обрано перший перемикач
            for i:=1 to n do            // перебираємо всі числа
                if prost(a[i]) then
                    begin                // якщо число просте
                        sum:=sum+a[i];    // збільшуємо значення суми
                        pr:=pr*a[i];      // збільшуємо значення
                                         // добутку
                        k:=k+1;            // збільшуємо кількість
                        memo2.Lines.Add(inttostr(a[i]))
                                         // виведення елемента масиву в мемо2
                    end;
            end;

```

```

1: for i:=1 to n do           // якщо обрано другий
                               // перемикач
    if sov(a[i]) then        // якщо число досконале
    begin
        sum:=sum+a[i];
        pr:=pr*a[i];
        k:=k+1;
        memo2.Lines.Add(inttostr(a[i]))
    end;
2: for i:=1 to n do           // якщо обрано третій
                               // перемикач
    if fib(a[i]) then        // якщо число Фібоначчі
    begin
        sum:=sum+a[i];
        pr:=pr*a[i];
        k:=k+1;
        memo2.Lines.Add(inttostr(a[i]))
    end;
3: for i:=1 to n do           // якщо обрано четвертий
                               // перемикач
    if arm(a[i]) then        // якщо число Армстронга
    begin
        sum:=sum+a[i];
        pr:=pr*a[i];
        k:=k+1;
        memo2.Lines.Add(inttostr(a[i]))
    end;
end;
if RadioGroup2.ItemIndex=0 then // виводимо суму
    memo2.Lines.Add('Сума чисел = '+inttostr(sum))
else
    if RadioGroup2.ItemIndex=1 then // виводимо добуток
        memo2.Lines.Add('Добуток чисел = '
+inttostr(pr))
    else
    begin
        if k>0 then           // виводимо середнє арифметичне
            memo2.Lines.Add('Середнє арифметичне чисел = '
+ floattostr(sum/k))
        end;
    end;
end;
end.

```

**ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ**

1. За допомогою якої базової конструкції можна перебрати всі елементи масиву?
2. Поясніть, як можна переписати функцію визначення простого числа без використання булевих змінних.
3. Чому у визначенні простого числа цикл FOR можна виконувати не до $x-1$, а до $x \div 2$? Доведіть це на прикладах.
4. До якого дільника слід перевіряти числа на належність до простих чисел у діапазоні від 2 до 100? від 2 до 10 000?

**ПРАКТИЧНІ ЗАВДАННЯ**

1. Організувати на формі введення масиву, що складається з N ($N \leq 20$) цілих чисел. Визначити кількість чисел у масиві, запис яких закінчується двома нулями (наприклад, 20 100, 400). Вивести на форму одержане значення.
2. Організувати на формі введення масиву, що складається з N ($N \leq 20$) цілих чисел. Визначити кількість чисел у масиві, у записі яких є два нулі (наприклад, 2010, 400). Вивести на форму одержане значення.
3. Організувати на формі введення масиву, що складається з N ($N \leq 10$) цілих чисел. Визначити суму чисел і кількість чисел у масиві, запис яких закінчується двома нулями (наприклад, 20 100, 400). Вивести на форму одержане значення.
4. Організувати на формі введення масиву, що складається з N ($N \leq 20$) цілих чисел. Визначити добуток чисел, запис яких починається й закінчується тією самою цифрою (наприклад, 4564, 5125). Вивести на форму одержане значення.
5. Організувати на формі введення масиву, що складається з N ($N \leq 15$) цілих чисел. Визначити кількість чисел, значення яких перевищує середнє арифметичне всіх чисел масиву. Вивести на форму знайдене значення й сам масив.

11.4**Вибірання елементів із масиву за певним критерієм****Знаходження мінімального і максимального елементів масиву**

Завдання пошуку мінімального і максимального елементів у не-впорядкованому масиві є одним із найпоширеніших у роботі з масивами. Розглянемо спочатку всі питання, пов'язані з пошуком мінімального елемента, а потім зупинимося на відмінностях пошуку максимального.

Алгоритм пошуку мінімального елемента масиву є доволі простим. Однак у процесі написання програмного коду можливі труднощі з вибором початкового значення мінімуму. Зустрічаються реалізації, де як початковий мінімум береться максимально можливе для даного типу значення. Такий підхід неприйнятний за умови змінювання типів даних, коли, наприклад, від цілочислового типу ми переходимо до дійсного.

Краще взяти за вихідне число перший елемент масиву, з яким будуть послідовно порівнюватись інші елементи. Якщо під час чергової перевірки виявляється, що перевірюваний елемент менший, ніж узятий за мінімальний, то новий елемент стає мінімальним, після чого триває перевірка решти елементів.

Отже, напишемо алгоритм:

