CAPITOLUL 13

13. Preprocesorul

Orice program C, în faza de compilare trece întâi printr-o preprocesare şi abia apoi se va genera codul echivalent în asamblare. Preprocesorul după cum îi spune şi numele, face prima procesare asupra sursei C conform directivelor incluse direct sau indirect, prin includere de bilioteci, de către programator.

Directivele preprocesorului sunt linii sursă al căror prim caracter diferit de spațiile albe este # (diez). Liniile care conțin directive ale preprocesorului sunt eliminate din textul sursă înainte ca acesta să fie citit de compilator. Preprocesorul înlocuieşte orice secvență de spații albe (inclusiv comentarii) cu un singur spațiu.

13.1. Directiva nulă

O linie care conține numai caracterul # (în afară de eventualele spații albe) este ignorată de preprocesor.

13.2. Directivele #define şi #undef

"#" "define" identificator [["(" lista_argumente ")"] corp] .
"#" "undef" identificator

Directiva #define defineşte o macroinstrucțiune. Dacă macroinstrucțiunea are argumente, paranteza stângă trebuie să apară în definiție lipită de numele definit. Argumentele se separă prin virgulă. Corpul macrodefiniției nu poate avea mai mult de o linie sursă (se poate continua linia cu \).

În faza de preprocesare, apariția numelui de macro în textul sursă (eventual urmat de o listă de argumente actuale in paranteze) este înlocuită cu corpul de macro, în care parametrii formali au fost înlocuiți cu textul parametrilor actuali.

După substituție, textul obținut este citit din nou, până când nu mai sunt necesare substituții (directivele preprocesorului nu sunt recunoscute în textul produs prin expandarea unei macroinstrucțiuni). Numele de macro nu sunt recunoscute în şiruri de caractere sau comentarii. O macroinstrucțiune nu se poate apela recursiv.

La apelul unei macro cu argumente, numai virgulele de pe primul nivel sunt luate ca separatoare de argumente (nu şi cele care apar în paranteze sau între apostrofe sau ghilimele).

În corpul unui macro se pot folosi următorii operatori care nu acționează decât în faza de preprocesare:

- ## care dispare total, nerămânînd nici măcar un spațiu, şi
- # care, plasat in fața unui argument formal, îl înlocuieşte cu argumentul actual PLASAT ÎN GHILIMELE.

Directiva #undef şterge o macrodefiniție din tabela de macrodefiniții.

13.3. Directiva #include

#include <nume_fisier>
#include "nume_fisier"
#include nume_macro

Numele de macro până la urmă trebuie să se expandeze în ceva de forma <nume_fisier> sau "nume_fisier". Efectul directivei constă din inserarea în unitatea de compilare, în locul directivei #include, a textului din fişierul respectiv. Textul inserat poate conține şi el directive de preprocesare, inclusiv directiva #include.

Dacă în numele de fişier apare şi o cale de acces, se va căuta numai acolo. Altfel, dacă s-a folosit forma cu ghilimele, se caută în directorul curent. Urmează, dacă tot nu s-a localizat fişierul, să se caute pe rând în directoarele din lista dată la instalare, sau pe linia de comandă, sau în mediul integrat.

13.4. Directive de compilare condiționată

"#" "if" expresie_constantă_1
[text_1]
["#" "elif" expresie_constantă_2
text_2]...
["#" "else"
text_else]
"#" "endif"

Dacă expresia_1 nu este nulă, atunci se compilează text_1; altfel, dacă expresia_2 nu este nulă, atunci se compilează text_2; etc.; altfel, dacă este prezentă directiva #else, se compilează text_else. Directivele de compilare condiționată pot fi îmbricate.

Expresia condițională poate conține operatorul de preprocesare defined. Acesta are un argument, un nume de macro eventual în paranteze. Valoarea expresiei defined (nume) este 1 daca numele este definit cu #defined, sau 0 dacă nu există o macro cu acel nume. Un identificator definit cu un corp de macro vid se consideră definit.

#ifdef nume ... echivalentă cu #if defined (nume)
#ifndef nume ... echivalentă cu #if !defined (nume)

13.5. Directiva #line

#line număr_linie ["nume_fisier"]

Efectul directivei constă în aceea că în fişierul produs de C (preprocesorul C oferit ca utilitar) sau in mesajele de eroare, etc. liniile care urmează după #line vor fi referite ca şi cum ar începe cu linia numărul număr_linie şi ar veni din fişierul nume_fişier.

De obicei directiva #line nu apare în programele scrise cu mâna, ci mai ales în programele generate automat. Dacă numele de fişier lipseşte el rămâne cel valabil în acel moment.

În argumentele directivei #line, macroinstrucțiunile sunt expandate, ca şi la directiva #include.

13.6. Directiva #error

#error mesaj

Efectul ei constă în generarea unui mesaj de eroare de forma

Error: nume_fişier line număr_linie : Error directive : mesaj

Compilarea unei directive #error opreşte compilarea. De obicei apare în fişierele antet (incluse cu directiva #include), în cadrul unei secvențe compilate condiționat. De exemplu:

```
#if !defined ( __cplusplus )
   #error Nu se poate compila decit in C++
#endif
```

13.7. Directivele #pragma

#pragma argsused

Poate apare numai în afara unei funcții. Are ca efect suprimarea mesajului de avertisment referitor la neutilizarea unui parametru formal, dar numai în prima funcție compilată după apariția directivei.

