§ 8. Підпрограми

Підпрограми призначені для реалізації алгоритмів опрацювання окремих частин деякої складної задачі. Вони дають змогу реалізовувати концепцію структурного програмування, суть якого полягає в розкладанні складної задачі на послідовність простих підзадач і в складанні для алгоритмів розв'язування кожної підзадачі відповідних підпрограм. Розрізняють два види підпрограм — підпрограми-процедури та підпрограми-функції. Підпрограми поділяються на стандартні та підпрограми користувача. Стандартні підпрограми

створювати не потрібно — вони містяться у стандартних модулях System, Crt, Dos, Graph тощо. ***Підпрограма користувача*** — це поіменована група команд, яку створюють і описують в основній програмі в розділах **procedure** або **function** і до якої звертаються з будь-якого місця програми потрібну кількість разів.

1. Процедури (procedure). Загальний опис процедури:

```
procedure <назва> (<список формальних параметрів>);
        <розділи описів і оголошень процедури>;
    begin
        <розділ команд процедури>
    end;
```

У списку ***формальних*** параметрів перераховують *змінні разом із зазначенням їхніх типів.* Розрізняють ***параметри-аргументи*** (інший термін: параметри-значення) — вхідні дані для процедури, і ***параметри-результати*** (інший термін: параметри-змінні), через які можна повертати результати роботи процедури в основну програму. Перед списками параметрів-результатів *кожного* типу записують слово **var**. Зауважимо, що масиви фіксованих розмірів у списках формальних параметрів не можна описувати за допомогою слова **array** (див. зразки програм).

Розділи описів і оголошень у підпрограмах мають таку саму структуру як і в основній програмі.

Приклад. Розглянемо процедуру з назвою Cina, яка визначає с — вартість *k* хвилин телефонної розмови з похвилинною оплатою 0.6 грн. + 20% ПДВ.

```
procedure Cina(k:integer; var c:real);
  begin
    c:=k*0.6;  c:=c+0.2*c;
  end;
```

У наведеному прикладі k є формальним параметром-аргументом, с — формальним параметром-результатом.

До процедури звертаються з розділу команд основної програми або іншої підпрограми. Процедуру викликають за допомогою команди виклику:

```
<назва процедури> (<список фактичних параметрів>);
```

Параметри, які записують у команді виклику процедури, називаються ***фактичними***. Фактичними параметрами-аргументами можуть бути сталі, змінні, вирази, а параметрами-результатами — лише змінні. Типи даних тут не зазначають.

Між фактичними і формальними параметрами має бути відповідність за кількістю й типами. Зверніть увагу, відповідні фактичні та формальні параметри можуть мати різні імена.

Команда виклику функціонує так: значення фактичних параметрів присвоюються відповідним формальним параметрам процедури, виконується процедура, визначаються параметри-результати, значення яких надаються (повертаються) відповідним фактичним параметрам у команді виклику.

Змінні, описані в розділі описів основної програми, називаються ***глобальними***. Вони діють у всіх підпрограмах, з яких складається програма. Змінні, описані в розділі описів конкретної процедури, називаються ***локальними***. Вони діють тільки в межах даної процедури.

Процедури можуть отримувати і повертати значення не тільки через параметри-результати, але й через глобальні змінні. Тому списків параметрів у процедурі може і не бути.

Задача 1. Розв'язати задачу про кількість викликів на АТС з попереднього параграфа, використовуючи три процедури: 1) для визначення кількості викликів за кожну секунду (надамо їй назву Kilvykl); 2) для обчислення суми викликів за перші 10 секунд (Sumavykl); 3) для визначення найбільшої кількості викликів за деяку секунду (Maxkilvykl). Використати функцію random.

```
program ATS1;
uses Crt;
type vyklyk= array[1..10] of integer;
var y:vyklyk;  max,s:integer;
procedure Kilvykl(var y:vyklyk); {Процедура Kilvykl визначає
var i:integer;                   кількість викликів кожної секунди}
begin
   for i:=1 to 10 do
      begin
      y[i]:=random(i);
      writeln('y(' ,i ,')=' ,y[i]:5);
      end;
end;
procedure Sumavykl(y:vyklyk; var s:integer); {Процедура
                  обчислює суму викликів за перші 10 секунд}
var i:integer;
begin
   s:=0; for i:=1 to 10 do s:=s+y[i];
   writeln('Сума викликів S=',s:3);
end;
procedure Maxkilvykl(y:vyklyk; var max:integer);
var i:integer;                   {Процедура Maxkilvykl визначає
begin                            найбільшу кількість викликів}
   max:=y[1];                    {за деяку секунду}
   for i:=2 to 10 do
```

```
        if max<y[i] then max:=y[i];
    write('Максимальна кількість викликів за одну ');
    writeln('секунду дорівнює',max:3)
end;
begin
    clrscr;
    randomize;
    Kilvykl(y);                {Виклик процедури Kilvykl}
    Sumavykl(y,s);             {Виклик процедури Sumavykl}
    Maxkilvykl(y,max);         {Виклик процедури Maxkilvykl}
    readln
end.
```

