**Препроцесор С**

**Директиви С препроцесор: #define, #include, #undef, #if, #ifdef, #ifndef, #else, #endif**

**Езикът C/C++ предоставя на разположение възможност, която в другите изици от високо ниво на програмиране не се среща. Тази възможност се сруща само в Асемблерните езици. Превеждането на на сорс кода в C/C++ започва р работата напрепроцесора, който преработва програмата и подава резултата от своята работа на компилатора. Препроцесора**

**поддържа следните функции:**

**------------------------------------------------------------------------  
Използване на макродефиниции;  
Включване на хедър файлове;  
Условна компилация.**

**Като програма преппроцесорът работи по следния начин:  
Анализира сорс кода на програмата;  
Замества или вмъква пасажи от сорса с други , които са допустими в езика С++ конструкции;  
Резултатът от работата на препроцесора е ASCII- файл, който ще се използува от компилатора. Преработката, която трябва да извърши препроцесора се задава с помощта на директивите му. Те започват със знака # и в противовес на обикновените С++ инструкции не завършват с точка и запетая. Препроцесора притежава следните директиви:  
  
Директива  
Значение  
#include  
#define  
#undef  
#if  
#elif \*  
#else  
#endif  
#ifdef  
#ifndef  
#error \*  
#line  
#pragma \*  
Включване на файл.  
Дефиниция на макрос.  
Нулиране на макрос.  
Проверка на константен израз от целочислен тип.  
Проверка за алтернатива на една #if директива.  
Изпълнява се, ако нито една от алтернативите #if или #elif не е TRUE.  
Край за алтернатива #if .  
Проверява дали е дефинирана символна константа.  
Проверява дали не е дефинирана символна константа.  
Дава възможност за извеждане на съобщения за грешка.  
Задава на следващия след директивата ред желана стойност.  
Директива на компилатора. Инструкция за компилатора.  
Означениете със символа ‘\*’ директиви се отнасят само за C++ компилатора.**

**Включване на файлове.  
  
Включването на съдържанието на друг файл в сорс на програмата се извършва с помощта на директивата #include. Директивата е позната от примерите в предишните глави и има следния синтаксис на употреа:  
  
#include <dateiname.h>  
или  
#include “dateiname.h”,  
  
при това dateiname.h и в двата случая е името на файла.   
Разликата е ,че в първия случай препроцесоът търси файла в стандартната Include-директория /папка INCLUDE/. Във втория случай препроцесора ще извърши търсенето в директорията, в която се намира съответния сорс или ще извърши търсенето в съответствие с указан път на достъп. Ако търсенето в последния случай е неуспешно, то то ще бъде продължено в стандартната дириктория по подразбиране. Заглавните файлове съдържат обикновено константи, дефиниция на типове, декларации на променливи и прототипи на функции. Заглавните файлове могат да съдържат други заглавни файлове. Разширението им може да бъде произволно, но като общо предписание е избрано \*.h. Единственото изискване е данните да бъдат написани в ASCII-формат.  
Заедно със системните файлове, които се доставят със програмното окръжение на средата, програмистът може да създава собствени файлове, които да използува и които са свързани неговия проект.   
  
5.3. Макродефиниции.  
  
Макрадефиницията е дефиниция, която свързва символно име с валидна в езика С символна последователност. МАкродефиницията започва с #define- директивата и има следния синтаксис:  
  
#define konst\_name [text] ;  
  
при което konst\_name е име на символната константа, а текст е произволна символна последователност, на която се съпоставя символната константа. Средните скоби не се поставят в програмата. Когато процесорът срещне съответното име в сорса, той замества преди компилацията всички символни константи със съответните последователности. В много случай символното име на константта се нарича и име на макрос. Обикновено това име се пише с голяма буква.  
Примери:  
#define CNST 25.2  
#define MAX 10  
#define DURCH 15.8  
#define PI 3.14159  
  
Първата дефиниция дефинира име символна константа за стойността 25.2. Символичното име може да бъде използувано във всеки израз, който е допустим за езика C/C++. Тъй като константатат PI е дефинирана, то са допустими следните следните изрази:  
  
C - Ausdruck C++ - Ausdruck  
printf (“%f“, PI); cout <<PI;  
printf (“%f“,PI\*DURCH\*DURCH); cout << PI\*DURCH\*DURCH;  
  
които при компилацията се замества с:   
  
printf (“%f“, 3.14159); cout <<PI;  
printf (“%f“,3.14159\*15.8\*15.Cool; cout << 3.14159\*15.8\*15.8;  
int array [MAX]; /\* Deklaration einer Matrize \*/  
  
Символните константи се могат да бъдат заместени не само с числа, но и с произволни символни последователност. Дългите символни последователности могат да бъдат пренесени на нов ред с помощта на символа „\”. С помощта на макродефиниции една С++ програма може така да бъде написана, че да изглежда като паскалска програма.  
  