```
-----
var
  a: array [1..10] of real;
  i: integer;
  min: real;
begin
  min:=a[1];           //задаємо початкове значення мінімуму
  for i:=2 to 10 do    //цикл для решти елементів
    if a[i]<min then    //якщо поточний елемент менший
                        //за мінімальний
      min:=a[i];       //мінімуму присвоїти нове значення
  label1.Caption := floattostr(min);
                        //виведення мінімального значення min
end;
-----
```

Запропонований алгоритм коректний, проте він має суттєвий недолік, пов'язаний із тим, що в деяких задачах не досить лише визначити мінімальний елемент. Часто виникає необхідність далі обробити (переставити, видалити, вставити та ін.) мінімальний елемент.

Для розв'язання таких задач необхідно знати не тільки значення мінімального елемента, але і його індекс. Звичайно, після наведеного фрагмента коду можна написати й цикл, у якому буде порівнюватись уже знайдений мінімум із кожним з елементів масиву, а також запам'ятовуватись їхній індекс (у разі рівності значень). Однак такий спосіб є неефективним. Замість цього переписемо алгоритм пошуків мінімального елемента, у якому будемо шукати не значення мінімального елемента, а його індекс:

```
-----
var
  a: array [1..10] of real;
  i, pmin: integer;
```

```

begin
    pmin:=1;                      //задаємо початкове значення
                                //індексу мінімуму
    for i:=2 to 10 do             //цикл для решти елементів
        if a[i]<a[pmin] then       //якщо поточний елемент менший
                                //за мінімальний
            pmin:=i;              //індексу мінімуму присвоїти
                                //нове значення
    label1.Caption := floattostr(a[pmin]);
                                //виведення мінімального значення a[pmin]
end;
-----

```

У пошуку мінімуму актуальним стає питання: а який із мінімальних елементів буде знайдено у випадку, якщо їх кілька? Мається на увазі, що елементів, які дорівнюють мінімальному, у масиві може бути декілька. Відповідь криється в операції порівняння. У випадку використання конструкції **IF a[i]<a[pmin] THEN ...** ми знайдемо індекс першого з елементів, які дорівнюють мінімальному. Якщо ж використати **IF a[i]<=a[pmin] THEN ...**, то як відшукуване значення буде взято індекс останнього елемента. Для ряду задач, у яких передбачена наявність у вихідному наборі даних кількох рівних мінімальних значень, вибір знака порівняння має суттєве значення. Якщо ж в умові задачі не уточнюється, порядковий номер якого саме мінімуму необхідно відшукати, то використовується порівняння за допомогою знака «строого менше».

У пошуку максимального елемента масиву відмінність програмного коду полягає тільки в операції порівняння:

```

var
    a: array [1..10] of real;
    i, pmax: integer;
begin
    pmax:=1;                      //задаємо початкове значення
                                //індексу максимуму
    for i:=2 to 10 do             //цикл для решти елементів
        if a[i]>a[pmax] then       //якщо поточний елемент більший
                                //за максимальний
            pmax:=i;              //індексу максимуму присвоїти
                                //нове значення
    label1.Caption := floattostr(a[pmax]);
                                //виведення максимального значення a[pmax]
end;
-----

```

У випадку роботи відразу з двома (а іноді й більше) масивами програма ускладнюється: тепер необхідно використовувати два поточні індекси — для першого і другого масивів. Розглянемо це на прикладі.

Приклад 22. В одновимірному масиві, що складається з 15 дійсних чисел (числа задані випадковим чином у діапазоні від -100 до 100), визначити позиції максимального і мінімального елементів, переписати всі елементи масиву, які перебувають між двома знайденими позиціями, в інший масив. Вивести на форму одержаний масив.

Розв'язання. Для розв'язання цієї задачі розробимо форму, подану на рис. 11.7. Текст програми має такий вигляд:

```

-----
unit Unit1;
interface
uses Windows, Messages,
SysUtils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls, Buttons;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Memo1: TMemo;
    Memo2: TMemo;
    SpeedButton1: TSpeedButton;
    Label1: TLabel;
    Label2: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure SpeedButton1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
  a,b :array [1..15] of real;

implementation
{$R *.dfm}

```

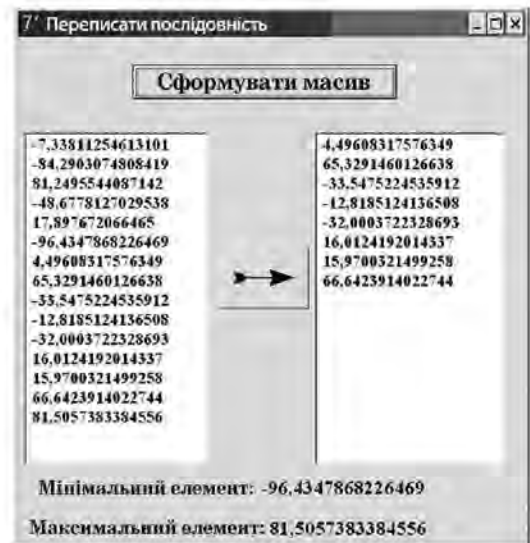


Рис. 11.7. Перенесення з одного масиву в інший послідовності чисел, що перебуває між максимальним і мінімальним елементами



