**Створення і використання функцій**

Принципи програмування на мові Сі засновані на понятті функції. Ми вже розглянули кілька функцій:printf( ), scanf( ), getchar( ), putchar( ). Ці функції є системними, однак ми створили і кілька своїх власних функцій під загальним ім'ям main( ). Виконання програми завжди починається з команд, що містяться в функції main (), потім остання викликає інші функції. Розглянемо питання, як створювати свої власні функції і робити їх доступними для функції main( ), а також для інших функцій.

**Функція** - це самостійна одиниця програми, спроектована для реалізації конкретного завдання. Виклик функцій призводить до виконання деяких дій. Наприклад, при зверненні до функції printf( ) здійснюється виведення даних на екран. Загалом, функції можуть виконувати дії і отримувати значення величин, які використовуються в програмі.

Чому ми користуємося функціями? По-перше, вони позбавляють нас від повторного програмування. Якщо конкретне завдання в програмі необхідно виконати кілька разів, ми напишемо відповідну функцію тільки один раз, а потім будемо викликати її завжди, коли потрібно. По-друге, ми можемо застосовувати одну функцію, наприклад putchar( ), в різних програмах.

|  |  |
| --- | --- |
| ! | Якщо деяка задача виконується тільки в одній програмі, краще оформити її рішення у вигляді функції, тому що функції підвищують рівень модульності програми і, отже, полегшують її читання, внесення змін і корекцію помилок. Додаткова перевага зазначеного підходу полягає в тому, що якщо ми створимо функції загального вигляду, то їх можна буде використовувати і в інших програмах. |

Багато програмістів думають про функції, як про "чорну скриньку". Вони задають її через інформацію, що надходить і отримані результати. Все, що відбувається всередині чорної скриньки, їх не стосується. Що нам потрібно знати про функції? Потрібно знати, як їх можна визначати, як до них звертатися і як встановлювати зв'язки між функцією і програмою, її викликає.

Абстракція управління в мові Сі забезпечується за допомогою функцій. Всі функції можуть бути рекурсивними. У мові Сі відсутні підпрограми (процедури), проте повернення функцією значення в зухвалу програму не обов'язковий, отже, функції можуть бути розділені на дві категорії - функції, які повертають значення, і функції, що не повертають значення в зухвалу програму (підпрограми).

*Визначення функцій, які повертають значення, мають такий вигляд*:

[static]

тип-результату ім'я-функції (формальні аргументи)  
опис формальних параметрів

{

Тіло функції

}

де ім'я функції - правильний ідентифікатор, а тіло функції має вигляд

визначення і опис  
оператори

|  |  |
| --- | --- |
| ! | Все, що взято в квадратні дужки, може і не бути static - ми розглянемо в лекції 10. Вказівка типу-результату функції в мові Сі не є обов'язковим. Якщо тип результату не вказано, то передбачається, що результат має тип int. Оскільки вказівку типу функції призводить до більшої ясності і легкості читання програми, а також спрощує знаходження в ній помилок, тип функції завжди повинен бути вказаний явно. |

Як результат функція не може повертати масив (див. Лекцію 13) або функцію, але може повертати покажчик на масив або функцію. Виконання функції, що повертає значення, зобов'язана завершитися оператором return виду

return e;

який забезпечує видачу результату e. Функція, що повертає значення, може містити більше одного оператора

return.

Визначення функції, не повертає значення, мають наступний формат:

[static] void ім'я-функції (формальні аргументи)  
опис формальних параметрів

{

Тіло функції

}

Виконання такої функції завершується, якщо виконано її тіло або оператор return виду

return;

Функція, що не повертає значення, може містити більше одного оператораreturn.

*Клас пам’яті* static (Необов'язковий) обмежує видимість функції і інших зовнішніх визначень. Функція з класом пам'яті static невидима поза містить її файлу. Якщо в тексті програми є звернення до функції, то необхідний опис функції, який в тексті повинен бути вміщений раніше її визначення. Описи функції мають наступну форму:

[static или extern] тип-результату ім'я-функції ( );

[static или extern] void ім'я-функції ( );

Якщо в описі не вказано клас пам'яті (див. Лекцію 10), то за замовчуванням, передбачається extern.

|  |  |
| --- | --- |
| ! | Для компіляторів, нездатних обробляти тип void, програміст може визначити тип void як  #define void int  і використовувати його для визначення функцій, які не повертають значення. Рекомендується слідувати цій угоді для поліпшення ясності програми. Однак в таких випадках компілятор буде не в змозі виявити некоректне використання цих функцій для повернення значень, оскільки насправді розглядаються функції повертають значення, і ці значення мають тип int. |

### Аргументи функції

Формальний аргумент - змінна в функції яка викликається, а фактичний аргумент - конкретне значення, присвоєне цій зміннії викликаючою програмою. Фактичний аргумент може бути константою, змінною або більш складним виразом. Незалежно від типу фактичного аргументу він спочатку обчислюється, а потім його величина передається функції. Фактичний аргумент - це конкретне значення, яке присвоюється змінної, званої формальним аргументом.