#include <iostream.h>  
#include <conio.h>  
#define PI 3.14159  
#define DURCH 15.2  
#define begin {  
#define end }  
#define write(s) cout<<s  
#define program void main ()  
#define readln getche ()  
  
program  
begin  
write ("Die fläche des Kreises ist ");  
write (PI\*DURCH\*DURCH);  
readln;  
end  
Ergebniss  
  
Die fläche des Kreises ist 725.833  
  
Това е интересен пример, но не е препоръчително синтаксисъс на Pascal-програмите да бъде имитиран в C++ програми по този начин. Когато в един сорс се срещнат две дефиниции на една и съща константа, то тогава първата дефиниция е валидна до момента на срещата на второта.  
  
5.4. Константи без текст за заместване. Отмяна на дефиниция на константи.   
  
Възможна е употребата на костати без текст, който да ги замества. Използува се следния синтаксис:  
  
#define konst\_name  
  
Поради липса на текст с който константата konst\_name да бъде заместена на всички места където се среща тя ще бъде отделена. Независимо от това тази константа се смята за дефинирана и с нейна помощ могат да се формулират условия, от които зависи изпъленнието на други инструкции. Начинът за отменяне на дефиницията на една константа се извършва в съответствие със следния синтаксис #undef:  
  
#undef konst\_name.  
  
Тази инструкция се използува по-рядко, защото случаите в които може да е от полза са малко.  
Когато се отмени дефиницията на една символна константа с помощта на директивата #undef, то тя след това повече няма да съществува и нейното използуване след този момент ще се счита като греша от компилатора. Тази ситуация се онагледява от следния програмен абзац::  
Пример:  
. . . . .   
#define PI 3.14159  
#define DURCH 15.2  
#define begin {  
#define end }  
#define write(s) cout<<s  
#define program void main ()  
#define readln getche ()  
  
program  
begin  
write ("Die fläche des Kreises ist ");  
write (PI\*DURCH\*DURCH);  
#undef PI /\* Löschen der Konstante PI \*/  
write (PI); /\* Fehlermeldung PI ist nicht mehr definiert \*/  
readln;  
end  
  
Коментарите след директивите на препроцесора се описва с помощта на /\* \*/, обаче тъй като някой С++ компилатори използуват C препроцесор това може да доведе до грешка.   
  
5.5. Макроси без параметри.  
  
Понятието макрос описва в общия случай последователност от инструкции или команди, които са обединени от едно общо име. Тези инструкции могат в една програма да се да бъдат използувани вместо последователност от инструкции. Както и преди беше споменато някои книги не правят разлика между символните константи и и макросите. Ако желаем да бъдем изчурпателни, то за един макрос мое да се даде следната дефиниция. За макрос се говори, когато с дадено име е свързано изпълнението на определени инструкции, в противен случай се говори за символна константа. Синтаксисът за дефиниране на макрос без пареметри е следния и той не се отличава от този на символната константа.  
  
#define makro\_name text  
  
Под text се разбира една или повече инструкции, една или повече операции за които ще бъде използува макросът например с дефиницията   
  
#define readln getche ()  
  
се дефинира макросът readln, при който програмата спира до тогава, докато не се етисне произволен клавиш. Текстът може да се състои от множество инструкции:  
Пример:  
  
#include <stdio.h>  
#include <conio.h>  
#define CLS clrscr (); /\* Makro – Der Bildschirm wird gelöscht \*/  
#define MINR 10 /\* Manimaler Wert von Radius des Kreises \*/  
#define MAXR 15 /\* Maximaler Wert von Radius des Kreises \*/  
#define PI 1.14159  
#define NEWL printf ("\n") /\* Makro – neue Zeile \*/  
#define CIRC for (int rad=MINR; rad<MAXR; rad++)\   
{printf ("S = %f, R = %d", PI\*rad\*rad, rad); NEWL;} /\* Einnesten Makro in Makro \*/  
void main ()  
{  
CLS; /\* Schirm löschen \*/  
CIRC; /\* Berechnung der Fläche des Kreises \*/  
}  
  
Ergebnis  
  
S = 114.159000, R = 10  
S = 138.132390, R = 11  
S = 164.388960, R = 12  
S = 192.928710, R = 13  
S = 223.751640, R = 14  
  
Тъй като е допустимо един макрос да бъде влоен в друг макрос, то е възможно мокросът NEWL да бъде вложен в макроса CIRC .  
  
