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, bтипу 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, символьний, 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 виконуваній 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;
```

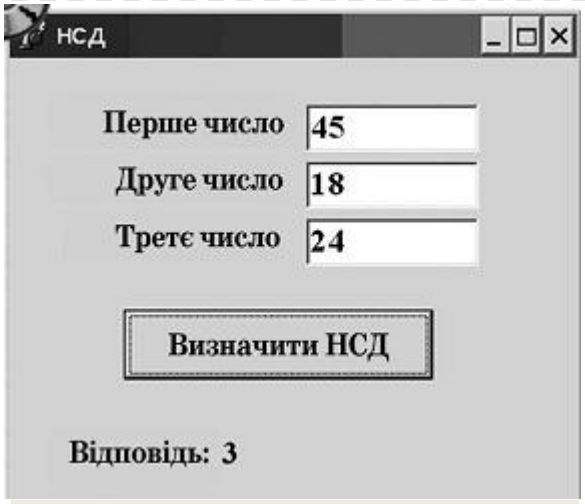

Рис. 1. Пошук НСД трьох цілих додатних чисел

Приклад 1. Реалізувати як функцію пошук НСД (найбільшого спільного дільника) двох цілих додатних чисел. В основній програмі визначити НСД для трьох чисел, враховуючи, що НСД (a, *b, c)* = НСД (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. 1. Слід звернути увагу на використання типу даних cardinal, що являє собою ціле чотирибайтне число без знака.

Програма матиме такий вигляд:

```
unit Unitl;
interface
uses Windows, Messages, Syslltils, Variants, Classes,
Graphics, Controls, Forms, Dialogs, StdCtrls;

type
  TForml = class(TForm)
    Editl: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Labell: Tlabel;
    Label2: Tlabel;
    Label3: Tlabel;
    Buttonl: TButton;
    Label4: TLabel;
    procedure ButtonlClick(Sender: TObject);
end;

 var
   Forml: TForml;
   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 TForml.ButtonlClick(Sender: TObject);
begin
  a:=strtoint(editl.Text); // зчитування першого значення
  b:=strtoint(edit2.Text); // зчитування другого значення
  c:=strtoint(edit3.Text); // зчитування третього значення
  1abe14.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 $\rightarrow$ 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.

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:=Forml.CIientWidth;
  height:=Forml.CIientHeight
end;
```

Присвоювання значень вихідних параметрів усередині підпрограми приводить до присвоювання значень змінних, переданих як аргументи:

```
procedure TForml.ButtonlClick(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.1) — можна сполучати в одній підпрограмі.

Таблиця 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 2. 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*.

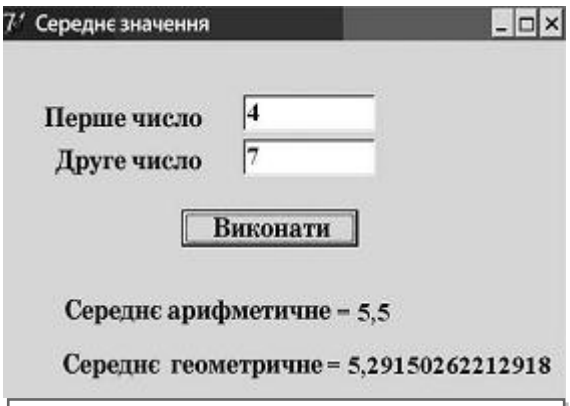

Рис.2. Пошук середнього арифметичного й середнього геометричного значень

Розв'язання. Скористаємося формою, поданою на **рис. 2**. У результаті маємо:

```
unit Unitl;
interface
uses Windows, Messages, SysUtils, Variants, Classes, Graphics,
Controls, Forms, Dialogs, StdCtrls;
type
  TForml = class(TForm)
    Editl: TEdit;
    Edit2: TEdit;
    Labell: TLabel;
    Label2: TLabel;
    Buttonl: TButton;
    Label3: TLabel;
    Label4: TLabel;
    procedure ButtonlClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

 var
  Forml: TForml; a, b, si, 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 TForml.ButtonlClick(Sender: TObject);
begin
  a:=StrToFloat(Editl.Text);
  // прочитати перше число
  b:=StrToFloat(Edit2.Text);
   // прочитати друге число
  Average(a, b, si, s2);
  // виклик процедури Average
  label3.Caption := 'Середнє арифметичне = ' + FloatTostr(sl);
   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. Припустімо, 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 заголовок процедури перед назвою додати ім'я класу форми. Ім'я класу форми в Delphi автоматично формується з літери «T» 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 TForml.EnableButtons(Enabl: Boolean);
```

Якщо виклик створеної нами процедури здійснюється із самої Forml, то можна звертатися до процедури просто за ім'ям. Наприклад, помістимо на форму прапорець типу TCheckBox і в обробнику його події OnClick напишемо:

```
procedure TForml.CheckBoxlClick(Sender: TObject);
begin
   EnableButtons(checkboxl.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 TForml.ButtonlClick(Sender: TObject);
begin
      x:=l; y:=9;
       z:=-2;
       p(x,y,z);
   labell.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.