Харківський національний університет радіоелектроніки

Кафедра КІТАМ

ЛЕКЦІЯ 6

ПОНЯТТЯ ЦИКЛУ. ОПЕРАТОРИ ЦИКЛУ. ВИКЛИК ФУНКЦІЇ

Ас. каф. КІТАМ Бронніков А.І., 2017 г. ©

Цикли у мові С

Циклом називається блок коду, який для виконання завдання потрібно повторити кілька разів.
Кожен цикл складається з:
блоку перевірки умови повторення циклу;
тіла циклу.
Цикл виконується до тих пір, поки блок перевірки умови повертає істинне значення.
Тіло циклу містить послідовність операцій, яка виконується в разі істинного умови повторення циклу. Після виконання останньої операції тіла циклу знову виконується операція перевірки умови повторення циклу. Якщо ця умова не виконується, то буде виконана операція, що стоїть безпосередньо після циклу в коді програми.
У мові Сі наступні види циклів:
while – цикл з передумовою;
do ... while – цикл з постумовою;
for – параметричний цикл (цикл з заданим числом повторень).

Цикли у мові С

Цикл з передумовою while

Загальна форма запису

```
   while (Умова)
{
   БлокОпераций;
}
```

Якщо Умова виконується (вираз, що перевіряє Умова, не дорівнює нулю), то виконується БлокОпераций, укладений у фігурні дужки, потім Умова перевіряється знову. Послідовність дій, що складається з перевірки Умови та виконання БлокаОпераций, повторюється до тих пір, поки вираз, що перевіряє Умова, не стане хибним (рівним нулю). Пристрій виходить з циклу, і проводиться виконання операції, що стоїть після оператора циклу.

Цикли у мові С

Порахувати суму чисел від 1 до введеного k.

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k; // оголошуємо цілу змінну k
   int i = 1;
   int sum = 0; // початкове значення суми дорівнює 0
   printf ("k =");
   scanf ("% d", & k); // вводимо значення змінної k
   while (i <= k) // поки і менше або дорівнює k
   {      sum = sum + i; // додаємо значення і до суми
     i++; }// збільшуємо і на 1
   printf ("sum =% d \ n", sum); // вивід значення суми
   getchar (); getchar ();
   return 0; }
```

Цикли у мові С

При побудові циклу while, в нього необхідно включити конструкції, що змінюють величину перевіряється вираження так, щоб врешті-решт воно стало хибним (рівним нулю). Інакше виконання циклу буде здійснюватися нескінченно (нескінченний цикл).

```
while (1)
{  БлокОпераций; }
```

while – цикл з передумовою, тому цілком можливо, що тіло циклу не буде виконано жодного разу якщо в момент першої перевірки перевіряється умова виявиться помилковим. Наприклад, якщо в наведеному вище коді програми ввести k = -1, то отримаємо результат, що сума дорівнює 0.

Цикли у мові С

Цикл з постумовою do...while
Загальна форма запису

```
  do {
  БлокОпераций;
} While (Умова);
```

Цикл do ... while – це цикл з умовою поста, де істинність виразу, який перевіряє Умова перевіряється після виконання Блоку Операцій, укладеного у фігурні дужки. Тіло циклу виконується до тих пір, поки вираз, що перевіряє Умова, не стане хибним, тобто тіло циклу з умовою поста виконається хоча б один раз.
Використовувати цикл do ... while краще в тих випадках, коли повинна бути виконана хоча б одна ітерація, або коли ініціалізація об'єктів, що беруть участь в перевірці умови, відбувається всередині тіда циклу.

Цикли у мові С

Перевірити, що користувач ввів число від 0 до 10:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
#include <stdlib.h> // для використання функції system ()
int main () {
   int num; // оголошуємо цілу змінну для числа
   system ("chcp 1251"); // переходимо на російську мову в консолі
   system ("cls"); // очищаємо екран
   do {
     printf ("Введіть число від 0 до 10:"); // запрошення користувачеві
     scanf ("% d", & num); // введення числа
   } While ((num <0) || (num> 10)); // повторюємо цикл поки num <0 або num> 10
   printf ("Ви ввели число% d", num); // виводимо введене значення num - від 0 до 10
   getchar (); getchar ();
   return 0; }
```

