Предпроцесорни директиви

Дефиниране на константи и макроси

В C++ съществуват дава начина за деклариране на константи. Посредством декларацията `cost` или посредством предпроцесорната директива `#define`. Докато при деклариране на константата посредством `const`, се заделя място за нея в работната област на програмата, то при използването на `#define` механизъма е по-различен. Ако дадена константа е дефинирана посредсвом `#define` в процеса на компилация предпроцесора проверява целият сорс за наличието на идентификатора на константата и го заменя с действителната му стойност.

Пример:

```
#define PI  3.14
void main()
{
  ….
   S = 2*R*PI;
  ….
}
```

Директивата `#define` се използва също и за дефиниране на макроси. Тук можем да си зададем въпроса "Какво представлява макроса?". **Макроса представлява обособена част от програмата, която може да се повтаря многократно.** По своята същност макроса наподобява подпрограмите но има и някой различия:

- Кодът на макроса и на подпрограмите се намира на едно място в програмния код.
- Макроса и подпрограмата могат да се извикат от произволно място в програмата посредством името си.
- На макроса и на подпрограмата могат да се предават параметри.
- Компилирания код на подпрограмата се намира само на едно място, докато този на макроса се съдържа на всякъде където се извиква
- При използването на подпрограми е необходимо по-голямо количество оперативна памет, отколкото при макросите (при извикване на подпрограма в стека се записват копия на параметрите които се предават и на адреса на връщане).
- Макросите се изпълняват по-бързо. Дължи се на обстоятелството, че не се осъществява обръщение към стека за запис на параметрите и адреса на връщане.

Как се дефинират макроси в C++? Синтаксисът на една макро дефиниция е следният:

#define <име> <код_на_макроса>

Пример:

```
#include <conio.h>
#include <stdio.h>
#define KEY while( getch() != '\n')
int main(int argc, char** argv){
  int I;
  printf("\n Input nummber: ");
  I = getchar();
  KEY;
}
```

Действителният код който се подава на компилатора от предпроцесора е :

```
#include <conio.h>
#include <stdio.h>
int main(int argc, char** argv){
  int I;
  printf("\n Input nummber: ");
  I = getchar();
  while( getch() != '\n'); // Добавен код от предпроцесора
}
```

Както се вижда идентификатора **KEY** е заменен с израза **while(getch() != '\n').** Тази замяна ще се извършва навсякъде в кода, където ще се срещне идентификатора **KEY**.

Макроси с параметри

В C++ е позволено предаването на параметри на макроси. Синтаксиса на една такава конструкция е следния:

#define <име>(Парам1,Парам2,…) <макрос>

Както е видно от дефиницията е възможно да бъдат предадени един или няколко параметъра.

Пример:

```
#include <iostream>
#define PI 3.14
#define S( R )  PI*R*R
using namespace std;
int main(int argc, char** argv) {
  double r;
  cout << "\n Input radius: ";
  cin >> r;
  cout << "\n face circle:" <<  S( r );
  return 0;
}
```

И съответния код за компилация :

```
#include <iostream>
using namespace std;
int main(int argc, char** argv) {

  double r;
  cout << "\n Input radius: ";
  cin >> r;
  cout << "\n face circle:" <<  3.14*r*r ;
}
```

Един и същ идентификатор може да приема различни значения за определени области от програмата. Всяко едно значение важи до срещането на нова декларация. За да се върнем към предходната декларация може да се използва нов оператор **#define** или да се използва директивата **#undef**.

Директивата **#undef** има следният синтаксис: **#undef <име>.**

```
#define N 100
void main()
{
   ….
   for(I=1 ; I < N; I++)
   …
}

#define  N 200
int SampleFun1()
{
   ….
   for(I=1 ; I < N; I++)
   …
}

#undef  N
int SampleFun1()
{
   ….
   for(I=1 ; I < N; I++)
   …
}
#define N
```

Отмяната на макродефиниция става като се декларира отново същият идентификатор но без да му се задава стойност.

Включване на файлове

C++ изпълнява директива **#include** съгласно дефиницията в K&R, за включване на заглавни файлове(файлове с декларации и дефиниции - **????????.H**) и има следния формат.

```
#include "fname.h"
#include <fname.h>
```

При директива във формата **#include "fname.h"**, ако предпроцесорът не може да намери файла в подразбиращата се директория, той ще го търси и в директориите зададени в пътя за търсене на компилатора.

При директива във формат **#include <fname.h>**, файлът се търси само в директориите, зададени в пътя за търсене на компилатора.

Забележка: Заглавните файлове са файлове с декларации и дефиниции. Те осъществяват интерфейса между различните функции на програмата, като осигуряват общ достъп до информация за прототипите на функциите, различни видове макроси, декларации на променливи и константи. Прието е те да носят разширение ".h" или ".hpp". Заглавните файлове се включват в други файлове чрез директива **#include** на предпроцесора на C++. Освен декларации и дефиниции, те могат да съдържат и други команди на предпроцесора, а също така и самата директива **#include** за включване на друг файл.