```

procedure TForm1.Button1Click(Sender: TObject);
var
    i: integer;
begin
    Memo1.Clear; // очищення Memo1
    for i:=1 to 15 do
        a[i]:=random*200-100;
    for i:=1 to 15 do // виведення масиву a
        Memo1.Lines.Add(floattostr(a[i]));
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
    randomize;
end;

procedure TForm1.SpeedButton1Click(Sender: TObject);
var
    i, pmin, pmax, p: integer;
begin
    Memo2.Clear; // очищення Memo1
    pmin:=1; // задавання початкових значень
    pmax:=1; // для індексів
    for i:=2 to 15 do // перебираємо всі елементи масиву
    begin
        if a[i]<a[pmin] then // якщо поточний елемент менший
            // за мінімальний
            pmin:=i; // запам'ятати його позицію
        if a[i]>a[pmax] then // якщо поточний елемент більший
            // за максимальний
            pmax:=i; // запам'ятати його позицію
    end;

    // виведення мінімального і максимального елементів
    Label1.Caption:='Мінімальний елемент: ' + floattostr(a[pmin]);
    Label2.Caption:='Максимальний елемент: ' + floattostr(a[pmax]);
    p:=0; // початкова позиція в масиві b
    if pmin<pmax then // якщо мінімум зустрівся раніше
        for i:=pmin+1 to pmax-1 do
        begin
            inc(p); // зсунути позицію p
            b[p]:=a[i]; // писати в позицію p
        end
    else

```

```

for i:=pmax+1 to pmin-1 do
begin
  inc(p);
  b[p]:=a[i];
end;
for i:=1 to p do // вивести масив b
  Memo2.Lines.Add(floattostr(b[i]));
end;

end.

```



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Поясніть, які типи даних використовуються для індексів мінімального (максимального) елемента масиву й самого значення мінімуму (максимуму).
2. Які переваги дає пошук позиції максимального (мінімального) елемента порівняно з пошуком значення максимального (мінімального) елемента?
3. У чому відмінність порівнянь «менше» і «менше або дорівнює», що використовуються в пошуку значення мінімального елемента?
4. Чи будуть відрізнятися результати у випадках використання в програмі порівнянь «менше» і «менше або дорівнює», якщо відомо, що в масиві немає повторюваних значень?
5. Чи працюватимуть запропоновані в розгляді цієї теми приклади у випадку, якщо всі елементи масиву рівні між собою?



ПРАКТИЧНІ ЗАВДАННЯ

1. Організувати на формі введення масиву, що складається з N ($N \leq 20$) цілих чисел. Знайти мінімальне значення в першій половині масиву й максимальне значення в другій половині масиву. Поміняти місцями знайдені елементи в масиві. Вивести одержаний масив у компонент Мемо.
2. Організувати на формі введення масиву, що складається з N ($N \leq 15$) цілих чисел. Знайти мінімальне значення серед парних елементів масиву. Вивести на формі знайдене значення і сам масив.
3. Організувати на формі введення масиву, що складається з N ($N \leq 25$) дійсних чисел. Визначити індекс елемента масиву, який має мінімальне значення. Якщо мінімальний елемент перебуває в першій половині масиву, то вивести масив у порядку його введення. Якщо мінімальний елемент перебуває в другій половині масиву, то вивести масив в оберненому порядку порівняно з введенням.

4. Організувати на формі введення масиву, який складається з N ($N \leq 20$) цілих чисел. Якщо в масиві міститься більш ніж один елемент, що дорівнює максимальному елементу, то змінити знак у всіх елементів, що дорівнюють максимуму, і вивести на формі масив.
5. Організувати на формі введення масиву, що складається з N ($N \leq 15$) цілих чисел. На місце елемента, який іде за кожним мінімальним елементом, поставити нуль. Вивести одержаний масив.

11.5

Стандартні функції сортування масиву.

Злиття масивів

Необхідність відсортувати якісь величини виникає в програмуванні дуже часто. Наприклад, вхідні дані подаються «впереміш», а вашій програмі зручніше обробляти впорядковану послідовність. Існують ситуації, коли попереднє сортування даних дозволяє скоротити змістову частину алгоритму в рази, а час його роботи — в десятки разів.

Сортуванням, або впорядкуванням, масиву називається розташування його елементів у порядку зростання (або спадання). Якщо не всі елементи різні, то треба говорити про невисхідний (або неспадний) порядок.

⇒ Сортування методом бульбашки

Перший алгоритм, із яким мають справу всі програмісти, вивчаючи сортування, — це так зване бульбашкове сортування (*bubble sort*). Щоб продемонструвати цей принцип сортування, скористаємося масивом, який складається з 10 елементів. Послідовність дій можна описати так. Порівнюємо дев'ятий і десятий елементи. Якщо десятий елемент більший за дев'ятий, міняємо їх місцями. Тепер переходимо до восьмого і дев'ятого елементів. Якщо дев'ятий елемент більший за восьмий, міняємо їх місцями. Те саме зробимо для пар (7, 8), (6, 7) і т. д., поки не дійдемо до першого й другого елементів.

Після першого проходу по всьому масиву найбільший елемент опиниться на першій позиції: у кожному порівнянні максимальний елемент виявляється більшим за попередній і міняється з ним місцями — ніби «спливає» вгору. Повернемося знову до дев'ятого й десятого елементів. Виконаємо описаний вище процес, але цього разу зупинимося на другому й третьому елементах, тобто продовжимо процес сортування, зменшуючи з кожним новим циклом кількість елементів, що переглядаються, і роблячи так доти, доки весь масив не буде відсортований.

Наведемо програмний код, що реалізовує сортування методом бульбашки.

Приклад 23.