Якщо для зв'язку з деякою функцією потрібно більше одного аргументу, то поряд з ім'ям функції можна задати список аргументів, розділених комами. Наприклад:

void print\_num(i,j)

int i,j;

{

printf("значення i=%d. Значення j=%d.", i,j);

}

Звернення в програмі до даної функції буде таким:

print\_num(6,19);

Особливу увагу потрібно приділити правилам передачі аргументів при зверненні до функцій. Синтаксис мови Сі передбачає тільки один спосіб передачі аргументів - передачу за значеннями. Це означає, що формальні параметри (аргументи) функції локалізовані в ній, тобто недоступні поза визначення функції і ніякі операції над формальними параметрами в тілі функції не змінюють значення фактичних параметрів. Передача параметрів за значенням передбачає наступні дії:

* При компіляції функції виділяються ділянки пам'яті для формальних параметрів. Формальні параметри виявляються внутрішніми об'єктами функції. При цьому для параметрів типу float формуються об'єкти типу double. Для параметрів типу char, short int створюються об'єкти типу int. Якщо параметром є масив, то формується покажчик на початок цього масиву і він служить поданням масиву-параметра в тілі функції (про покажчику масивів описано в 12 лекції даного курсу).
* Обчислюються значення виразів, використаних в якості фактичних параметрів при виклику функції.
* Значення виразів- фактичних параметрів заносяться в ділянки пам'яті, виділені для формальних параметрів функції. При цьому float перетворюється в double, a char, short int - в тип int.
* У тілі функції виконується обробка з використанням значень внутрішніх об'єктів-параметрів, і результат передається в точку виклику функції як повертається нею значення.
* Ніякого впливу на фактичні параметри функція не робить.

### Повернення значень

Напишемо функцію, яка обчислює абсолютну величину числа. Абсолютна величина числа - це його значення, якщо відкинути знак. Наприклад, абсолютна величина 125 - це 125, а абсолютна величина числа (-125) - це теж 125. Назвемо цю функцію abs (). Входом для цієї функції буде будь-яке число, для якого ми хочемо знайти абсолютну величину. Вихідна величина повертається за допомогою ключового слова мови Сі - return. Оскільки функція abs () повинна бути викликана іншою функцією, ми створимо просту функцію main (), основною метою якої буде перевірка, чи працює функція abs (). Програма, спроектована для того, щоб перевірити роботу функції саме таким чином, називається драйвером. Драйвер піддає функцію послідовним перевірок. Якщо результати виявляються задовільними, то її можна помістити в програму, яка заслуговує на більшу увагу. Термін драйвер зазвичай відноситься до програм, керуючим роботою пристроїв:

/\*драйвер\*/

int main( )

{

int a=100, b=0, c=-122;

int d,e,f;

d=abs(a); e=abs(b); f=abs(c);

printf("%d, %d, %d\n",d,e,f);

}

int abs(int x) /\* функція, що обчислює величину числа \*/

{

int y;

y = (x < 0) ? -x : x;

/\* повертає значення y програми що викликає \*/

return(y);

}

Результат роботи програми виглядає так:

100 0 122

Ключове слово return вказує на те, що значення виразу, укладеного в круглі дужки, буде присвоєно функції, що містить це ключове слово (оператор). Тому, коли функція abs () вперше викликається драйвером, значенням abs (a) буде число 100, яке потім присвоюється змінній d.

Змінна y є внутрішнім об'єктом функції abs (), але значення y передається в програму що викликає за допомогою оператора return. Дія, яку робить оператор d=abs(a);

По іншому можна виразити так:

abs(a);

d=y;

Такий записом ми скористатися не можемо! Програма яка викликає навіть не підозрює про те, що змінна y існує.

Оператор return надає і деякі інші дії. Він завершує виконання функції і передає управління наступного оператора в функції яка викликає. Це відбувається навіть у тому випадку, якщо оператор return є не останнім оператором тіла функції:

/\* Функція, що обчислює абсолютну величину числа, друга версія \*/

int abs(int x)

{

if(x < 0)

return(-x);

else

return(x);

}

Ця версія програми простіше. Для користувача обидві версії невиразні, так як у них є один і той же вхід, і вони забезпечують один і той же вихід. Тільки внутрішні структури обох функцій різні.

