Laborator 3 – Operaţii de intrare-ieşire standard

Directiva #define

Directiva #define este utilizată pentru definirea constantelor şi a pseudofuncţiilor. În figura 1. este prezentată forma generală a directivei #define.

SINTAXĂ	Forma 1: #define nume_macro secventa_caract sau Forma 2: #define nume_pseudofunctie(lista parametri) expresie
EXEMPLE	1. #define G 9.81 2. #define TRUE 1 #define FALSE 0 3. #define er “eroare de calcul\n“ 4. #define S “acesta este un mesaj foarte lung\ utilizat spre exemplificare“ 5. #define MAX 20
EXEMPLE	6. #define X 34 #define Y X+7 7. #define ABS(x) (x)<0?-(x):(x)

Fig. 1

În forma 1 a directivei #define:
nume_macro reprezintă un identificator şi este precedat şi urmat de cel puţin un spaţiu. Conform standardului ANSI C identificatorul se numeşte macrocomandă iar procesul de înlocuire, înlocuire a macrocomenzii;
secventa_caract reprezintă o succesiune de caractere, care începe cu primul caracter care nu este alb. Secvenţa de caractere odată începută poate fi încheiată doar printr-un caracter delimitator de rând nou. Dacă lungimea şirului depăşeşte un rând, acesta se poate continua pe rândul următor înserând un backslash (\) la sfârşitul rândului.
Cu ajutorul acestei construcţii se va înlocui nume_macro cu secventa_caract peste tot în textul sursă al programului care urmează construcţiei. De obicei programatorii în C folosesc

litere mari pentru nume_macro. Această formă a directivei #define este utilizată frecvent pentru a defini constante. De aceea nume_macro se mai numeşte şi constantă simbolică.

Observaţii:
1. Dacă nume_macro apare într-o linie de comentariu sau într-un şir de caractere, înlocuirea nu va avea loc.
2. Este indicat să se insereze toate directivele #define la începutul fişierului sau într-un fişier antet separat.
În exemplele din figura 4.1 efectul directivei #define este următorul:
1. se defineşte constanta de acceleraţie gravitaţională G şi i se atribuie valoarea 9.81. Este mai sugestiv sa folosim constanta simbolică G decât valoarea ei. De asemenea dacă această constantă se foloseşte de mai multe ori în program este mai simplu să folosim numele ei prescurtat în program.
2. se definesc TRUE (adevărat) cu valoarea 1 şi FALSE (fals) cu valoarea 0, în scopul utilizării lor apoi în programul sursă.
3 şi 4. Se defineşte un mesaj de eroare atribuind identificatorului er respectiv S şirul de caractere corespunzător.
5. se defineşte constanta MAX cu valoarea 20. Ea poate reprezenta dimensiunea maximă a unui tablou. Numele acestei macrocomenzi va fi utilizat ori de câte ori este necesar. Astfel la declararea tabloului a, se poate utiliza această constantă.
float a[MAX];

Astfel dacă se doreşte modificarea dimensiunii tabloului se va modifica doar instrucţiunea #define şi se va recompila programul. Toate referirile ulterioare la MAX vor fi actualizate automat la recompilarea programului.

6. Lui X i se atribuie valoarea 34 după care odată definită această constantă ea poate fi folosită la definirea altei constante. Deci Y va lua valoarea 34+7, deoarece preprocesorul face doar substituţii nu şi calcule.

Forma 2 a directivei #define este utilizată pentru crearea de pseudofuncţii. Această formă poartă numele de macrocomandă tip funcţie (function-like macro).
lista_parametri este o listă de parametri care conferă pseudofuncţiei o flexibilitate deosebită în generarea diverselor rezultate.

La fiecare apariţie a numelui pseudofuncţiei, argumentele folosite în definiţie pot fi înlocuite prin argumentele efective din program. Aceste macrodefiniţii sunt mai rapide ca funcţiile obişnuite dar ocupă un spaţiu de memorie mai mare. Dacă dimensiunea macrocomenzii tip funcţie este mare va duce la o creştere a dimensiunii programului, datorită codului care apare două ori.

Directiva #undef

Această directivă reprezintă opusul directivei #define. Ea şterge definiţia dată anterior unui identificator. Când un identificator nu mai este necesar, acesta se şterge şi se recuperează spaţiul de memorie pe care îl ocupă şi se evită conflictul cu alte date care au acelaşi nume. Directiva #undef este folosită pentru a permite localizarea numelor macromenzilor numai în acele zone de cod care le necesită.

Forma generală a acestei directive este prezentată în figura 2.

SINTAXĂ	#undef nume_macro
EXEMPLE	#define MAX1 20 #define MAX2 10 float a[MAX1][MAX2]; . #undef MAX1 #undef MAX2

Fig. 2.

În exemplul din figura 2 se definesc MAX1 şi MAX2, se operează cu ele la declararea tabloului bidimensional a după care din punctul in care apare directiva #undef, definiţia pentru MAX1 şi MAX2 este eliminată.