```

var
  a: array [1..10] of integer=(4,18,5,8,90,41,1,12,29,59);
  i,j,n,buf: integer;
begin
  n:=10;
  for i:=1 to n-1 do
    for j:=n downto i+1 do
      if a[j]>a[j-1] then
        begin
          buf:=a[j];
          a[j]:=a[j-1];
          a[j-1]:=buf;
        end;
      end;
    end;
  end.

```

Слід зупинитися на частині алгоритму, у якій здійснюється обмін значень між двома змінними. Припустимо, ми маємо дві змінні: X і Y . Необхідно зробити обмін між ними, тобто значення, що зберігається в змінній X , записати в змінну Y , а значення змінної Y записати в змінну X .

Наведемо наочний приклад, у якому задіяні не змінні, а склянка з апельсиновим соком і чашка з кавою. У результаті переливань необхідно зробити так, щоб у склянці була кава, а в чашці — сік. Для виконання таких дій необхідна ще одна посудина (нехай це буде інша склянка). Послідовність дій показано на рис. 11.8. Перший крок — переливаємо в допоміжну склянку сік. Потім у склянку, що звільнилася, переливаємо каву. І на останньому, третьому, кроці наливаємо в чашку з допоміжної склянки сік. Міркуючи аналогічно, можна здійснити обмін між двома змінними X і Y .

Розглянемо весь процес переставлень методом бульбашки на прикладі цілочислового масиву:



Рис. 11.8. Наочний приклад, що демонструє послідовність дій під час обміну значень між двома змінними

4	18	5	8	90	41	1	12	29	59
---	----	---	---	----	----	---	----	----	----

Результати переставлень після кожного i -го кроку подано в табл. 11.1.

Таблиця 11.1

Послідовність переставлень методом бульбашки

Крок 1	90	4	18	5	8	59	41	1	12	29
Крок 2	90	59	4	18	5	8	41	29	1	12
Крок 3	90	59	41	4	18	5	8	29	12	1
Крок 4	90	59	41	29	4	18	5	8	12	1
Крок 5	90	59	41	29	18	4	12	5	8	1
Крок 6	90	59	41	29	18	12	4	8	5	1
Крок 7	90	59	41	29	18	12	8	4	5	1
Крок 8	90	59	41	29	18	12	8	5	4	1
Крок 9	90	59	41	29	18	12	8	5	4	1

Як крок розглядається одна ітерація зовнішнього циклу, тобто розташування одного елемента масиву на своїй позиції.

Основна проблема бульбашкового сортування полягає в тому, що переставляються тільки сусідні елементи. Якщо елемент із найбільшим значенням опиняється аж у кінці списку, він буде мінятися місцями із сусідніми елементами доти, доки не досягне першої позиції.

➡ Сортування методом простого вибирання

Наступним за складністю алгоритмом є сортування методом простого вибирання (*straight selection sorting*). Ідея цього методу полягає в поданні вихідного масиву (завдовжки n) у вигляді двох частин: «підсумку» і «залишку». «Підсумок» — це вже відсортована частина масиву. Вона впорядкована й розташовується в елементах із меншими індексами (на початку масиву). Ділянка масиву, що називається залишком, розташована впритул до «підсумку» і містить вихідні числа невідсортованої частини вихідного масиву.

Припустимо, що перший елемент «залишку» є j -м елементом масиву, і складемо алгоритм сортування вибиранням. Він міститиме такі кроки:

Крок 1. Закладено $j = 0$, тобто вважається, що підсумкова ділянка порожня.

Крок 2. У залишку масиву шукається мінімальний елемент, який міняється місцем із першим елементом залишку (j -м елементом масиву). Після цього значення j збільшується на одиницю, цим розширюючи підсумкову ділянку масиву (відсортовану частину вихідного масиву).

Крок 3. Якщо $j < n - 1$, то повторюється крок 2. У протилежному випадку закінчується алгоритм, оскільки підсумок стає рівним усьому масиву.

Наведемо програмний код, який реалізовує сортування методом простого вибирання.

Приклад 24.