5.6. Макроси с параметри.  
  
Макроси, които съдържат параметри се дефинират в съответствие със следния синтаксис:  
  
#define makro\_name(pr1, pr2, . . . prn) text  
  
При което рr1, pr2, . . . prn, са имената на макропараметрите. Списъкът от параметри трябва непосредствено и без празни символи да се запише след името на макроса. В противен случай те ще бъдат изтълкувани като част от текста за заместване. Например с последователността:   
  
#define MULT(x, y) (x)\*(y)  
  
Ако се дефинира макросът MULT с цел умножението на две произволни числа, то параметрите в макродефиницията са формални параметри, които не трябва да се дефинират и служат, само за да заемат място в паметта. Актуалните стойности се предават при извикване на макроса. При извекването на MULT (10, 12) двете стойности са актуални парамери, така че стойностите 10 и 12 ще се предадата,а резултатът ще бъде 10\*12=120. Изразът  
MULT(10, 12)  
Ще бъде заместен от препроцесора с   
(10)\*(12)  
Може да се види, че се замества не макросът със съответния текст, а актуалтите параметри се поставят на мястото на формалните параметри. Важно е да се отбележи, че вместо параметри от тип int могат да се предадат и параметри от тип float:  
  
MULT(12.5 15.3)  
  
Трябва да се допълни, че могат да се предадат произволни имена на променливи като параметри и те биха могли да не съвпадат с имената на формалните параметри. При макродефинициите е от голямо значение мястото на скобите. Макропараметрите венаги трябва да се заграждат в скоби, за да бъдат избегнати грешки в случаи като следния. Нека е зададен макросът MULT по следния начин:  
  
#define MULT(x, y) x\*y  
  
Изразът   
  
MULT(x, y+1);   
  
, който би трябвало да извърши умножението на стойността y+1 на x, поради липсата на скоби за параметрите ще доведе до следното:  
  
x\*y+1 // x= 10 , y = 12  
  
Стойността ще бъде /x\*y = 121/, което е и грешно. Ако параметрите са заградени в скоби, то тогава резултатът би бил:  
  
#define MULT(x, y) (x)\*(y)   
  
Наличието на скоби за параметрите (x)\*(y) /x = 10, y = 12 / дава като резултат следното 130.  
Фантът, че не само параметрите, но и самият текст за заместване трябва да бъде ограден със скоби може да се онагледи със следния пример. Нека да е необходимо да се напише макрос, който да извършва деление на две числа.  
  
#define DIV(x, y) (x)/(y)  
  
Изразът:   
  
5\* DIV(12, 10)   
  
ще доведе до 5\*(12)/(10) = 60/10 = 6. Получената стойност ще бъде 6, която обаче не е правилният резултат, тъй като към израза се прилага стандартния приоритет и асоциативни правила на изпълнение на операциите. Избягването на такива грешки се постига с поставянето на целия заместващ текст в скоби.  
  
#define DIV(x, y) ((x) / (y))  
  
Тъй като развитието на макроса води до получаване на резултат 5, то това ще е правилният резултат.  
  
#include <stdio.h>  
#define DIV(x, y) ((x)/(y))  
void main ()  
{  
printf ("Erg = %d",5\*DIV(12, 10));  
}  
  
Дефиницията на макрос ое да бъде отменена по същия начин, по които се отменя и дефиницита на символна константа. Директивата е #undef. Достатъчно е да се запише името на макроса без параметри.   
Синтаксис :  
  
#undef makro\_name  
След дефиницията   
  
#define DIV(x, y) ((x)/(y))  
с помощта на директивата   
  
#undef DIV  
дефиницията на макроса DIV се отменя. Името DIV за препроцесора няма да има повече значение и следващато извикване на името ще доведе до грешка.   
  
5.7.Оператор ##  
Използуването на този оператор е много рядко. Използува се основно в текста за заместване и обединява два основни символа. Той действа подобно на конкатенация между символите.   
  