#pragma exit nume_functie [prioritate]
#pragma startup nume_functie [prioritate]

Inserează funcțiile citate (definite ca void nume_functie (void)) pentru apel înainte de main (respectiv după terminarea funcției main). Prioritatea este un întreg de la 0 la 255. Funcțiile cu priorități mai mici vor fi apelate mai devreme la lansare şi mai târziu la terminarea execuției. Prioritățile de la 0 la 63 sunt rezervate pentru rutinele din bibliotecile Turbo. Dacă prioritatea lipseşte, se ia 100. Funcțiile citate trebuie declarate înaintea apariției directivei.

#pragma inline

Anunță compilatorul că în program există cod inline în limbaj de asamblare. Este echivalentă cu opțiunea de compilare -B. Întâlnirea directivei #pragma inline, dacă opțiunea -B nu este activă, are ca efect relansarea compilării cu opțiunea -B activată.

#pragma option

#pragma option optiune [,optiune]...

Opțiuni de compilare care pot apare în #pragma option NUMAI ÎNAINTE DE FOLOSIREA UNUI NUME DE MACRO CARE ÎNCEPE CU DOUĂ LINIUȚE DE SUBLINIERE, SAU ÎNAINTE DE APARIȚIA PRIMULUI ATOM LEXICAL C:

-E nume_asamblor
-f opțiune_flotantă
-i număr_caractere_semnificative
-m model_memorie
-n director_ieşire
-o nume_fisier_ieşire
-u (generare de liniuțe de subliniere pentru globali)
-z nume_segment

Opțiuni care pot apare numai dacă apar în afara oricăror declarații de obiecte sau funcții:

-1 (instrucțiuni 80186)
-2 (instrucțiuni 80286)
-a (controlul alinierii)
-ff (control "rapid" al operațiilor în virgula mobilă)
-G (optimizare pentru viteză)
-k (control normal al stivei)
-N (controlul stivei)
-O (controlul optimizării)
-p (secvență de apel Pascal)
-r (alocare registre)
-v (control pentru depanare)
-y (control asupra numerelor de linii în codul obiect)

Opțiuni care pot fi date oriunde si au efect imediat:

-A (setul de cuvinte cheie)
-C (comentarii îmbricate)
-d (contopirea şirurilor duplicate)
-g număr (oprire după număr avertismente)
-j număr (oprire după număr erori)
-K (tipul char este fără semn)

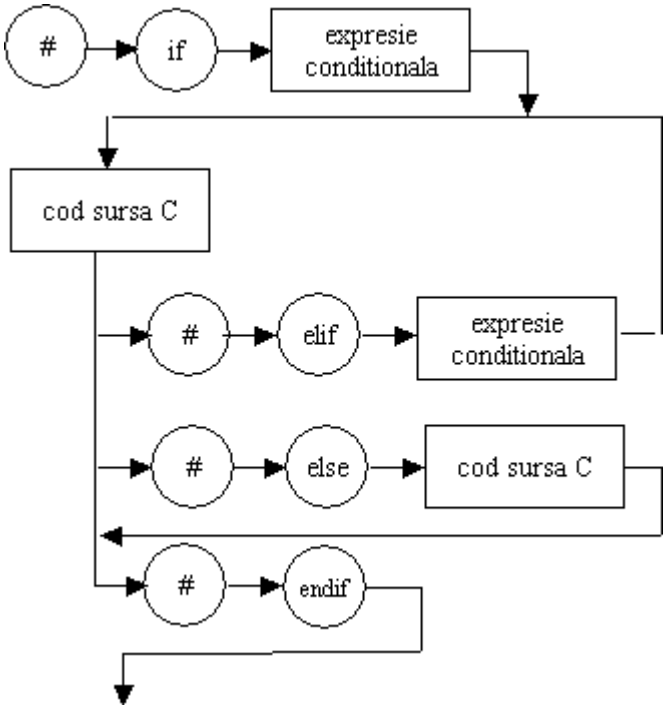
#pragma warn
#pragma warn +xxx Avertismentele xxx sunt activate
#pragma warn -xxx Avertismentele xxx sunt inactivate

După cum am văzut preprocesorul permite şi declararea anumitor zone de cod astfel încât ele sa fie compilate în anumite condiții cum ar fi :

- ➔ nu sunt definite anumite constante sau variabile locale
- ➔ codul este util, dar pentru depanare

➔ doresc să impiedic folosirea unor funcții din bibliotecă întru-cât sunt rescrise de oricine(supraîncărcarea funcțiilor)

Sintaxa generala :



Exemplu 1.
pentru headere personale să zicem fişierul test.h

```
#ifndef  __TEST
#define __TEST
.......
corpul
.......
#endif

Sau pentru depanare
#if DEBUG
        printf(“.......”);
#endif

Sau daca nu am definite ni;te valori speciale
#ifndef  M_PI
#define M_PI 3.1415....
#endif.
```

Scurtă prezentare a regulilor de bază de care trebuie ţinut cont la proiectarea unei applicaţii

Stil de programare nerecomandat	Stil recomandat
Punerea declaraţiilor cu extern în fişierul.c	Referirile la funcţie/variabilă extern Se pun în fişierele header.h
Accesul comun la date între funcţii realizat prin variabile globale	Accesul comun se face prin transfer de argument
Folosirea de date globale când nu e cazul	Se rezolvă prin declarare de tip static
Crearea a numeroase funcţii cu scopuri speciale	Creearea unor primitive generale
Utilizarea numelor de fişier sau variabilă fără legătură cu contextul	Folosirea de nume care se apropie de pseudocod
Folosirea constante numerice	Se definesc nişte nume pt. acele valori
Folosirea goto	Folosirea structurilor de control a fluxurilor
Scrirea de cod redundant	Se folosesc funcţii în cazul în care se observă acelaşi lucru