ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Урок №9

Програмиране на език C

Съдържание

1. Вграждане

Ключов израз inline.

В предишния урок се запознахме с понятието функции. И изяснихме, че веднага щом програмата се сблъска с извикване на функция, тя веднага се обръща към тялото на дадената функция и го изпълнява. Този процес съществено съкращава кода на програмата, но за сметка на това, увеличава времето за изпълнение за сметка на постоянните обръщения към описанието на конкретната извикана функция. Но, не винаги е така. Някои функции в езика С може да се определят с използването на специалната служебна дума inline.

Дадения спецификатор позволява да се определи функцията като вграждаема, тоест, да може да се поставя в текста на програмата в местата къде се обръща към тази функция. Например, следващата функция е определена като подставена:

```
inline float module(float x = 0, float y = 0)
{
        return sqrt(x * x + y * y);
}
```

Обработвайки всяко извикване на вградената функция module, компилаторът поставя на нейно място – в текста на програмата – кода на операторите на тялото на функцията. По този начин при многократно извикване на поставената функция, размерът на програмата може да се увеличи. Но се спестява време за търсене на извикваната функция и връщане от нея в основната функция на програмата.

Забележка: Най-ефективно е да се използват подставани функции в тези случаи, когато тялото на функцията

се състои само от няколко оператора.

Случва се така че компилатора не може да определи функцията като вграждаема и просто игнорира ключовата дума inline. Да изредим причините, които водят до такъв резултат:

1. Прекалено голям размер на функцията.
2. Функцията е рекурсивна. (с това понятие ще се запознаете в следващите уроци)
3. Функцията се повтаря в един и същи израз няколко пъти
4. Функцията съдържа цикъл, switch или if.

Както виждате всичко е много просто, но функцията inline не е единствения начин за вграждане. За това ще разкаже следващата тема от урока.

Разкриване на макро.

Освен търсене на функция за вграждане в програмата на повтарящ се фрагмент се използва тъй нареченото отваряне на макрос. За целта се използва директива на препроцесора #define със следния синтаксис:

```
#define Име_на_макрото(Параметри) (Израз)
```

```
#include <iostream>

#define SQR(X) ((X) * (X))
#define CUBE(X) (SQR(X)*(X))
#define ABS(X) (((X) < 0)? -(X) : X)

using namespace
std; void main()
{
       y = SQR(t + 8) - CUBE(t - 8)
         ; cout <<sqrt(ABS(y)) ;
}
```

1. С помощта на директивата #define се обявяват три макрота sqr(x), cube(x) и abs(x).
2. Във функцията main протича извикване на гореописаните макроси по име.
3. Предпроцесора разкрива макрото (тоест поставя на мястото на повикване израз от директивата #define) и предава получилият се текст на компилатора.
4. След вграждането, израза в main изглежда за програмата по следния начин:

```
y = ((t+8) * (t+8)) - ((((t-8)) * (t-8)) *
(t-8)); cout << sqrt(((y < 0)? -(y) : y));
```

Забележка: Трябва да се обърне внимание на използването на скоби при обявяването на макро. С помощта им избягваме грешки в последователността на изчисленията. Например:

```
#define SQR(X) X * X
y = SQR(t + 8); //разкрива макрото t+8*t+8
```

В примера при извикване на макро SQR първо се изпълнява умножение 8 на t, а след това към резултата се прибавя значението на променливата t и осмица, въпреки че, нашата цел беше да получим квадрата на сумата t+8.

Сега сте напълно запознати с понятието вграждане и можете да го използвате в своите програми.

2. Презареждане на функциите

Всеки път, когато изучаваме нова тема, на нас е важно да разберем, къде може да приложим нашите знания на практика. Целта на презареждането на функциите е в това, че няколко функции обладаващи едно име, да се изпълват по различен начин и да възвръщат различни значения при обръщение към тях различни по тип и количество фактически параметри.

Например, може да потрябва функция, възвръщаща максималното от значенията на елементите на едномерния масив, предавайки го в качеството на параметър. Масивите, използвани като фактически параметри, могат да съдържат различни типове елементи, но потребителят на функцията не трябва да се безпокои за типа на резултата. Функцията винаги трябва да възвръща значението на същия тип, какъвто е типът на масива - фактически параметър.

