Процедурне програмування

Методи процедурного програмування базуються на моделі побудови програми як деякої сукупності функцій. Прийоми програмування пояснюють, як розробляти, організовувати та реалізовувати функції, що складають програму.

Функція - модуль, що містить деяку послідовність операцій. Її розробка та реалізація у програмі може розглядатися як побудова операцій, що вирішують конкретну задачу. Однак взагалі функція може розглядатися окремо як єдина абстрактна операція, і, щоб її використовувати, користувачеві необхідно зрозуміти інтерфейс функції - її вхідні дані та результати виконання. Припустимо, необхідно розробити функціональний модуль, що розв'язує наступне завдання: існує вхідний список певних даних, який необхідно відсортувати, переставляючи його елементи у визначеному порядку.

Ця функція може бути описана, як абстрактна операція сортування даних, що може бути частиною вирішення деякої підмножини задач. Функція, що реалізує цю операцію, може бути використана у багатьох програмах, якщо вона створена як абстракція, що не залежить від контексту програми.

Функції мають параметри, тому їх операції узагальнені для використання будь-якими фактичними аргументами відповідного типу. Що є вхідними даними для функції ? Вхідними даними для неї є аргументи та глобальні структури даних, що використовуються функцією. Вихідними даними є ті значення, які функція повертає, а також зміни глобальних даних, модифікації. Будь-яка програма на мові С складається з функцій, причому одна з яких обов'язково повинна мати ім'я ***main().***

Функції

Функціями називають самостійні, логічно завершені фрагменти програм, що мають власне ім’я та призначені для виконання заданих дій, останньою з яких може бути повернення деякого значення. Функції формують окремий тип даних мови С. Використання функцій дає змогу структуризувати програму (поділити складні процеси на окремі частини, виділити основні кроки в алгоритмі, а потім розкрити деталі реалізації в окремих функціях), уникнути багаторазового запису одних і тих самих дій та скористатись раніше зробленим, уможливлює розпаралелення процесу програмування, спрощує пошук помилок та внесення доповнень, робить програму лаконічною.

Всі функції, включаючи ***main ( )***, рівноправні. Особливість ***main ( )*** у тому, що вона розпочинає роботу програми, а її завершення означає кінець виконання усієї програми. Саме в функції ***main ( )*** найчастіше вказують основні кроки алгоритму. Кожна функція С-програми записується окремо, вкладати функції одна в одну не можна.

Функцію програми (або групу функцій) можна відкомпілювати автономно і зберігати в окремому ***obj***-файлі. Об’єктні коди долучаються до складу виконавчого коду програми на етапі редагування зв’язків (компонування).

Структура функцій

Синтаксис опису функції :

тип_поверт_значення ім'я_функції ([список_аргументів])

{

оголошення внутрішніх змінних

оператори тіла функції

}

У заголовку вказується тип значення, яке повертає функція в точку її виклику після завершення виконання. Тип значення, яке повертається функцією може бути будь-яким, за виключенням масиву та функції (але може бути покажчиком на масив чи функцію). Якщо функція не повертає значення, то вказується тип ***void***.

Ім’я функції формується за правилами запису ідентифікаторів.

Після імені функції в круглих дужках оголошується список параметрів - їх називають формальними параметрами або аргументами функції. Кожен з параметрів оголошується у списку із окремим зазначенням типу. Функція може мати порожній список, але круглі дужки ***( )*** вказуються обов'язково.

```
 /* піднесення дійсного числа до цілого степеня  */
double  Pow (double base, int n)
{
   double prod;
  for ( prod = 1.0; n>0; n-- )
        prod *= base;
  return prod;
}
```

Тіло функції реалізує дії, які повинна виконати ця функція. Внутрішні змінні, що необхідні для роботи функції, називають локальними змінними. Вони оголошуються на початку тіла функції перед першим оператором. В прикладі – це змінна ***prod***

Областю дії внутрішніх змінних функції та її формальних параметрів є область тіла даної функції. Такі змінні створюються на час виконання функції і стають невизначеними після її завершення.