Завдання 1. Розв'яжіть задачу № 14 свого варіанта.

2. Функції (function). Функція, на відміну від процедури, може повертати в місце виклику лише один результат простого стандартного типу.

Загальний опис функції:

```
function <назва>(<список формальних параметрів>) : <тип функції>;
    <розділи описів і оголошень функції>;
begin
    <розділ команд функції, де має бути така команда:
    назва:=вираз>
end;
```

У розділі команд функції має бути команда присвоєння значення деякого виразу назві функції. Результат функції повертається в основну програму через її назву (як і випадку використання стандартних функцій, таких як sin, cos). Виклик функції здійснюється лише з виразів так:

```
    <назва> (<список фактичних параметрів>).
```

Приклад. Створимо функцію для обчислення $\mathrm{tg}(x)$ та обчислимо значення виразу $\mathrm{tg}(x)+\mathrm{ctg}(x)+\mathrm{tg}^2(x)$.

```
program Myfunc;
uses Crt;
var x,y:real;
function tg(x:real):real;
begin
    tg:=sin(x)/cos(x)
end;
begin clrscr;
    writeln('Введіть x');
    readln(x);
    y:=tg(x)+1/tg(x)+sqr(tg(x));
    writeln('y=', y:5:2); readln
end.
```

Задача 2. Розв'язати задачу про виробництво цукерок на фабриці з попереднього параграфа, використовуючи функції і процедури користувача.

```
program Fabryka1;
uses Crt;
const n=5;
type  vytraty = array [1..n,1..n] of real;
var   imin:integer; a:vytraty;
function func(i,j:integer):real;
begin
   func:=2*abs(sin(i))+j;
end;
procedure Table(var a:vytraty);
var i,j:integer;
begin
   writeln('                    Вид сировини');
   writeln('          1      2      3      4      5');
   for i:=1 to n do          {Утворимо таблицю затрат}
      begin
      write(i,' сорт');
      for j:=1 to n do
         begin
         a[i,j]:=func(i,j);  {Використаємо створену функцію}
         write( a[i,j]:7:2); {Роздрукуємо елементи i-го рядка}
         end;
      writeln               {Перейдемо на новий рядок}
      end;
end;
procedure MinSyrov(a:vytraty; var imin:integer);
var i,j:integer; min:real;
begin
   imin:=1; min:=a[1,3];
   for i:=2 to n do
      if a[i,3]<min then
         begin
         min:=a[i,3]; imin:=i
         end;
   writeln('Найменше сировини третього виду ');
   writeln('необхідно для цукерок ',imin,' сорту')
end;
begin
   clrscr;
   Table(a);                    {Виклик процедури Table}
   MinSyrov(a,imin);            {Виклик процедури MinSyrov}
   readln
end.
```

Завдання 2. Розв'яжіть задачу № 15 свого варіанта, використовуючи функції та процедури.

3. Рекурсивні функції. ***Рекурсією*** *називається алгоритмічна конструкція, де підпрограма викликає сама себе.* Рекурсія дає змогу записувати циклічний алгоритм, не використовуючи команду циклу. Розглянемо спочатку поняття стеку.

Стек — це структура даних в оперативної пам'яті, де дані запам'ятовуються і зберігаються за принципом «перший прийшов — останнім пішов». Аналогом у військовій справі є ріжок для набоїв до автомата.