Потребителят има възможност да конструира спецификацията на файла в директива **#include**, включително и разделителите, с помощта на макроопределения. Ако редът след ключовата дума започва с идентификатор, предпроцесорът преглежда текста за макроопределения. Ако символният низ е ограден в кавички или ъглови скоби, C++ не търси макроопределения. Да разгледаме примерите:

```
#define myincl "c:\msvc\include\mystuff.h"
#include myincl
#include "myincl"
```

Първата директива **#include** ще принуди предпроцесора да търси файл със спецификация c:\msvc\include\mystuff.h, а при втората ще се търси myincl.h в подразбиращата се директория. При макроси за директива **#include** не е позволено да се използува конкатенация на символни низове и механизъма за долепяне на аргументи от макроопределението.

Условна компилация

C/C++ поддържа дефиницията на K&R за условна компилация чрез заместване на определени редове с редове, съдържащи интервали. По този начин се игнорират редове, започващи с директива **#if**, **#ifdef**, **#ifndef**, **#else**, **#elseif** и **#endif**, както и редовете, които не трябва да бъдат компилирани в резултат на включените директиви. Всички директиви за условна компилация трябва да завършват в същия файл, в който е началото им.

C++ поддържа ANSI оператора **#define (Символ)**, който дава стойност 1 ("истина"), ако **Символ** е дефиниран преди това (с

помощта на **#define**). В противен случай стойността е 0 ("неистина"). Например:

```
#if defined(mysym)
```

което е идентично с

```
#ifdef myssym
```

Освен това defined може да се използува многократно в сложни изрази след директива **#if**. Например:

```
#if defined(mysym) || defined(hissym)
```

Друго предимство на C++ е, че позволява използването на оператор **sizeof** в изрази за предпроцесора. Допустими са следните директиви:

```
#if (sizeof(void *) ==2)
#define SDATA
#else
#define LDATA
#endif
```

Директива #line

Директивата **#line** е предназаначена за задаване номер на линията в листинга на програмата. Следващия пример илюстрира съвместното използване на **#line** и макросите **__LINE__** и **__FILE__.** В примера се задава начален номер на линиите 151 от файл с име copy.c.

```
#line 151 "copy.c"
```

В този пример макроса ASSERT предефинира макросите **__LINE__** и **__FILE__** при принтиранена съобщенията за грешка когато за дадения cocpc файл е дефинирана to print an error message about the source file if a given "assertion" is not true.

```
#define ASSERT(cond) if( !(cond) ) \
{printf( "assertion error line %d, file(%s)\n",__LINE__,
__FILE__ );}
```

Директива #error (ANSI C)

С поддържа директива **#**error, която е спомената без да е дефинирана в ANSI стандарта. Форматът е:

```
#error Съобщение_за_грешка
```

Когато директивата е включена в програмния файл, при условна компилация, ако условието не се изпълни, предпроцесорът ще прочете директивата **#error** и ще прекрати работата си със следното съобщение:

```
Fatal: ИмеНаФайл #line Error directive:
Съобщение_за_грешка
```

Предпроцесорът преглежда текста на програмата, за да изключи

коментарите, но показва всички останали текстове, без да следи за вписани макроопределения.

Директива #pragma (ANSI)

C/C++ поддържа директивата **#pragma**, чиято дефиниция в стандарта ANSI не е напълно изяснена. Задачата ѝ е да позволява директиви, специфични за различни реализации на езика, в следния формат:

#pragma <ИмеНаДиректива>

С директива **#pragma** C/C++ може да дефинира желаните директиви без опасност от конфликти с работата на други компилатори, поддържащи директивата **#pragma**. Това е така, тъй като по дефиниция, ако компилаторът не разпознава директивата **"ИмеНаДиректива"**, той игнорира оператора **#pragma**. C/C++ разпознава две директиви **#pragma** - **inline** и **warn**.

#pragma inline

Директивата inline има формата:

#pragma inline

Тя е еквивалентна на опция **-b** на компилатора (за компилаторите на Borland). Осведомява компилатора, че програмата съдържа вписани редове на асемблер и обикновено стои в началото на файла, тъй като компилаторът се рестартира сам с опция **-b** при срещането на директива **#pragma inline**.

В действителност програмистът може да не използува опция **-b** и **#pragma inline**, тъй като компилаторът ще бъде рестартиран и ще използва опция **-b** автоматично при първия срещнат вписан оператор на асемблер. Целта на явното използване на опцията или директивата е да се спести времето, което би се изгубило поради рестартирането, след като вече част от програмата е обработена.

Втората **#pragma** има формата:

#pragma warn

Тя позволява да се променят специфичните опции -wxxx за командната версия (или специфичните опции на C/C++ - интегрирана работна среда: Display warning...On). Например, ако програмата съдържа директивите:

```
#pragma warn +xxx
#pragma warn -yyy
#pragma warn .zzz
```

Предупредителното съобщение xxx ще бъде включено (дори ако подменюто <O/C/Errors> е с опция off). Предупредителното съобщение **yyy** ще бъде изключено, а **zzz** ще бъде в положението (включено или изключено), в което е било, когато е започнала компилацията на файла.

Празна директива (ANSI)

Само поради стремеж към пълнота, стандартът ANSI разпознават празната директива, която се състои от ред, съдържащ само символа **#**. Компилаторите игнорират директивата.