/\* третя версія функції abs( ) \*/

int abs(int x)

{

if(x < 0) return(-x);

else return(x);

printf("Работа закінчена!\n");

}

Наявність оператора return перешкоджає тому, щоб оператор друку коли-небудь виконався в програмі.  
Можна користуватися оператором

return;

Його застосування призводить до того, що функція, в якій він міститься, завершує своє виконання і управління повертається в зухвалу функцію. Оскільки у даного оператора відсутній вираз в дужках, ніяке значення не передається функції.

Всі функції в програмі, написаній на мові Сі, рівноправні. Кожна з них може викликати будь-яку іншу функцію і, в свою чергу, кожна може бути викликана будь-якою іншою функцією. Це робить функції мови Сі дещо відмінними від процедур Паскаля, оскільки процедури в Паскалі можуть бути вкладені в інші процедури, причому процедури, описані в різних процедурах, є недоступними для процедур, описаних в інших незалежних процедурах.

У процедури main( ). є специфіка. Вона полягає в тому, що після складання програми, що складається з декількох функцій, її виконання починається з першого оператора функції main( ).

Кожна функція може викликати саму себе. Дія, заключається в тому, що функція викликає сама себе, називається рекурсією.

***Рекурсивної називають функцію,***яка прямо або побічно сама викликає себе. Саме можливість прямого або непрямого виклику дозволяє розрізняти пряму або опосередковану рекурсію. При кожному зверненні до рекурсивної функції створюється новий набір об'єктів автоматичної пам'яті, локалізованих в тілі функції. Функція називається **побічно рекурсивною** в тому випадку, якщо вона містить звернення до іншої функції, що містить прямий або непрямий виклик обумовленої (першої) функції. У цьому випадку за текстом визначення функції її рекурсивність (непряма) може бути не видна. Якщо в тілі функції явно використовується виклик цієї ж функції, то тут має місце пряма рекурсія.

|  |  |
| --- | --- |
| ! | Так як при кожному зверненні до рекурсивної функції створюється новий набір об'єктів автоматичної пам'яті, локалізованих в тілі функції, то при використанні рекурсивних алгоритмів з глибокої вкладеністю рекурсії може швидко статися переповнення стека реалізації рекурсій, тому надійніше використовувати ітераційні алгоритми. Наприклад, нехай потрібно реалізувати "серветку Серпінського" (геометричний фрактал). Як вона утворюється? Вимальовується трикутник і в ньому середні лінії. В утворених при кутах вихідного трикутника нових трикутниках знову малюються середні лінії і так далі до заданого порядку вкладеності рекурсії. Отримана "серветка Серпінського" допускає інше, не рекурсивне, побудова за допомогою моделювання методом Монте-Карло. |

### Локальні змінні

Змінні, відомі тільки одній функції, а саме тій, яка їх містить, називаються **локальними змінними**. Тепер ми підкреслюємо, що локальні змінні є дійсно локальними. Навіть в тому випадку, коли ми використовуємо одне і те ж ім'я для змінних в двох різних функціях, компілятор вважає їх різними змінними.

**знаходження адрес**

В результаті виканання операції & визначається адреса комірки пам'яті, яка відповідає змінній. Якщо age - ім'я змінної, то &age – її адреса. Можна уявити собі адресу як осередок пам'яті, але можна розглядати його і як мітку, яка використовується компілятором для ідентифікації змінної. Припустимо, ми маємо оператор

age=105;

Нехай також адреса осередку, де розміщується змінна age - 15125. В результаті виконання оператора

printf("%d %d\n", age, &age);

Отримаємо 105 15125.

### 

### Покажчики, перше знайомство

***Покажчик -*** деяке символічне уявлення адреси. В описаному прикладі & age означає покажчик на змінну age. Фактична адреса - це число, а символічне уявлення адреси & age є константою типу покажчик. Адреса комірки, що відводиться змінній age, в процесі виконання програми не змінюється. У мові Сі є і змінні типу покажчик. Точно так само як значенням змінної типу char є символ, а значенням змінної типу int - ціле число, значенням змінної типу покажчик служить адреса деякої величини. Якщо ми дамо вказівником ім'я ptr, то зможемо написати, наприклад, такий оператор:

ptr = &age;

/\* привласнює адресу age змінній ptr \*/

Ми говоримо в цьому випадку, що ptr вказує на age. Різниця між двома формами запису, ptr і &age, полягає в тому, що ptr - це змінна, в той час як &age - константа. У разі необхідності ми можемо зробити так, щоб змінна ptr вказувала на який-небудь інший об'єкт:

ptr = &name;

/\*ptr вказує на name, а не на age\*/

Тепер значенням змінної ptr є адреса змінної name.