Цикли у мові С

Параметричний цикл for
Загальна форма запису:
for (Ініціалізація; Умова; Модифікація)
{ БлокОпераций; }
for – параметричний цикл (цикл з фіксованим числом повторень). Для організації такого циклу необхідно здійснити три операції:
ініціалізація – присвоювання параметру циклу початкового значення;
умова – перевірка умови повторення циклу, найчастіше - порівняння величини параметра з деяким граничним значенням;
модифікація – зміна значення параметра для наступного проходження тіла циклу.
Ці три операції записуються в дужках і розділяються крапкою з комою (;;). Як правило, параметром циклу є цілочисельна змінна.
Ініціалізація параметра здійснюється тільки один раз – коли цикл for починає виконуватися.
Перевірка Умови повторення циклу здійснюється перед кожним можливим виконанням тіла циклу. Коли вираз, що перевіряє Умова стає хибним (рівним нулю), цикл завершується. Модифікація параметра здійснюється в кінці кожного виконання тіла циклу. Параметр може як збільшуватися, так і зменшуватися.

Цикли у мові С

Порахувати суму чисел від 1 до введеного k:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k; // оголошуємо цілу змінну key
   int sum = 0; // початкове значення суми дорівнює 0
   printf ("k =");
   scanf ("% d", & k); // вводимо значення змінної k
   for (int i = 1; i <= k; i ++) // цикл для змінної i від 1 до k з кроком 1
   {      sum = sum + i; } // додаємо значення i до суми
      printf ("sum =% d \ n", sum); // вивід значення суми
   getchar (); getchar ();
   return 0;}
```

Цикли у мові С

У записі циклу for можна опустити одне або кілька виразів, але не можна опускати крапку з комою, що розділяють три складові циклу. Код попереднього прикладу можна представити у вигляді:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k; // оголошуємо цілу змінну key
   int sum = 0; // початкове значення суми дорівнює 0
   printf ("k =");
   scanf ("% d", & k); // вводимо значення змінної k
   int i = 1;
   for (; i <= k; i ++) {// цикл для змінної i від 1 до k з кроком 1
       sum = sum + i;    } // додаємо значення i до суми
   printf ("sum =% d \ n", sum); // вивід значення суми
   getchar (); getchar ();
   return 0; }
```

Цикли у мові С

Параметри, що знаходяться в виразах в заголовку циклу можна змінити при виконанні операції в тілі циклу, наприклад:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k; // оголошуємо цілу змінну key
   int sum = 0; // початкове значення суми дорівнює 0
   printf ("k =");
   scanf ("% d", & k); // вводимо значення змінної k
   for (int i = 1; i <= k;) { // цикл для змінної i від 1 до k з кроком 1
    sum = sum + i; // додаємо значення i до суми
     i ++; }// додаємо 1 до значення i
   printf ("sum =% d \ n", sum); // вивід значення суми
   getchar (); getchar ();
   return 0;}
```

Цикли у мові С

У циклі for може використовуватися операція кома -, - для поділу декількох виразів. Це дозволяє включити в специфікацію циклу кілька ініціюючих або коригувальних виразів. Вирази, до яких застосовується операція кома, будуть обчислюватися зліва направо, наприклад:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k; // оголошуємо цілу змінну key
   printf ("k =");
   scanf ("% d", & k); // вводимо значення змінної k
   for (int i = 1, j = 2; i <= k; i ++, j + = 2) // цикл для змінних
   {// (i від 1 до k з кроком 1) i (j від 2 з кроком 2)
     printf ("i =% d j =% d \ n", i, j); } // виводимо значення i i j
      getchar (); getchar ();
   return 0;}
```