Observaţie:
Limbajul C++ permite reiniţializarea unui identificator dat printr-o macrodefiniţie prin plasarea lui într-o altă directivă #define. Nu este neapărat necesară utilizarea directivei #undef între două directive #define ale aceluiaşi identificator.

Directiva #include

Cu ajutorul acestei directive se cere compilatorului să se includă şi alte fişiere sursă în afară de cel în care se găseşte directiva. În figura 3. sunt prezentate cele două forme generale ale acestei directive de preprocesor.

SINTAXĂ	**Forma1:** #include <nume fisier> **Forma2:** #include “nume fisier“
EXEMPLE	1. #include <stdio.h> 2. #include <conio.h> 3. #include <math.h> 4. #include “nume1.c” 5. #include “k:\\user\\student\\nume2.c”

Fig. 3.

Cele două forme ale directivei #include diferă după modul în care se comandă compilatorului căutarea fişierului inclus. După etapa de preprocesare, conţinutul fişierelor incluse iau parte la compilare împreună cu programul sursă unde acestea au fost incluse.

În cazul formei 1 se cere compilatorului să caute fişierul în directoare speciale unde se află numai anumite fişiere numite fişiere standard, cum sunt cele care conţin prototipuri pentru funcţiile de bibliotecă. Directoarele se pot specifica în prealabil de către programator prin comanda Include Directories a opţiunii Directories din meniul Options. Ordinea de căutare în aceste directoare corespunde cu ordinea în care au fost ele definite. Dacă fişierul nu este găsit se primeşte un mesaj de eroare.

În cazul formei 2, căutarea se face în directorul curent sau conform "căii" specificate unde se află fişierul de inclus.
Observaţii:
1. Este indicat să se insereze toate directivele #include la începutul fişierului sursă.
2. Fişierele dintr-o directivă #include pot conţine la rândul lor alte directive #include. Standardul ANSI C stabileşte un număr minim de nivele de imbricare egal cu 8, acest număr poate diferi în funcţie de compilator.

În exemplele din figura 3. efectul directivei #include este următorul:
1., 2. şi 3. Se includ fişierele standard:
stdio.h care conţine prototipurile unor funcţii de intrare-ieşire;
conio.h care conţine prototipurile unor funcţii de intrare-ieşire;
math.h care conţine prototipurile unor funcţii matematice.

4. Se include fişierul nume1.c aflat în directorul curent.

5. Se include fişierul nume2.c aflat în directorul specificat de cale. Caracterul "\" se dublează conform definiţiei de reprezentare a caracterului într-un şir.

Operaţii de intrare-ieşire standard

Limbajul C este singurul limbaj de programare care utilizează funcţii din biblioteca standard pentru operaţiile de intrare/ieşire (I/O) şi nu defineşte cuvinte-cheie pentru efectuarea acestor operaţii. Sistemul I/O al limbajului C oferă un mecanism flexibil şi coerent de transfer al datelor între dispozitive.
Operaţiile de intrare/ieşire se împart în:
- operaţii de intrare/ieşire standard sau pentru consolă;
- operaţii de intrare/ieşire pentru fişiere.

Funcţiile pentru operaţiile de intrare/ieşire standard efectuează intrări de date de la tastatură (citire) şi ieşiri de date pe terminal (scriere sau afişare). În tabelul 1 sunt prezentate cele mai importante funcţii pentru operaţii de intrare/ieşire standard.

Tab. 1.

Funcţii pentru operaţii de intrare	Funcţii pentru operaţii de ieşire
Funcţia getch( )	Funcţia putch ( )
Funcţia getche( )	Funcţia puts( )
Funcţia gets( )	Funcţia putchar( )
Funcţia getchar( )	Funcţia printf( )
Funcţia scanf( )	

Funcţia getch( )

Cu ajutorul acestei funcţii se citeşte direct de la tastatură un caracter fără ecou pe ecran. Aceasta înseamnă că nu este afişat caracterul introdus. Prototipul acestei funcţii se află în fişierul antet <conio.h> (figura 4). Funcţia nu necesită nici un argument şi returnează un întreg care reprezintă codul ASCII al caracterului introdus.

SINTAXĂ	#include <conio.h> int getch(void);

Fig. 4.

Observaţii:

1. Funcţia se poate apela ca operand în expresii.
2. Dacă dorim să vizualizăm fereastra utilizator şi să blocăm programul pentru a analiza conţinutul curent al ecranului se utilizează funcţia getch( ). Apăsând o tastă corespunzătoare unui caracter ASCII, programul se deblochează şi apoi se continuă execuţia programului.

Funcţia getche( )

Cu ajutorul acestei funcţii se citeşte direct de la tastatură un caracter cu ecou pe ecran. Aceasta înseamnă că este afişat pe ecran caracterul introdus. Prototipul acestei funcţii se află în fişierul antet <conio.h> (figura 5). Funcţia nu necesită nici un argument şi returnează un întreg care reprezintă codul ASCII al caracterului introdus.