```

var
  a: array [1..10] of integer=(4,18,5,8,90,41,1,12,29,59);
  i, j, n, buf, IndexOfMin: integer;
begin
  n:=10;
  for i:=1 to n-1 do
  begin
    IndexOfMin:=i;
    for j:=i+1 to n do
      if a[j]<a[IndexOfMin] then
        IndexOfMin:=j;
    if IndexOfMin<>i then
      begin
        buf:=a[i];
        a[i]:=a[IndexOfMin];
        a[IndexOfMin]:=buf;
      end;
    end;
  end;
end.

```

Розглянемо весь процес переставлень методом простого вибирання на прикладі того самого цілочислового масиву:

4	18	5	8	90	41	1	12	29	59
---	----	---	---	----	----	---	----	----	----

Результати переставлень після кожного i -го кроку показано в табл. 11.2.

Таблиця 11.2

Послідовність переставлень методом простого вибирання

Крок 1	1	18	5	8	90	41	4	12	29	59
Крок 2	1	4	5	8	90	41	18	12	29	59
Крок 3	1	4	5	8	90	41	18	12	29	59
Крок 4	1	4	5	8	90	41	18	12	29	59
Крок 5	1	4	5	8	12	41	18	90	29	59
Крок 6	1	4	5	8	12	18	41	90	29	59
Крок 7	1	4	5	8	12	18	29	90	41	59
Крок 8	1	4	5	8	12	18	29	41	90	59
Крок 9	1	4	5	8	12	18	29	41	59	90

⇒ Сортування уставлянням

Найпростіший спосіб сортування — це впорядкування даних у міру їх надходження. У цьому випадку, коли вводиться кожне нове значення, елементи, введені раніше, уже утворюють відсортовану послідовність. У літературі такий спосіб сортування зустрічається під назвою «сортування уставлянням» або «сортування методом вставок» (*insertion sort*).

Алгоритм сортування уставлянням складається з таких кроків:

Крок 1. Перший елемент записати «без роздумів».

Поки не закінчиться послідовність даних для введення, для кожного нового її елемента виконувати такі дії:

Крок 2. Почавши з кінця вже наявної впорядкованої послідовності, всі її елементи, більші за новий елемент, зсунути на один крок назад.

Крок 3. Записати новий елемент на звільнене місце.

При цьому, зрозуміло, можна прочитати всі елементи одночасно, записати їх у масив, а потім розглядати один за одним, розставляючи в потрібному порядку, ніби повторюючи введення масиву. На принцип переставлення елементів і результат роботи алгоритму це не вплине.

Наведемо програмний код, що реалізовує цей спосіб сортування.

Приклад 25.



```

var
  a: array [1..10] of integer=(4, 18, 5, 8, 90, 41, 1, 12, 29, 59);
  i,j,n,x: integer;
begin
  n:=10;
  for i:= 2 to n do
    if a[i-1]>a[i] then
      begin
        x:=a[i];
        j:= i-1;
        while (j>0)and(a[j]>x) do
          begin
            a[j+1]:= a[j];
            j:= j-1;
          end;
        a[j+1]:= x;
      end;
  end.

```

Результати переставлень після кожного i -го кроку подано в табл. 11.3.

Наведена реалізація сортування уставлянням має одну особливість: значення поточного елемента зберігається в локальній змінній, а потім у процесі пошуку потрібного місця для його вставлення (внутрішній цикл) ми переміщуємо кожен елемент, значення якого більше за поточне, на одну позицію праворуч, цим переміщуючи вліво «дірку» в списку. Врешті ми знаходимо потрібне місце для розміщення збереженого значення.

Таблиця 11.3

Послідовність переставлень методом уставлянь

Крок 1	4	18	5	8	90	41	1	12	29	59
Крок 2	4	5	18	8	90	41	1	12	29	59
Крок 3	4	5	8	18	90	41	1	12	29	59
Крок 4	4	5	8	18	90	41	1	12	29	59
Крок 5	4	5	8	18	41	90	1	14	29	59
Крок 6	1	4	5	8	18	41	90	12	29	59
Крок 7	1	4	5	8	12	18	41	90	29	59
Крок 8	1	4	5	8	12	18	29	41	90	59
Крок 9	1	4	5	6	12	18	29	41	59	90



ТЕОРЕТИЧНІ ПИТАННЯ ДЛЯ САМОКОНТРОЛЮ

1. Якими є завдання сортування масиву?
2. У чому полягають особливості сортування методом бульбашки? Чому так називається цей метод?
3. Поясніть, як здійснюється обмін значень між двома змінними.
4. У чому полягають особливості використання сортування методом простого вибирання?
5. Якими є особливості використання сортування уставлянням?

Практична робота № 13

Обробка одновимірних масивів: обчислення підсумкових показників, кількості елементів, використання стандартної функції сортування



Мета роботи: навчитися писати програми для обробки одновимірних масивів.

Завдання. У масиві констант записано 10 прізвищ учнів. У ще одному масиві з 10 цілих чисел кожному учневі виставлено оцінку з предмета. Визначити й вивести: середній бал за даною таблицею оцінок, максимальний бал, мінімальний бал, прізвища учнів із максимальним балом, прізвища учнів із мінімальним балом. Вивести кількість учнів, що встигають на 10 балів і вище, кількість учнів, які мають 3 бали й нижче. Вивести в Методі список учнів за абеткою та їхні оцінки.

Хід роботи

1. Запустимо середовище Delphi та створимо новий додаток.

2. Для розв'язання цього завдання розробимо форму, показану на рис. 11.9. На форму винесено компоненти `OptionGroup`, `Button` і `Мемо`.

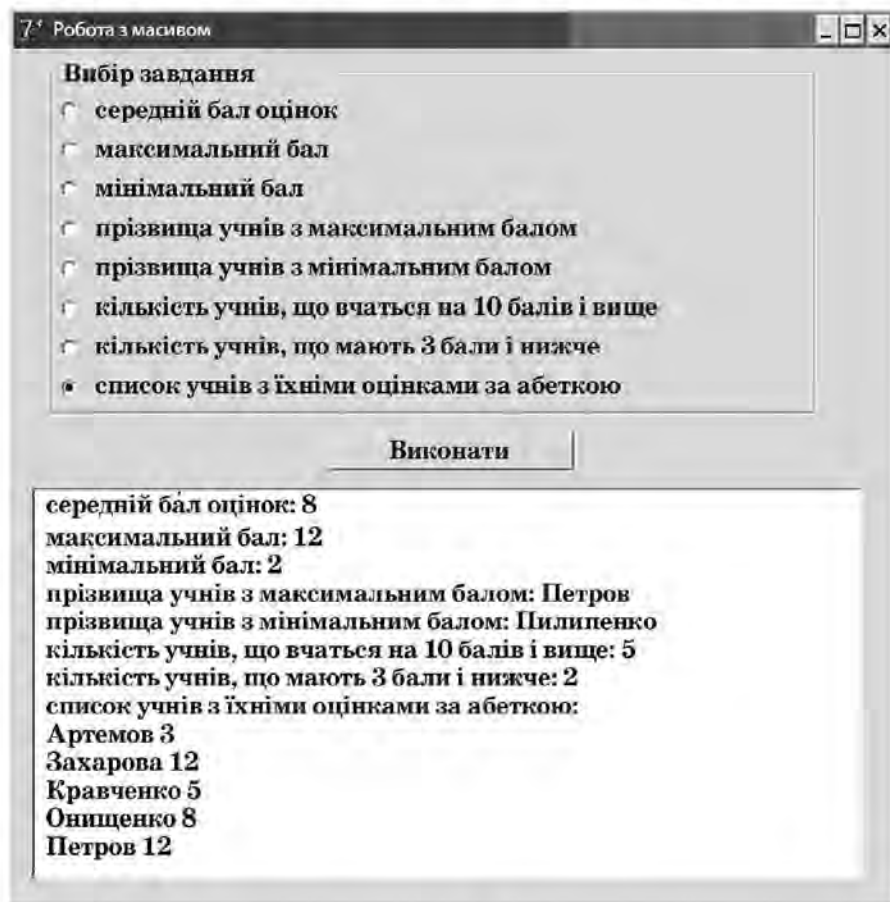


Рис. 11.9. Приклад завдання на обробку одновимірних масивів

3. В інтерфейсній частині програми опишемо два масиви констант:

```
var
  Form1: TForm1;
  fam: array [1..10] of string=('Кравченко', 'Петров',
    'Середа', 'Іванов', 'Онищенко', 'Шевченко', 'Петруніна',
    'Захарова', 'Пилипенко', 'Артемов');
  oz: array [1..10] of byte=(5, 12, 11, 7, 8, 10, 10, 12, 2, 3);
```

4. В обробник події `OnClick` запишемо такий програмний код:

```
implementation
{$R *.dfm}
procedure TForm1.Button1Click(Sender: TObject);
var
  i, j: integer;
```

```

sr: real;
k, pmax, pmin, min, max, n, buf1: integer;
buf2: string;
begin
  k:=RadioGroup1.ItemIndex;
                                     // номер обраного перемикача
  case k of
    0:begin
      sr:=0;                          // підрахунок середнього арифметичного бала
      for i:=1 to 10 do
        sr:=sr+oz[i];
      sr:=sr/10;
      memo1.Lines.Add(RadioGroup1.Items[k]+' : '+floattostr(sr));
    end;
    1:begin
      max:=oz[1];                     // пошук максимальної оцінки
      for i:=2 to 10 do
        if oz[i]>max then
          max:=oz[i];
      memo1.Lines.Add(RadioGroup1.Items[k]+' : '+inttostr(max));
    end;
    2:begin
      min:=oz[1];                     // пошук мінімальної оцінки
      for i:=2 to 10 do
        if oz[i]<min then
          min:=oz[i];
      memo1.Lines.Add(RadioGroup1.Items[k]+' : '+inttostr(min));
    end;
    3:begin
      pmax:=1;                        // прізвища з максимальним балом
      for i:=2 to 10 do
        if oz[i]>oz[pmax] then
          pmax:=i;
      for i:=1 to 10 do
        if oz[i]=oz[pmax] then
          memo1.Lines.Add(RadioGroup1.Items[k]+' : '+fam[i]);
    end;
    4:begin
      pmin:=1;                        // прізвища з мінімальним балом
      for i:=2 to 10 do
        if oz[i]<oz[pmin] then
          pmin:=i;
      for i:=1 to 10 do
        if oz[i]=oz[pmin] then
          memo1.Lines.Add(RadioGroup1.Items[k]+' : '+fam[i]);
    end;
  end;

```