Функція завершує роботу, коли виконано всі оператори її тіла або коли зустрінеться оператор ***return*** . Якщо функція повертає певне значення (її тип відмінний від ***void***), то це значення має бути передане через вираз, записаний в операторі ***return.*** Значення виразу заноситься у буфер обміну, звідки його можна отримати у тій точці програми, з якої викликають функцію.

Виклик функції. Прототипи функцій.

Оператори тіла функції виконуються тоді, коли здійснюється звертання до функції. Виклик функції:

ім'я_функції (список фактичних параметрів)

тут *список фактичних параметрів* – це послідовність виразів, кожен з яких задає значення відповідного формального параметра Значення, повернене функцією, можна використовувати у всіх виразах як звичайний операнд, тип якого збігається з типом значення функції.

Приклад звертання до функції ***Pow ( )*** , оголошеної вище:

r = (Pow (x, 3) – Pow (y, 4))/2

Дозволяється автономно (через окремий оператор) викликати функції, які повертають певне значення. У цьому випадку значення, яке повертає функція, ігнорується. Приклад – виклик бібліотечної функції ***printf ()***, яка повертає значення типу ***int.***

Функції типу ***void*** (ті, що не повертають значення), подібні до процедур Паскаля. Оператор ***func();*** виконує функцію ***void func()*** , тобто передасть керування функції, доки не виконаються усі її оператори. Коли функція поверне керування в основну програму, тобто завершить свою роботу, програма продовжить своє виконання з того місця, де розташовується наступний оператор за оператором ***func().***

```
#include<stdio.h>
void func1(void);
void func2(void);
 void  main()
{
   func1();
   func2();
}
void func1 (void)
{      /* тіло */       }
void func2 (void)
{       /* тіло */       }
```

Звернемо увагу на те, що текст програми починається з оголошення прототипів функцій - схематичних записів, що повідомляють компілятору ім'я та форму кожної функції у програмі. У великих програмах це правило примушує Вас планувати проекти функцій та реалізовувати їх таким чином, як вони були сплановані. Будь-яка невідповідність між прототипом (оголошенням) функції та її визначенням (заголовком) призведе до помилки компіляції. Кожна з оголошених функцій має бути визначена у програмі, тобто заповнена операторами, що її виконують. Спочатку йтиме заголовок функції, який повністю співпадає з оголошеним раніше прототипом функції, але без заключної крапки з комою. Фігурні дужки обмежують тіло функції. В середині функцій можливий виклик будь-яких інших функцій, але неможливо оголосити функцію в середині тіла іншої функції. Нагадаємо, що Паскаль дозволяє працювати із вкладеними процедурами та функціями.

Прототипи стандартнних бібліотечних функцій записані у заголовочних файлах ***.h***

Приклад програми, що розв'язує тривіальне завдання - обчислює корені звичайного квадратного рівняння:

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <math.h>
float A,B,C;
/*функція прийому даних*/
void GetData()
{
        clrscr();
        printf("Input A,B,C:");
        scanf("%f%f%f",&A,&B,&C);
}
/*функція запуску основних обчислень*/
void Run()
{
        float D;
        float X1, X2;
```

```
if ((A==0) && (B!=0))
        {
                X1 = (-C)/B;
                printf("\nRoot: %f",X1);
                exit(0);
        }
        D = B*B - 4*A*C;
        if (D<0) printf("\nNo roots...");
        if (D==0)
        {
                X1=(-B)/(2*A);
                printf("\nTwo equal roots: X1=X2=%f",X1);
        }
        if (D>0)
        {
                X1 = (-B+sqrt(D))/(2*A);
                X2 = (-B-sqrt(D))/(2*A);
                printf("\nRoot X1: %f\nRoot X2: %f",X1,X2);
        }
  }
```

```
/*головна функція програми*/
void main()
{
    GetData();
    Run();
}
```

Якщо вказано, що функція повертає значення типу ***void***, то її виклик слід організовувати таким чином, аби значення, що повертається, не використовувалося б. Просто кажучи, таку функцію неможливо використовувати у правій частині виразу.