Цикли у мові С

Вкладені цикли

У Сі допускаються вкладені цикли, тобто коли один цикл знаходиться всередині іншого:

```
for (i = 0; i <n; i ++) // зовнішній цикл - Цікл1
{    for (j = 0; j <n; j ++){ // вкладений цикл - Цікл2
       ; // блок операцій Цікла2
  }    // блок операцій циклу1; }
```

Наприклад вивести числа від 0 до 99, по 10 в кожному рядку:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
  for (int i = 0; i <10; i ++) // цикл для десятків
  {      for (int j = 0; j <10; j ++){ // цикл для одиниць
          printf ("% 2d", i * 10 + j); } // виводимо обчислене число (2 знакоместа) і пробіл
        printf ("\ n");} // в зовнішньому циклі переводимо рядок
     getchar (); // scanf () не використовувався
  return 0; }  // тому консоль можна утримати одним викликом getchar().
```

Цикли у мові С

При виборі циклу необхідно оцінити необхідність перевірки умови при вході в цикл або по завершенні проходження циклу. Цикл з умовою поста зручно застосовувати у випадках, коли для перевірки умови потрібно обчислити значення виразу, яке потім буде розміщено в тілі циклу (див. Вище приклад введення числа від 0 до 10).

Цикл с передумовою використовується в разі якщо всі змінні, які беруть участь у вираженні, перевіряє умови, проініціалізовані заздалегідь, але точне число повторень циклу невідомо чи передбачається складна модифікація змінних, що беруть участь у формуванні умови повторення циклу.

Якщо цикл орієнтований на роботу з параметром, для якого заздалегідь відомо кількість повторень і крок зміни, то більш привабливим є параметричний цикл. Дуже зручно використовувати параметричний цикл при роботі з масивами для перебору елементів.

Цикли у мові С

Оператори переривання і продовження циклу break і continue
У тілі будь-якого циклу можна використовувати оператори переривання циклу – break і продовження циклу – continue.
Оператор break дозволяє вийти з циклу, але не завершити його.
Оператор continue дозволяє пропустити частину операторів тіла циклу і почати нову ітерацію.

Цикли у мові С

Вивести числа від 0 до 99 нижче головної діагоналі:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   for (int i = 0; i <10; i ++) {// цикл для десятків
        for (int j = 0; j <10; j ++){ // цикл для одиниць
            if (j> i) // якщо число одиниць більше числа десятків в числі
          break; // виходимо з вкладеного циклу і переходимо до нового рядка
       printf ("% 2d", i * 10 + j); } // виводимо обчислене число (2 знакомісця) і пробіл
          printf ("\ n");} // в зовнішньому циклі переводимо рядок
      getchar (); // scanf () не використовувався
   return 0; } // тому консоль можна утримати одним викликом getchar().
```

Цикли у мові С

Вивести числа від 0 до 99 виключаючи числа, що закінчуються на 5 або 8:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   for (int i = 0; i <10; i ++) // цикл для десятків
   {      for (int j = 0; j <10; j ++) // цикл для одиниць
     {   if ((j == 5) || (j == 8)) // якщо число одиниць в числі одно 5 або 8,
         continue; // переходимо до наступної ітерації циклу
       printf ("% 2d", i * 10 + j); // виводимо обчислене число (2 знакоместа) і пробіл
     }      printf ("\ n"); // в зовнішньому циклі переводимо рядок
   }    getchar (); // scanf () не використовувався
   return 0; } // тому консоль можна утримати одним викликом getchar().
```

Цикли у мові С

Оператор безумовного переходу goto
Загальна форма запису:
   goto мітка;
. . .
Мітка: Операція;
Виконання оператора goto викликає передачу управління в програмі операції з позначкою Міткою. По суті Метка є ідентифікатором адреси операції, якій має бути передано управління. Для відділення Мітки від Операції використовується двокрапка.
Мітка може розташовуватися в програмі як до оператора goto, так і після нього. Імена Меток утворюються за тими ж правилами, що й імена змінних.

Цикли у мові С

Вивести всі цілі числа від 5 до 0:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int main () {
   int k = 5;
M1: if (k <0) // якщо k <0,
     goto M2; // переходимо на мітку M2 (виходимо з програми)
   printf ("% d", k); // виводимо значення k
   k--; // зменшуємо k на 1
   goto M1; // переходимо на мітку M1 (повторюємо операції вище)
M2: getchar ();
   return 0; }
```