За реализация на презареждане на функцията е необходимо за всяко име да се определи колко различни функции са свързани с него, тоест колко варианта на извикване са допустими при обръщение към него. Да предположим, че функцията за избор на максимално значение на елемент от масива е длъжна да работи за масивите от типа int, long, float, double. В този случай ще се наложи да се напишат 4 различни варианта на функция с едно и също име. В нашия пример тази задача е решена по следния начин :

```
#include <iostream>

using namespace std;

long max_element(int n, int array[])
// Функция за масиви с елементи от типа int.
{
       int value = array[0];
       for (int i = 1; i < n; i++)
              value = value > array [i] ? value :
        array [i] ; cout << "\nFor (int) : ";
        return long(value);
}
long max_element(int n, long array[])
// Функция за масиви с елементи от типа long.
{
       long value = array[0];
       for (int i = 1; i < n; i++)
              value = value > array[i] ? value :
        array[i]; cout << "\nFor (long) : ";
        return value;
}
double max_element(int n, float array[])
// Функция за масиви с елементи от типа float.
{
       float value = array[0];
       for (int i = 1; i < n; i++)
              value = value > array[i] ? value :
        array[i]; cout << "\nFor (float) : ";
        return double(value);
}
double max_element(int n, double array[])
// Функция за масиви с елементи от типа double.
{
       double value = array[0];
       for (int i = 1; i < n; i++)
              value = value > array[i] ? value :
        array[i]; cout << "\nFor (double) : ";
        return value;
}

void main()
{
       int x[] = { 10, 20, 30, 40, 50, 60 };
       long f[] = { 12L, 44L, 5L, 22L, 37L, 30L };
       float y[] = { 0.1, 0.2, 0.3, 0.4, 0.5, 0.6 };
       double z[] = { 0.01, 0.02, 0.03, 0.04, 0.05, 0.06 };
       cout << "max_elem(6,x) = " << max_element(6,x); cout
       << "max_elem(6,f) = " << max_element(6,f); cout <<
       "max_elem(6,y) = " << max_element(6,y); cout << "max
       elem(6,z) = " << max_element(6,z);
}
```

1. В програмата показахме независимостта на претоварените функции от типа възвращаемо значение. Две функции, обработващи цели масиви (int, long), връщат значение на един тип long. Две функции, обработващи вещественi масиви (double, float), и двете връщат значение от типа double.

2. Разпознаване на претоварени функции при извикване, се изпълнява по техните параметри. Претоварените функции трябва да имат еднакви имена, но спецификациите на техните параметри трябва да се различават по количество и (или) по типа, и (или) по разположение.

ВНИМАНИЕ!!!

```
double multy (double x) {
        return x * x * x;
}
double multy (double x, double
        y) { return x * y * y;
}
double multy (double x, double y, double z)
{
        return x * y * z;
}
```

Всяко от следващите обръщения към функцията multy() ще бъде еднозначно идентифицирано и правилно обработено:

```
multy (0.4)
     multy (4.0, 12.3)

     multy (0.1, 1.2, 6.4)
```

Обаче, добавяне в програмата функции с начални значения на параметрите:

```
double multy (double a = 1.0, double b  = 1.0, double c = 1.0, double d = 1.0)
{
         return a * b + c * d;
}
```

Завинаги ще объркa всеки компилатор при опит за обработване, например, такова извикване, което ще доведе до грешка на етапа за компилиране. Бъдете внимателни!!!

```
multy(0.1, 1.2);
```

Следващия раздел на урока ще разкаже за алтернативните решения, позволяващи да се създаде универсална функция.

3. Шаблони на функции

Шаблоните във функции в език С позволяват да се създаде общо опреде.

В предишната тема, за да се използва една и съща функция с различни типове данни ние създавахме отделна претоварена версия на тази функция за всеки тип. Например:

```
int Abs(int N){
  return N < 0 ? -N : N;
}
double Abs(double N){
  return N < 0. ? -N : N;
}
```

Сега, използвайки шаблона ние може :

```
template <typename T> T Abs (T N)
{
  return N < 0 ? -N : N;
}
```

Сега да обсъдим това което ни се е получило.