#define CONC(symbol) int z##symbol =5\*DIV(12, 10)  
По този начин се дефинира един макрос, който дефинира една променлива от тип int с име Zn, след което на тази променлива й се присвоява стойност с помощта на макроса DIV(x, y). Следва програмата, използуваща отзи макрос:  
  
#include <stdio.h>  
#define DIV(x, y) ((x)/(y))  
#define CONC(zeichen) int z##zeichen=5\*DIV(12,10)  
  
void main ()  
{  
CONC(1); /\* Variabledefinition \*/  
printf ("z1=%d", z1); /\* Herausgabe des Wertes der Variable z1 \*/  
}  
  
Ergebnis  
z1=5  
  
5.8. Условна компилация.  
С помощта на директивите #if, #else, #elif, #endif, #ifdef и #ifndef и #define е възмоно, само тогава да се компилират определени части от програмата, когато е изпълнено определено условие. За да се извърши условна компилация е необходимо да се съблюдава следния синтаксис:  
  
#if konst\_ausdr  
Programmteil\_1  
[#elif konst\_ausdr  
Programmteil\_2  
#elif konst\_ausdr   
Programmteil\_3  
. . . . . .   
#else   
Programmteil\_N  
]  
#endif  
  
С помощта на тази конструкция е възможно сорс кода на програмата да се модифицира и да се избере тази версия, която най-добре подхожда на целите ни.**

**Частта от израза konst\_ausdr представлява в известен смисъл логическото условие на конструкцията #if съответно #elif, и чието оценяване дава като резултат цяло число. Скобите показват, че не е необходимо #elif или #else Zweige да съществуват , но те не участвуват в синтаксиса на конструкцията. Всеки #if завършва с #endif инструкция. Инструкциите между тях може да съдържа както C++ инструкции, така и други препроцесорни директиви.   
Ако препроцесорът при оценява един константен израз като различен от NULL, то инструкциите, намиращи се между енговото начало и край ще бъда предадени на компилатора за обработка. Другите части на програмата ще бъдат изключени от сорс кода.   
Ако нито един от константните изрази в директивите #if или #elif няма стойност TRUE, то тогава се компилира частта от програмата намираща принадлежаща на #else инструкцията. Това може да се онагледи със следния програмен код.   
Необходимо е да се създаде програма, която използува три различни формули в зависимост от стойността на символната константа FORM. Формулите са следните:  
  
1.a\*x\*x+b\*x+c  
2.a\*x\*x\*x+b\*x\*x+c  
3.a\*x+c  
Следва сорс кода на програмата.  
  
#include <stdio.h>  
#include <math.h>  
#define FORM 1  
#define a 10  
#define b 5  
#define c 15  
void main ()  
{  
float x, result;  
  
printf ("Geben Sie x ein :");  
scanf ("%f", &x);  
#if FORM==1 /\* Diese Zweige kompiliert, wenn FORM gleich 1\*/  
result=a\*x\*x+b\*x+c;  
printf ("F=%d result=%f", FORM, result);  
#elif FORM==2 /\* Diese Zweige kompiliert, wenn FORM gleich 2\*/  
result=a\*x\*x\*x+b\*x\*x+c;  
printf ("F=%d result=%f", FORM, result);  
#elif FORM==3 /\* Diese Zweige kompiliert, wenn FORM gleich 3\*/  
result=a\*x+c;  
printf ("F=%d result=%f", FORM, result);  
#else /\* Diese Zweige kompiliert, wenn FORM nicht oder anders definiert\*/  
printf ("Es gibt keine Formel zur berechnung ");  
#endif  
}  
  
Ergebnis  
Geben Sie x ein:5  
F=1 result=290.000000**

**Инструкции #error и #line  
  
Декларацията #error има следния синтаксис на използуване:  
#error message\_text  
Следната част от програма издава съобщение за грешка, защото стойността на константата MAX е по-малка от 100.  
  
#define MAX 50  
. . . . . .  
#if MAX <100  
#error Für MAX ist einen minimalen Wert von 100 erforderlich.  
#endif  
  
Директивата #error предизвиква прекъсване на компилацията, за да може да се извърши отстраняване на грешката.   
Директивата #line може да се използува при компилация и има следния синтаксис:  
  
#line deс\_konstante [“Zeil\_name”]  
  
Константата dec\_konstante е цяла десетична константа. Програмистът може да зададе произволен номер на реда който се намира след директивата #line освен това може да се зададе име на файл което да се използува вместо името на сорс-файла.  
  
void main ()  
{  
#line 10 "for\_anw"  
for (int i=0; i<10, i++);  
return 0;  
}  
Compiling NONAME00.CPP:  
Error FOR\_ANW 10: For statement missing ; in function main()  
Warning FOR\_ANW 12: 'i' is assigned a value that is never used in function main()  
  
Когато започне номерацията на радовете с 10 и името на сорса се замени с “for\_anw”, то тогава компилаторът издава съобщения, които се записват след тялото на програмата.**