Приклад. Рекурсивна функція обчислення суми цілих чисел від *a* до *b* має вигляд

```
function Suma(a,b:integer):integer;
begin
   if a=b then Suma := a          {Це стоп-умова рекурсії}
   else Suma := b + Suma(a, b-1)  {Це неявний цикл}
end;
```

Обчислимо функцію Suma(3, 5). Формально можна записати Suma(3, 5) = 5 + Suma(3, 4) = 5 + 4 + Suma(3, 3) = 5 + 4 + 3. Система виконує такі обчислення за два етапи: 1) спочатку формує стек, куди заносить числа 5, 4, 3. На другому етапі числа додає у зворотній послідовності (оскільки вони надходять зі стеку): 3+4+5 = 12.

Задача 3. Скласти рекурсивну функцію Factorial для обчислення факторіала числа $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$, ($0! = 1$, $1! = 1$), яка грунтується на багаторазовому (рекурсивному) застосуванні формули $n! = n \cdot (n-1)!$.

```
function Factorial(n : integer) : integer;
begin
   if n = 0 then Factorial := 1    {Це стоп-умова}
            else Factorial := n * Factorial(n-1)
end;
```

Обчислимо 4!: Factorial(4) = 4 · Factorial(3) = 4 · 3 · Factorial(2) = 4 · 3 · 2 · Factorial(1) = 4 · 3 · 2 · 1 · Factorial(0) = 4 · 3 · 2 · 1 · 1. У стек будуть занесені числа 4, 3, 2, 1, 1. Результат утвориться так: 1 · 1 · 2 · 3 · 4 = 24.

Зауваження. Застосовуючи рекурсію, потрібно правильно складати стоп-умови, які забезпечують закінчення циклічних обчислень.

Завдання 3. Скласти програму розв'язування задачі 6, використовуючи рекурсивні функції.

4. Відкриті масиви. У списках *формальних параметрів* підпрограми можна описувати так звані ***відкриті масиви*** (масиви заздалегідь невідомого розміру) так:

<назва масиву> : **array of** <назва базового типу>;

У підпрограмі мінімальне значення індекса такого масиву є 0, а номер останнього елемента дає стандартна функція **high**(<назва масиву>). Нумерація елементів такого формального масиву *у підпрограмі* починається з нуля. Відкритий масив використовують для почергового опрацювання у процедурі масивів різних розмірів.

Задача 4. Використовуючи підпрограми, утворити масив y, елементи якого задані формулою $y_m = fy(m)=10\cos(m)+2$, $m = \overline{1,7}$, та масив g з елементами $g_n=fg(n)=n^2/2$, $n = \overline{1,9}$. Обчислити в цих масивах суми елементів більших, ніж 2. Вивести на екран результати обчислень.

```
program MyProcedure;
{$F+}
uses Crt;
type  myfunc=function(n:integer):real;
var   y:array [1..7] of real;
      g:array [1..9] of real;

function fy(m:integer):real;
begin
   y:=10*cos(m)+2
end;

function fg(n:integer):real;
begin
   g:=n*n/2
end;

procedure Utvoryty(f:myfunc; var z: array of real);
var i:integer;
begin
   for i:=0 to high(z) do
      begin
      z[i]:=f(i+1); write(z[i]:5:2);
      end;
   writeln
end;

function suma(z: array of real):real;
var i:integer; s: real;
begin
   s:=0;
   for i:=0 to high(z) do
      if z[i]>2 then s:=s+z[i];
   suma:=s
end;
```

```
begin
   clrscr;
   Utvoryty(fy, y);
   Utvoryty(fg, g);
   write('Сума потрібних елементів  y - ');
   writeln('S=', suma(y):6:2);
   write('Сума потрібних елементів  g - ');
   writeln('S=', suma(g):6:2);
   readln
end.
```

Зауваження. Зверніть увагу на утворення та застосування нового типу – типу функції: **type** myfunc=**function**(n:integer):real. До цього типу віднесено конкретні функції $fy(x)$ та $fg(x)$. Завдяки типу myfunc можна за допомогою однієї процедури утворювати різні масиви. У зв'язку з цим використано директиву {$F+}, яка підтримує необхідну модель (far-модель) виклику функцій.