1. Идентификатор **T** е **параметър на типа.** Именно той определя типа на параметъра, предаван в момента на извикване на функцията.
2. Да предположим, че програмата извиква функцията Abs и ще й предаде значение от типа int:

```
cout << "Result - 5 = " << Abs(-5);
```

3. В дадения случай, компилатора автоматично създава версия на функцията, където вместо T се поставя тип *int*.

4. Сега функцията ще изглежда така:

```
int Abs (int N)
{
  return N < 0 ? -N : N;
}
```

5. Трябва да се отбележи, че компилатора създава версии на функциите за всеки вид извикване с различен тип данни. Такъв процес се нарича създаване на екземпляр на шаблона на функция.

Основни принципи и понятия при работа с шаблони.

Сега, след повърхностно запознаване ще разгледаме всичките особености при работа с шаблоните:

1.При определение на шаблона се използват два спецификатора: **template** и **typename**.

2.На мястото на параметъра от типа T може да се постави каквото и да е коректно име.

3.В квадратните скоби може да се запише повече от един параметър на типа.

4.Параметър на функцията е значение предавано във функцията при изпълнение на програмата.

5. Параметърът на типа сочи типа аргумент присвояван на функцията и се обработва само при компилация.

Процес на компилация на шаблона.

1.Определение на шаблона не предизвиква генерация на кода от компилатора самостоятелно. Последния създава код на функция само в момента на нейното извикване и генерира при това съответстващата версия на функцията.

2.Следващото извикване със същите типове данни на параметрите няма да провокира генерация на допълнително копие на функцията, а ще извика вече съществуващо копие.

3.Компилатора създава нова версия на функцията, само ако типа на предадения параметър не съвпада с нито един от предишните.

Пример за работа с шаблон:

```
template <typename T> T Max (T A, T B)
{
  return A > B ? A : B;
}
```

1. Шаблона генерира множество функции, връщащи по-голямото от две значения с еднакъв тип данни.

2. И двата параметъра са определени като параметри от типа T и при извикване на функцията, те задължително трябва да са от един тип. В дадения случай, са възможни такива повиквания на функция:

```
cout << "Большее из 10 и 5 = " << Max(10, 5) << "\n";
cout << "Большее из 'A' и 'B' = " << Max('A', 'B') << "\n";

cout << "Большее из 3.5 и 5.1 = " << Max(3.5, 5.1) << "\n";
```

А такова повикване ще доведе до грешка:

```
cout << "Большее из 10 и 5.55 = " << Max(10, 5.55); // ОШИБКА!
```

Компилатора няма да може да преобразува параметъра int в double. Решението на проблема за предаване на различни параметри е следния шаблон:

```
template <typename T1, typename T2> T2 Max(T1 A , T2 B)
{
   return A > B ? A : B;
}
```

В този случай T1 означава тип значене, предавано в качеството на първи параметър, а T2 - на втори.

ВНИМАНИЕ!!!

Всеки тип параметър, включен в ъглови скоби е длъжен задължително да се появи в списъка параметри на функцията. В противен случай ще произлезе грешка на етапа за компилация..

```
template <typename T1, typename T2> T1 Max(T1 A , T1 B)
{
  return A > B ? A : B;
}
    // ОШИБКА! списъка параметри трябва да включва T2 като тип параметър
```

Преопределяне на шаблоните на функция

1. Всяка версия на функция, генерирана с помощта на шаблон, съдържа един и същи фрагмент от кода.
2. Обаче за отделни тип параметри може да се осигури специална реализация на кода, тоест да се определи обикновена функция със същото име както и шаблона.
3. Обикновената функция ще преопредели шаблона. Ако компилатора намира типа предадени параметри съответстващи на спецификациите на обикновена функция, той ще я извика и няма да създаде функция по шаблон.

4. Домашна работа

1.Да се напише шаблон на функция за търсене на средно аритметичното значение на масива.

2.Написать претоварени шаблони на функция за намиране на корен от линейното (a*x + b = 0) и квадратното (a*x2+b*x + c = 0) уравнение. Забележка: във функцията се предават коефициентите на уравнението.

3.Да се напише функция, която приема в качеството на параметър вещественo число и количеството знаци после десетичната точка, които трябва да останат. Задачата на функцията е закръгляне на гореуказаното вещественo число с зададена точност.