```

5:begin
  n:=0; // кількість оцінок від 10 балів і вище
  for i:=1 to 10 do
    if oz[i]>=10 then
      n:=n+1;
  memo1.Lines.Add(RadioGroup1.Items[k]+' : '+inttostr(n));
end;
6:begin
  n:=0; // кількість оцінок від 1 до 3 балів
  for i:=1 to 10 do
    if oz[i]<=3 then
      n:=n+1;
  memo1.Lines.Add(RadioGroup1.Items[k]+' : '+inttostr(n));
end;
7:begin
  for i:=1 to 9 do // упорядкувати прізвища
                  // за абеткою
    for j:=10 downto i+1 do
      if fam[j]<fam[j-1] then
        begin
          buf1:=oz[j];
          oz[j]:=oz[j-1];
          oz[j-1]:=buf1;
          buf2:=fam[j];
          fam[j]:=fam[j-1];
          fam[j-1]:=buf2;
        end;
    memo1.Lines.Add(RadioGroup1.Items[k]+' : ');
    for i:=1 to 10 do
      memo1.Lines.Add(fam[i]+' '+inttostr(oz[i]));
    end;
  end;
end;
end;
end.

```

5. Збережемо проект на диску у своїй папці й запусимо програму.

СПИСОК ЛІТЕРАТУРИ

1. *Інформатика*. Програми для профільного навчання та допрофільної підготовки.— К.: Вид. група BHV, 2009.— 400 с.
2. *Архангельский А. Я.* Программирование в Delphi. Учебник по классическим версиям Delphi.— М.: ООО «Бином-Пресс», 2006.— 1152 с.
3. *Ахо Альфред В., Хопкрофт Джон, Ульман Джеффри Д.* Структуры данных и алгоритмы: Пер. с англ.— М.: Вильямс, 2001.
4. *Бакнелл Джулиан М.* Фундаментальные алгоритмы и структуры данных в Delphi: Пер. с англ. / Джулиан М. Бакнелл.— СПб.: ООО «ДиаСофт ЮП», 2003.— 560 с.
5. *Вирт Н.* Алгоритмы и структуры данных: Пер. с англ.— 2-е изд., испр.— СПб.: Невский диалект, 2001.
6. *Глинський Я. М., Анохін В. Є., Ряжська В. А.* Паскаль. Turbo Pascal і Delphi.— 2-е доп. вид.— Львів: «Деол», 2001.— 144 с.
7. *Глинський Я. М.* Інформатика. 10–11 класи: Навч. посібник: У 2-х ч. Ч. 1. Алгоритмізація і програмування.— 8-е вид.— Львів: СПД Глинський, 2008.— 256 с.
8. *Дарахвелидзе П. Г., Марков Е. П.* Программирование в Delphi 7.— СПб.: БХВ-Петербург, 2003.
9. *Завадський І. О., Заболотний Р. І.* Основи візуального програмування.— К.: Вид. група BHV, 2009.— 272 с.
10. *Руденко В. Д., Макаrchук О. М., Патланжоглу М. О.* Базовий курс інформатики: У 2-х кн. Кн. 2. Інформаційні технології.— К.: Вид. група BHV, 2006.— 368 с.
11. *Фаронов В. В.* Delphi. Программирование на языке высокого уровня.— СПб.: Питер, 2009.— 640 с.

ЗМІСТ

Вступ	3
Розділ 1. Основні поняття програмування	
Тема 1.1	Історія мов програмування. Поняття програми, мови програмування, компілятора та інтерпретатора. Сучасні мови програмування5
Тема 1.2	Алгоритм, основні властивості алгоритмів.....11
Розділ 2. Створення найпростішого проекту	
Тема 2.1	Знайомство з візуальним середовищем програмування. Елементи вікна середовища програмування. Програмна розробка й файли, які входять до її складу. Створення найпростішого проекту, його компіляція, збереження, виконання16
Тема 2.2	Додавання кількох рядків коду до обробника події, їх аналіз. Поняття форми, елемента керування, події, обробника події. Редагування коду обробника події.....23 <i>Практична робота № 1</i>29
Розділ 3. Алгоритми та їх програмна реалізація	
Тема 3.1	Способи опису алгоритмів. Складання й запис алгоритмів. Базові алгоритмічні конструкції32
Тема 3.2	Структура й складові елементи програм, записаних об'єктозорієнтованою мовою програмування35 <i>Практична робота № 2</i>37
Розділ 4. Форми та елементи керування	
Тема 4.1	Основні компоненти Windows-програми. Розроблення й застосування форм. Елементи керування та їхні атрибути ...39
Тема 4.2	Використання вікон48 <i>Практична робота № 3</i>53
Розділ 5. Атрибути, змінні, присвоювання і стандартні методи	
Тема 5.1	Поняття змінної і константи. Поняття ідентифікатора. Установлення атрибутів форм і елементів керування в програмі58
Тема 5.2	Типи даних. Оператор присвоювання63 <i>Практична робота № 4</i>66
Розділ 6. Налаштування програм	
Тема 6.1	Використання налагоджувача програм у візуальному середовищі програмування. Покрокове виконання програм, перегляд значень змінних під час виконання програми. Різновиди помилок, методи їх пошуку та виправлення68 <i>Практична робота № 5</i>72

Розділ 7. Операції

Тема 7.1	Поняття операції та виразу. Пріоритет операцій. Арифметичні операції. Операції над рядками77 <i>Практична робота № 6</i>81
----------	--

Розділ 8. Умовні оператори

Тема 8.1	Поняття про булеву логіку. Логічні (булеві) операції та операції відношення (порівняння)84
