ШАГ компьютерная академия

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ С

Dərs №9

C dilində proqramlaşdırma

Mündəricat

1. Daxiletmə 3
2. Funksiyaya yükləmə. 6
3. Şablon funksiyalar 10
4. Ev tapşırıqları 14

1. Daxiledilmə

inline açar sözü

Keçən dərsimizdə biz funksiya anlayışı ilə tanış olduq. Və aydınlaşdırdıq ki, proqram funksiyanın çağırılmasına rast gəldiyində, o, o anda funksiyanın gövdəsinə müraciət edir və onu icra edir. Bu proses proqram kodunu qisaldır, lakin hansısa təyin olunmuş funksiyaya tez-tez müraciət etmə nəticəsində onun icrası uzanır. Lakin bəzən belə olmaya da bilər. C dilində bəzi funksiyaları xüsusi inline açar sözü vasitəsilə də təyin etmək olar.

Bu spesifikator funksiyanı daxiledilmiş, yəni proqramın mətnində bu funksiyaya müraciət edilən yerlərdə yerləşdirilmiş kimi təyin etməyə imkan verir. Məsələn, növbəti funksiya daxiledilmiş kimi təyin edilmişdir:

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

Daxiledilmiş module funksiyanın hər bir çağırılmasında kompilyator onun çağırıldığı yerə - proqramın mətninə - funksiya gövdəsinin əmrlər kodunu yerləşdirir. Həmçinin, yerləşdirilən funksiyanın çoxsaylı çağırılması zamanı proqramın ölçüsü arta bilər, lakin çağırılan funksiyaya müraciətə və proqramın əsas funksiyasına (main) qaytarılmağa sərf olunan zaman istisna edilir.

Qeyd: Daxiledilmiş funksiyadan, əsasən onun bir neçə əmrdən ibarət olduğu zaman istifadə etmək daha səmərəlidir.

Belə hal olur ki, kompilyator vunksiyanı daxiledilmiş kimi təyin edə bilmir və inline açar sözünü nəzərə almır. Növbəti hallar buna səbəb ola bilər:

1. Funksiyanın ölçüsü çox böyükdür.
2. Funksiya rekursivdir (bu anlayışla növbəti dərslərdə tanış olacaqsınız)
3. Funksiya eyni ifadədə bir neçə dəfə təkrarlanır.
4. Funksiya dövr, switch və ya if əmrlərini ehtiva edir.

Gördüyünüz kimi – hər şey sadədir, lakin inline-funksiyalar yeganə daxiledtmə üsulu deyildir. Bunun haqqında dərsin növbəti mövzusunda bəhs edilir.

Genişlənmə makrosu.

Təkrarlanan fraqmentin proqrama daxil edilməsi üçün funksiyanın çağırılmasından başqa genişlənmə makroslarından da istifadə edilir. Bu məqsədlə #define prosessor direktivindən istifadə edilir. Sintaksisi növbəti şəkildədir:

```
#define Makros_adı(Parametrlər) (İfadə)


#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. #define direktivi vasitəsilə üç makros: SQR(x), CUBE(x) və ABS(x) təyin edilir.
2. main funksiyasının daxilindən yuxarıda verilmiş makroslara adları ilə müraciət edilir.
3. Prosessor makrosu açır (yəni, çağırılma yerinə #define diriektivindəki ifadəni verir) və alınmış mətni kompilyatora ötürür.
4. İfadə main funksiyasına yerləşdirildikdən sonra proqram üçün növbəti şəkildə görünür:

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

Qeyd: Makros elan edilən zaman mötərizələrin istifadəsinə diqqət etmək lazımdır. Onların vasitəsilə biz hesablamanın ardıcıllığında səhvlərin olması ehtimalını azaldırıq. Məsələn:

```
#define SQR(X) X * X
y = SQR(t + 8); // t+8*t+8 makrosunu açır
```

Misalda, SQR makrosu çağırılan zaman əvvəlcə 8 t-yə vurulur, sonra isə nəticə ilə t və 8 cəmlənir, aydındır ki, bizim məqsədimiz isə t+8 cəminin kvadratını hesablamaqdır.

Artıq siz daxiletmə anlayışı ilə tam tanışsınız və ondan öz proqramınızda istifadə edə bilərsiniz.

2. Funksiyaya yükləmə

Biz hər dəfə yeni mövzu öyrənərkən, bu biliklərin təcrübədə necə istifadə ediləcəyini bilməliyik. Funksiyaya yükləməkdə məqsəd ondan ibarətdir ki, eyni adlı funksiyalar onlara müxtəlif tiplərlə və müxtəlif sayda faktik parametrlərlə müraciət edərkən müxtəlif cür icra edilsinlər və müxtəlif qiymətlər qaytarsınlar.

Məsələn, parametr kimi ötürülmüş birölçülü massivin elementlərinin ən böyük qiymətini qaytaran funksiya tərtib etmək tələb olunur. Faktik parametr kimi istifadə olunan massiv müxtəlif tipli elementlər ehtiva edə bilər, lakin funksiyanı istifadə edən istifadəçi bundan narahat olmamalıdır. Funksiya faktik parametr kimi istifadə olunan massivin tipinə uyğun qiymət qaytarmalıdır.

Funksiyaya yükləmək üçün hər bir ad üçün neçə funksiyanın onunla əlaqəli olduğunu, yəni bu funksiyaya müraciət zamanı neçə çağırış variantının olduğunu təyin etmək lazımdır. Hesab edək ki, massivin ən böyük qiymətini təyin etməyə imkan verən funksiya int, long, float, double tipli massivlər üçün işləməlidir. Bu halda dörd müxtəlif variantda eyni adlı funksiyalar yazmaq lazım gələcək. Bizim misalda bu məsələ növbəti şəkildə həll edilmişdir:

```
#include <iostream>
using namespace std;

long max_element(int n, int array[])
//Elementləri int tipində olan massiv üçün funksiya.
{
	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[])
//Elementləri long tipində olan massiv üçün funksiya.
{
	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[])
//Elementləri float tipində olan massiv üçün funksiya.
{
	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[])
//Elementləri double tipində olan massiv üçün funksiya.
{
	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. Proqramda biz yüklənmiş funksiyaların qaytarılan qiymətin tipindən asılı olmadığını nümayiş etdirdik. Tam qiymətli (int, long) massivlər emal edən iki funksiya eyni long tipində qiymət qaytarır. Həqiqi qiymətli (float, double) massivlər emal edən iki funksiya eyni double tipində qiymət qaytarır.

2. Yüklənmiş funksiyaların çağırılması zamanı tanınması onların parametrlərinə əsasən yerinə yetirilir. Yüklənmiş funksiyalar eyni adda olmalıdırlar, lakin onların parametrləri sayına və (və ya) tipinə və (və ya) sırasına görə fərqlənməlidirlər.

DİQQƏT!!!

Yüklənmiş funksiyalardan istifadə zamanı onların parametrlərinin başlanğıc qiymətlərini təyin edərkən diqqətli olmaq lazımdır. Hesab edək ki, biz müxtəlif sayda parametrlərin hasilini hesablayan yüklənmiş funksiya təyin etmişik:

```
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;
}
```

multy() funksiyasına hər dəfə müraciət zamanı birmənalı şəkildə tanınacaq və düzgün emal ediləcək:

```
multy (0.4)
    multy (4.0, 12.3)
    multy (0.1, 1.2, 6.4)
```

Lakin proqrama parametrlərin başlanğıc qiymətləri olan funksiya əlavə edərkən:

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

İxtiyari kompilyator emala cəhd edərkən bunu qarışdırır, məsələn, belə çağırılma kompilyasiya mərhələsində səhvə yol açacaq. Diqqətli olun!!!

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

Dərsin növbəti bölməsində universal funksiya tərtib etməyə imkan verən alternativ həldən bəhs ediləcək.

3. Funksiya şablonları

C dilində funksiya şablonları müxtəlif tiplər üçün tətbiq olunan funksiyanın ümumi təyin edilməsinə imkan verir.

Əvvəlki mövzuda müxtəlif verilənlər tipi ehtiva edən eyni adlı funksiya istifadə etmək üçün biz hər bir tip üçün bu funksiyanın müxtəlif yükləmə versiyalarını tərtib etdik. Məsələn:

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

İndi isə biz şablondan istifadə edərək müxtəlif tipləri emal edən yeganə təyini reallaşdıra bilərik:

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

İndi bizdə nə alındığını müzakirə edək.

1. **T** adı **tip parametridir.** Məhz o funksiyanın çağırılması zamanı ötürülən parametrin tipini təyin edir.

2. Hesab edək ki, proqram Abs funksiyasını çağırır və onun qiymətini int tipində qaytarır:

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

3. Bu halda kompilyator avtomatik olaraq **T** yerinə *int* tipini qoyaraq funksiyanın versiyasını yaradır.

4. İndi funksiya növbəti şəkildə olacaq:

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

5. Qeyd etmək lazımdır ki, kompilyator istənilən çağırış və istənilən tip üçün funksiyanın versiyasını yaradır. Bu proses **funksiya şablonunun nümunəsinin yaradılması** adlanır.