SINTAXĂ	#include <conio.h> int getche(void);

Fig. 5

Funcţia putch( )

Această funcţie scrie un singur caracter la consolă şi are prototipul în fişierul antet <conio.h>. La revenirea din funcţia putch( ) se returnează codul imaginii afişate (valoarea argumentului de la apel) (figura 6).

SINTXĂ	#include <conio.h> int putch(expresie);

Fig. 6

Observaţii:

1. Dacă **expresie** se află în intervalul [32,126], atunci se va afişa un caracter corespunzător codului ASCII respectiv. Dacă **expresie** este în afara acestui interval, se afişează diferite imagini care pot fi folosite în diferite situaţii.
2. Se pot afişa caractere colorate în conformitate cu culoarea curentă setată în modul text de funcţionare al ecranului.

Exemple :

1. Să se citească cu ecou un caracter imprimabil sau alb de la terminal şi apoi să se afişeze pe ecran urmat de punct.

```
#include <conio.h>
void main(void)
{ int caract;
caract=getche();
putch(caract);
putch('.');
getch();
}
```

2. Să se introducă de la tastatură pe rând trei caractere după care să se tipărească acestea pe acelaşi rând.

```
#include <stdio.h>
#include <conio.h>
void main (void)
{ int caract1, caract2, caract3;
printf(" \nIntroduceti primul caracter:");
caract1=getche();
printf(" \nIntroduceti al doilea caracter:");
caract2=getche();
putch(caract2);
printf(" \nIntroduceti al treilea caracter:");
caract3=getche();
printf("\nAti introdus caracterele:");
putch(caract1);
putch(caract2);
putch(caract3);
printf("\nApasati orice tasta pentru continuare");
getch();
}
```

3. Să se citească de la tastatură litere şi să se afişeze cu majuscule dacă sunt scrise normal şi invers.

Conversia literelor mici în mari şi invers se realizează uşor în cazul utilizării codului ASCII. Codurile ASCII ale literelor mici se află în intervalul [97, 122] şi ale literelor mari în intervalul [65,90]. Aceste coduri corespund literelor in ordine alfabetică.

Pentru a transforma o literă mică în literă mare se face transformarea: litera-32, unde litera reprezintă codul ASCII al respectivei litere. Valoarea 32 poate fi reprezentată mai sugestiv prin 'a'-'A'. Astfel codul ASCII al literei mari se obţine cu expresia litera-'a'+'A'. Pentru a face transformarea inversă din literă mare în literă mică se va utiliza expresia: litera+'a'-'A', unde litera reprezintă codul ASCII al literei mari.

```
#include <stdio.h>
#include <conio.h>
void main (void)
{
int litera;
printf(" \nIntroduceti litera:");
litera=getche();
if (litera<=122 &&litera>=97)
  putch(litera-'a'+'A');
else
  if( litera<=90 &&litera>=65)
     putch(litera+'a'-'A');
  else
  printf("\nCaracterul introdus nu e litera");
printf("\nApasati orice tasta pentru continuare");
getch();
}
```

Aplicaţii propuse

1. Realizaţi un program prin care să vă scrieţi numele de la tastatură. Afișaţi-l apoi în ordine inversă.

2. Scrieţi un program care citește de la tastatură două caractere și afișează caracterul corespunzător codului ASCII sumă a caracterelor citite. Pentru calculul sumei folosiţi directiva define.

Soluţie :

1. Pentru un nume de 14 caractere, incluzând spaţiul dintre nume și prenume:

```
#include <stdio.h>
#include <conio.h>

void main(void)

{
	char a1,a2,a3,a4,a5,a6,a7,a8,a9,a10,a11,a12,a13,a14;
	a1=getche();
	a2=getche();
	a3=getche();
	a4=getche();
	a5=getche();
	a6=getche();
	a7=getche();
	a8=getche();
	a9=getche();
	a10=getche();
	a11=getche();
	a12=getche();
	a13=getche();
	a14=getche();

	putch('\n');
	putch(a14);
	putch(a13);
	putch(a12);
	putch(a11);
	putch(a10);
	putch(a9);
	putch(a8);
	putch(a7);
	putch(a6);
	putch(a5);
	putch(a4);
	putch(a3);
	putch(a2);
	putch(a1);

	getch();

}
```

2.

```
#include <stdio.h>
#include <conio.h>

#define sum(a,b) a+b

void main(void)

{
	char a,b;
	int c;
	a=getch();
	putch('\n');
	putch(a);
```

```
        printf("%d",a);
        putch('\n');
        b=getch();
        putch('\n');
        putch(b);
        printf("%d",b);
        putch('\n');
        putch('\n');
        c=putch(sum(a,b));
        printf("%d",c);
        getch();
}
```