#### Операція непрямої адресації \*

Припустимо, ми знаємо, що в зміннійptr міститься посилання на змінну name. Тоді для доступу до значення цієї змінної можна скористатися операцією непрямої адресації \*:

val = \*ptr;

/\* визначення значення, на яке вказує ptr \*/

Останні два оператора, взяті разом, еквівалентні наступному:

val = name;

#### Опис покажчиків

Приклади опису покажчиків:

int \*pi; /\* покажчик на змінну типу цілого \*/

char \*pc; /\*покажчик на символьну змінну\*/

float \*pf, \*pg;

/\* покажчики н азмінну з плаваючою крапкою\*/

Специфікація типу задає тип змінної, на яку посилається покажчик, а символ зірочка (\*) визначає саму змінну як покажчик. Опис виду int \*pi коже , що pi – це покажчик і що \*pi - величина типу int.

При виконанні функції інформація про змінну може передаватися функції в двох видах. Якщо ми використовуємо форму звернення

function1(x);

відбувається передача значення змінної x. Якщо ж ми використовуємо форму звернення

function2(&x);

відбувається передача адреси змінної x. Перша форма звернення вимагає, щоб визначення функції включило в себе формальний аргумент того ж типу, що й x:

function1(num)

int num;

Друга форма звернення вимагає, щоб визначення функції включало в себе формальний аргумент, який є вказівкою на об'єкт відповідного типу:

function2(&ptr)

int \*ptr;

|  |  |
| --- | --- |
| ! | Користуйтеся першою формою, якщо вхідне значення необхідної функції для деяких обчислень або дій, і другою формою, якщо функція повинна буде змінювати значення змінних в програмі що викликає. Програмісти, що працюють на мові Паскаль, можуть помітити, що перша форма виклику аналогічна по поводженню з параметром-значенням, а друга-з параметром-змінною. |

#### Підведемо підсумки по покажчикам

Коли за знаком & слідує ім'я змінної, результатом операції є адреса зазначеної змінної.  
Коли за знаком \* слідує покажчик на змінну, результатом операції є величина, вміщена в клітинку з вказаною адресою.

Приклад:

age = 105;

ptr =&age;/\*покажчик на age\*/

val= \*ptr;

Результатом виконання цього фрагмента є присвоювання значення 105 змінної val.

Типове визначення функції має такий вигляд:

ім'я (список аргументів)  
опис аргументів  
тіло функції

Наявність списку аргументів і описів не обов'язкові. Змінні, відмінні від аргументів, описуються всередині тіла, яке закривається в фігурні дужки.

Аргументи при цьому використовуються для передачі значень з програми яка викликає в функцію. Використання ключового слова return дозволяє передавати в викликаючу програму одне значення з функції, що викликається. Зазвичай виконання функції не робить ніякого впливу на значення змінних викликаючої програми. Щоб мати можливість безпосередньо змінювати значення змінних викликаючої програми, необхідно використовувати покажчики як аргументи. Це може виявитися необхідним в разі, якщо в викликаючу програму потрібно передати більше ніж одне значення.

Функції повинні мати той же тип, що і значення, які вони повертають в якості результатів. За умовчанням передбачається, що функції мають тип int. Якщо функція має інший тип, він повинен бути зазначений і в викликаючій програмі, і в самому визначенні функції.

*Функції зі змінною кількістю аргументів*

У мові Сі допустимі функції, кількість аргументів у яких при компіляції функції не фіксоване. Кількість і тип аргументів стає відомим тільки в момент виклику функції, коли явно заданий список фактичних аргументів (параметрів). При визначенні та описі таких функцій, що мають списки параметрів невизначеної довжини. Специфікація формальних параметрів закінчується комою і трьома крапками:

тип ім'я (специфікація-явних-параметрів,...);

Тут тип - тип поверненого функцією значення; имя – ім’я функції.

Специфікація явних параметрів - список специфікації параметрів, кількість і типи яких фіксовані і відомі в момент компіляції. Ці параметри є обов'язковими. Кожна функція зі змінною кількістю параметрів повинна мати хоча б один обов'язковий параметр. Після списку явних (обов'язкових) параметрів ставиться кома, а потім - три крапки. Компілятор знає, що подальший контроль відповідності кількості і типів параметрів при обробці виклику функції проводити не потрібно. Щоб функція зі змінною кількістю параметрів могла сприймати параметри різних типів, необхідно в якості вихідних даних якимось чином передавати їй інформацію про типах параметрів.

Приклад:

#include <stdio.h>

/\* Функція підсумовує значення своїх параметрів \*/

long summa(int m,...) /\*m - число параметрів\*/

{

int \*p=&m; /\* налаштували покажчик на параметр m\*/

long t=0;

for(;m>0;m--) t+=\*++p;

return t;

}

void main()

{

printf("\n summa(2,6,4)=%d",summa(2,6,4));

printf("\n summa(6,1,2,3,4,5,6)=%d", summa(6,1,2,3,4,5,6));

}