Функції та їх виклик

Функція – це самостійна одиниця програми, яка спроектована для реалізації конкретної підзадачі. Функція є підпрограмою, яка може міститися в основній програмі, а може бути створена окремо (в бібліотеці). Кожна функція виконує в програмі певні дії.

Сигнатура функції визначає правила використання функції. Зазвичай сигнатура є опис функції, що включає ім'я функції, перелік формальних параметрів з їх типами і тип значення, що повертається.

Семантика функції визначає спосіб реалізації функції. Зазвичай являє собою тіло функції.

Функції та їх виклик

Кожна функція в мові Сі повинна бути визначена, тобто повинні бути вказані:
тип значення;
ім'я функції;
інформація про формальні аргументах;
тіло функції.
Визначення функції має наступний синтаксис:

```
  Тип значення, що повертається ІмяФункції
(СписокФормальнихАргументів)
{     Тіло Функції;
   ...
   return (ЗначенняЩоПовертається); }
```

Наприклад, функція складання двох чисел:

```
float function(float x, float z)
{  float y;
  y=x+z;
  return(y); }.
```

Функції та їх виклик

У зазначеному прикладі повертається значення має тип float. Як повертається в зухвалу функцію передається значення змінної y. Формальними аргументами є значення змінних x і z. Якщо функція не повертає значення, то тип значення для неї вказується як void. При цьому операція return може бути опущена. Якщо функція не приймає аргументів, в круглих дужках також вказується void.
Розрізняють системні (в складі систем програмування) і власні функції.
Системні функції зберігаються в стандартних бібліотеках, і користувачеві не потрібно вдаватися в подробиці їх реалізації. Досить знати лише їх сигнатуру. Прикладом системних функцій, використовуваних раніше, є функції printf () і scanf ().

Функції та їх виклик

Власні функції – це функції, написані користувачем для вирішення конкретної підзадачі.
Розбиття програм на функції дає наступні переваги:
Функцію можна викликати з різних місць програми, що дозволяє уникнути повторення програмного коду.
Одну і ту ж функцію можна використовувати в різних програмах.
Функції підвищують рівень модульності програми і полегшують її проектування.
Використання функцій полегшує читання і розуміння програми і прискорює пошук і виправлення помилок.
З точки зору викликає програми функцію можна представити як якийсь "чорний ящик", у якого є кілька входів і один вихід. З точки зору викликає програми неважливо, яким чином проводиться обробка інформації всередині функції. Для коректного використання функції досить знати лише її сигнатуру.

Функції та їх виклик

Загальний вигляд виклику функції
Змінна = ІмяФункції (СписокФактичнихАргументів);
Фактичний аргумент – це величина, яка присвоюється формальному аргументу при виконанні функції. Таким чином, формальний аргумент – це змінна в викликається функції, а фактичний аргумент – це конкретне значення, присвоєне цієї змінної викликає функцією. Фактичний аргумент може бути константою, змінною або виразом. Якщо фактичний аргумент представлений у вигляді виразу, то його значення спочатку обчислюється, а потім передається в функцію, що викликається. Якщо в функцію потрібно передати кілька значень, то вони записуються через кому. При цьому формальні параметри замінюються значеннями фактичних параметрів в порядку їх слідування в сигнатурі функції.