Şablonla iş zamanı əsas prinsip və anlayışlar.

İlkin tanışlıqdan sonra biz şablonlarla işləməyin bütün xüsusiyyətlərinə baxaq:

1. Şablon təyin edildiyi zaman iki spesifikatordan istifadə edilir: **template** və **typename**.
2. T parametr tipinin yerinə ixtiyari düz ad qoymaq olar.
3. Günc mötərizələrinin daxilində bir neçə tip parametri yazmaq olar.
4. Funksiyanın parametri – proqramın icrası zamanı funksiyaya ötürülən qiymətdir.
5. Tip parametri – funksiyaya ötürülən arqumentin tipini göstərir və yalnız kompilyasiya zamanı hasil edilir.

Şablonun kompilyasiya prosesi.

1. Şablonun təyin edilməsi kodun kompilyator tərəfindən hasil edilməsini çağırmır. Sonuncu funksiya kodunu yalnız onun çağırılması zamanı yaradır və bu zaman funksiyanın uyğun versiyasını hasil edir.

2. Həmin verilənlər tipi parametrləri ilə növbəti çağırılma funksiyanın əlavə nüsxəsinin hasil edilməsinə səbəb olmur, əksinə onun artıq mövcud olan nüsxəsini çağırır.

3. Əgər ötürülən parametrin tipi əvvəlki çağırışların heç birinə uyğun gəlmirsə, kompilyator funksiyanın yeni versiyasını yaradır.

Şablonla işə aid nümunə:

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

1. Şablon eyni tipli iki qiymətdən ən kiçiyini qaytaran funksiyalar çoxluğunu hasil edir.

2. Hər iki parametr T tipli parametrlər kimi təyin edilir və mütləq eyni tipdə olmalıdırlar. Bu halda funksiyaların növbəti çağırılması mümükündür:

```
cout << "10 və 5-in ən böyüyü = " << Max(10, 5) << "\n";
cout << "'A' bə 'B'-nin ən böyüyü = " << Max('A', 'B') << "\n";

cout << "3.5 və 5.1-in ən böyüyü = " << Max(3.5, 5.1) << "\n";
```

Növbəti verilmiş çağırış forması isə səhvə yol açacaq:

```
cout << "10 və 5.55-in ən böyüyü = " << Max(10, 5.55); // SƏHV!
```

Kompilyator int parametrini double parametrinə çevirə bilməz. Müxtəlif parametrlərin ötürülməsi problemini növbəti şəkildə həll etmək olar:

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

Bu halda T1 birinci, T2 isə ikinci parametr kimi ötürülən qiyməti təyin edir.

DİQQƏT!!!

Künc mötərizələri daxilində verilmiş hər bir tip parametri funksiyanın parametrlər siyahısında MÜTLƏQ olmalıdır. Əks halda kompilyasiya mərhələsində səhv baş verəcək.

```
template <typename T1, typename T2> T1 Max(T1 A , T1 B)
{
  return A > B ? A : B;
}
    // SƏHV! Parametrlər siyahısı tip parametri kimi T2 ehtiva etməlidir.
```

Şablon funksiyalarının öncədən təyin edilməsi

1. Funksiyanın hər bir versiyasının şablon vasitəsi ilə hasil edilməsi eyni kod fraqmentini ehtiva edir.
2. Lakin, hər bir tip parametri üçün kodun xüsusi reallaşdırmasını, yəni şablon adı ilə eyni olan adi funksiyanın təyin edilməsini təmin etmək olar.
3. Adi funksiya şablonu öncədən təyin edir. Əgər kompilyator adi funksiyanın xüsusiyyətinə uyğun ötürülən parametrlərin tipini tapırsa, onda onu çağırır və şablona görə funksiya yaratmır.

4. Ev tapşırıqları

1. Massivin elementlərinin ədədi ortasını hesablayan şablon funksiya tərtib edin.
2. (a*x + b = 0) xətti və (a*x2+b*x + c = 0) kvadrat tənliklərin kökünü hesablayan yüklənmiş şablon funksiyalar tərtib edin. Qeyd: funksiyaya tənliyin əmsalları ötürülür.
3. Parametrlər kimi həqiqi ədəd və onluq nöqtədən sonra işarələrin sayını qəbul edən funksiya tərtib edin. Funksiyanın işi verilmiş həqiqi ədədi, verilmiş dəqiqlikdə yuvarlaqlaşdırmaqdan ibarətdir.