Тема 8.2	Формування умов88
Тема 8.3	Алгоритмічні конструкції одно-, дво- і багато- альтернативних розгалужень90
Тема 8.4	Виконання програм із розгалуженнями в покроковому режимі. Вкладені оператори розгалуження93 <i>Практична робота № 7</i>96 <i>Практична робота № 8</i>99

Розділ 9. Цикли

Тема 9.1	Алгоритмічна конструкція повторення та її різновиди: визначені та невизначені цикли, цикли з передумовою і з постумовою. Оператори циклів у мові програмування101
Тема 9.2	Розв'язування задач, у яких використовуються обчислення за ітераційними формулами. Вкладені цикли106 <i>Практична робота № 9</i>110 <i>Практична робота № 10</i>115

Розділ 10. Підпрограми

Тема 10.1	Поняття підпрограми. Оголошення підпрограми, її тіло та оператор її виклику. Поняття функції. Створення й використання власних функцій.117
Тема 10.2	Поняття процедури. Створення й виклик процедур. Підпрограми з параметрами. Поняття локальної та глобальної змінних122
Тема 10.3	Вбудовані процедури та функції: рядкові, перетворення типів даних, генератор псевдовипадкових чисел130 <i>Практична робота № 11</i>142

Розділ 11. Масиви

Тема 11.1	Поняття масиву. Оголошення одновимірного масиву. Індексація елементів. Введення даних у масив і відображення його вмісту151
Тема 11.2	Обчислення підсумкових показників для числового масиву. Пошук даних у масиві158 <i>Практична робота № 12</i>165
Тема 11.3	Обчислення підсумкових характеристик елементів, що задовольняють певні властивості168
Тема 11.4	Вибірання елементів із масиву за певним критерієм174
Тема 11.5	Стандартні функції сортування масиву. Злиття масивів180 <i>Практична робота № 13</i>185
Список літератури189

Навчальне видання
Серія «Курс за вибором»
КАЩЕЄВ Леонід Борисович
КОВАЛЕНКО Сергій Володимирович
КОВАЛЕНКО Світлана Миколаївна
ІНФОРМАТИКА
ОСНОВИ ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ

Навчальний посібник

Код Т13516Р. Редактор *О. В. Костіна*. Технічний редактор *О. В. Сміян*.

Підписано до друку 27.02.2011. Формат 70×100/16. Папір офсетний.

Гарнітура Шкільна. Друк офсетний. Ум. друк. арк. 15,56.

ТОВ Видавництво «Ранок». Свідоцтво ДК № 3322 від 26.11.2008.

61071 Харків, вул. Кібальчича, 27, к. 135.

Адреса редакції: 61145 Харків, вул. Космічна, 21а.

Тел. (057) 719-48-65, тел./факс (057) 719-58-67.

Для листів: 61045 Харків, а/с 3355. E-mail: office@ranok.com.ua

З питань реалізації звертатися за тел.: у Харкові – (057) 712-91-44, 712-90-87;
Києві – (044) 599-14-53, 417-20-80; Білій Церкві – (04563) 6-90-92; Вінниці – (0432) 55-61-10;
Дніпропетровську – (056) 785-01-74; Донецьку – (062) 261-73-17; Львові – (032) 244-14-36;
Житомирі – (0412) 41-27-95, 41-83-29; Івано-Франківську – (0342) 72-41-54;
Кривому Розі – (056) 401-27-11; Миколаєві – (0512) 35-40-39; Одесі – (048) 737-46-54;
Рівному – (0362) 26-34-20; Сімферополі – (0652) 54-21-38 ; Хмельницькому – (0382) 706-316;
Тернополі – (0352) 49-58-36 ; Черкасах – (0472) 51-22-51, 36-72-14;
Чернігові – (0462) 62-27-43

E-mail: commerce@ranok.com.ua.

«Книга поштою»: 61045 Харків, а/с 3355. Тел. (057) 717-74-55, (067) 546-53-73.

E-mail: pochta@ranok.com.ua

www.ranok.com.ua

КУРС ЗА ВИБОРОМ

ІНФОРМАТИКА

Основи візуального програмування

Видання є частиною навчально-методичного комплексу
«Курс за вибором: Основи візуального програмування»

До комплексу входять:

- ☐ навчальний посібник «Інформатика. Основи візуального програмування» із CD-диском
- ☐ зошит для практичних робіт
- ☐ методичні рекомендації для вчителя

Навчальний посібник містить:

- ☐ систематизований матеріал з основ візуального програмування в середовищі Delphi
- ☐ приклади розв'язання задач із детальним описом
- ☐ теоретичні питання для самоконтролю
- ☐ практичні завдання для самостійного виконання
- ☐ практичні роботи, передбачені програмою

CD-диск містить:

- ☐ проекти, тексти та візуальні компоненти всіх програм, розглянутих у посібнику
- ☐ графічні матеріали до завдань

На основі матеріалів CD-диску можна самостійно
створювати візуальні додатки, модифікуючи вихідні тексти
програм

**ЗА ПРОГРАМУВАННЯМ
У ВІЗУАЛЬНИХ СЕРЕДОВИЩАХ — МАЙБУТНЄ!**

ISBN 978-611-540-765-1



9 786115 407651

ВИДАВНИЦТВО
РАНОК
www.ranok.com.ua

Навчально-методична література видавництва «РАНОК»

УСІ КНИГИ ТУТ!

КУПИТИ: WWW.RANOK.COM.UA

ЗАВАНТАЖИТИ: WWW.E-RANOK.COM.UA

ЗАМОВИТИ: pochta@ranok.com.ua

безкоштовний каталог видань: (057) 717-74-55