Функції та їх виклик

Повернення у попередню функцію

Після закінчення виконання викликається функції здійснюється повернення значення в точку її виклику. Це значення присвоюється змінної, тип якої повинен відповідати типу значення, що повертається функції. Функція може передати в зухвалу програму тільки одне значення. Для передачі значення, що повертається в зухвалу функцію використовується оператор return в одній з форм:

```
return (ВозвращаемоеЗначеніе);
return ВозвращаемоеЗначеніе;
```

Дія оператора наступне: значення виразу, укладеного в дужки, обчислюється і передається в зухвалу функцію. Значення, що повертається може використовуватися в зухвалій програмі як частина деякого виразу.

Оператор return також завершує виконання функції і передає управління наступного оператора в викликає функції. Оператор return не обов'язково повинен знаходитися в кінці тіла функції.

Функції та їх виклик

Функції можуть і не повертати значення, а просто виконувати деякі обчислення. У цьому випадку вказується порожній тип значення void, а оператор return може або відсутні, або не повертати жодного значення. Наприклад, розрахувати суму двох чисел:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
// Функція обчислення суми двох чисел
int sum (int x, int y) // в функцію передаються два цілих числа
{    int k = x + y; // обчислюємо суму чисел і зберігаємо в k
   return k; } // повертаємо значення k
int main ()
{    int a, r; // опис двох цілих змінних
   printf ("a =");
   scanf ("% d", & a); // вводимо a
   r = sum (a, 5); // виклик функції: x = a, y = 5
   printf ("% d + 5 =% d", a, r); // висновок: a + 5 = r
   getchar (); getchar (); // ми використовували scanf (),
   return 0; } // тому getchar () викликаємо два рази.
```

Функції та їх виклик

Прототип необхідний для того, щоб компілятор міг здійснити перевірку відповідності типів переданих фактичних аргументів типам формальних аргументів. Імена формальних аргументів на прототипі функції можуть бути відсутні.

Якщо в прикладі вище тіло функції додавання чисел розмістити після тіла функції main, то код буде виглядати наступним чином:

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int sum (int, int); // сигнатура
int main ()
{    int a, r;
   printf ("a =");
   scanf ("% d", & a);
   r = sum (a, 5); // виклик функції: x = a, y = 5
   printf ("% d + 5 =% d", a, r);
   getchar (); getchar ();
   return 0; }
int sum (int x, int y) // семантика
{    int k;
   k = x + y;
   return (k); }.
```

Функції та їх виклик

Рекурсивні функції

Функція, яка викликає сама себе, називається рекурсивної функцією.

Рекурсія – виклик функції з самої функції.

Приклад рекурсивної функції – функція обчислення факторіала.

```
#define _CRT_SECURE_NO_WARNINGS // для можливості використання scanf
#include <stdio.h>
int fact (int num) // обчислення факторіала числа num
{    if (num <= 1) return 1; // якщо число не більше 1, повертаємо 1
   else return num * fact (num - 1); } // рекурсивний виклик для числа на 1 менше
// Головна функція
int main ()
{    int a, r;
   printf ("a =");
   scanf ("% d", & a);
   r = fact (a); // виклик функції: num = a
   printf ("% d! =% d", a, r);
   getchar (); getchar ();
   return 0; }.
```

функції та їх виклик

Функція	Опис
int abs(int x)	Модуль цілого числа x
double acos(double x)	Арккосинус x
double asin(double x)	Арксинус x
double atan(double x)	Арктангенс x
double cos(double x)	Косинус x
double cosh(double x)	Косинус гіперболічний x
double exp(double x)	Експонента x
double fabs(double x)	Модуль речового числа
double fmod(double x, double y)	Остаток от ділення x/y
double log(double x)	Натуральний логарифм x
double log10(double x)	Десятковий логарифм x
double pow(double x, double y)	x у ступені y
double sin(double x)	Синус x
double sinh(double x)	Синус гіперболічний x
double sqrt(double x)	Квадратний корінь x
double tan(double x)	Тангенс x
double tanh(double x)	Тангенс гіперболічний x