CURS 3.
Constructii de bază. Tipuri. Instrucţiuni C

5.1 Construcții de bază în C

Caracterele
La scrierea programelor C se foloseşte un subset al setului de caractere al codului ASCII. Caracterele din acest set se codifică în intervalul [0,127]. Un astfel de întreg poate fi păstrat în binar pe un octet (8 biți).

Se împart în trei grupe:
- caractere grafice în intervalul (32,127), aici intrând literele mari având codul ASCII în intervalul [65,96], literele mici [97,122], cifrele [48,57] şi caracterele speciale
- caractere negrafice (de control); au coduri în intervalul [0,32) cu excepția caracterului Del având codul ASCII 127.
- caracterul spaţiu având codul ASCII 32

Codul ASCII de valoare 0 defineşte caracterul nul; este un caracter impropriu, nu poate fi generat de la tastatură şi nu are efect în ieşire. Este utilizat pentru a determina un şir arbitrar de caractere deoarece nici un caracter de la intrare nu poate coincide cu el.

Codul ASCII 10 deplasează cursorul în coloana 1 din linia următoare. În C referirea la acest cod se face cu combinaţia '\n'.

Nume (identificatori)
Un *identificator* este o succesiune de litere, cifre si caracterul de subliniere (_) care începe cu o literă (mică, mare) sau cu caracterul de subliniere. Sunt semnificative primele 32 de caractere. Literele mici se consideră distincte de cele mari.

Exemplu: x, a5b9, p_i_c, _cifra
Contraex: 9a$, a+b, a%w, a_salut!

Identificatorii sunt folosiți pentru a identifica sau denumi funcții și variabile.

Cuvinte cheie
Sunt identificatori rezervaţi ai limbajului, având o semnificaţie bine determinată. Acestea nu pot fi utilizate decât conform sintaxei limbajului. Au fost definite 32 de cuvinte cheie:

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Unele compilatoare de C mai adaugă la aceste liste încă câteva cuvinte cheie.

5.2. Tipurile de bază din limbajul C

Un tip de date reprezintă mulţimea valorilor pe care le pot lua datele de tipul respectiv, modul de reprezentare a acestora în memorie precum şi operaţiile care se pot efectua cu datele respective.
Tipurile de baza definite în limbajul C sunt: întreg, real şi caracter. Unele dintre aceste tipuri diferă de la o implementare la alta a limbajului.

Un tip de date reprezintă:

- modul de reprezentare a datei respective in memoria calculatorului. Astfel, se precizează cum se convertesc biţii din memoria calculatorului pentru a produce data reprezentată de valoare cu tipul respectiv
- dimensiunea pe care o ocupă in memorie o dată de tipul respectiv
- multimea de valori admisibile pentru datele de tipul respectiv.

În tabelul următor sunt prezentate tipurile fundamentale, memoria necesară stocării valorilor de acel tip şi limita valorilor ce pot fi memorate într-o variabilă de acel tip.

Tipul	Dimensiune memorie	Limita valorilor
char	1 byte (octet)	$-2^7 \dots 2^7-1$
int	depinde de implementare (uzual 4 bytes)	
short int	2 bytes	
unsigned char	1 byte	0..255
unsigned int	depinde de implementare	
long int	4 bytes	
unsigned long int	4 bytes	
float	4 bytes	
double	8 bytes	
long double	8 bytes	

Primul tip este tipul caracter. O variabilă de acest tip va avea ca valoare codul ASCII asociat caracterului respectiv.
De exemplu, dacă unei variabile i se atribuie caracterul ’a’, variabila va conţine valoarea 97 (numărul de ordine al caracterului ‘a’ în codul ASCII).

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Tipul întreg cel mai des utilizat în programe este **int,** dar numărul de bytes pe care se memorează valorile de acest tip diferă de la o implementare la alta. Numărul de bytes al unei valori de tip int reprezintă cuvantul calculatorului respectiv. Astfel, tipul int este echivalent cu **short int** sau **long int**, in functie de implementare.

Celelalte tipuri intregi sunt obținute prin folosirea prefixului **signed** sau **unsigned**, indicând dacă valorile respective sunt cu semn (conțin și valori negative), respectiv fără semn (valori naturale).

Ultimele trei tipuri din tabel sunt tipuri reale, diferența dintre ele constând în cantitatea de memorie utilizată, intervalul de valori și precizia memorarii valorilor (numarul de zecimale retinute).

Valorile reale se reprezintă conform notației din standardul IEEE (folosind semnul, mantisa și exponentul). De exemplu, pentru tipul **float** se utilizeaza reprezentarea in simpla precizie, folosind un bit de semn, 7 biti pentru exponent si 24 de biti pentru mantisa. Aceasta reprezentare are un exponent in limita a 10^{-37} si 10^{38} cu pina la 7 zecimale precizie. Valoarea maximă a unui float este de 1.701411 E38.

Din tabel se observă ca nu există un tip logic, asa cum este el definit în alte limbaje de programare. In limbajul C nu s-a definit acest tip dar se folosește următoarea convenție: o expresie este considerată adevarată daca valoarea obtinuta la evaluarea ei este nenula si falsa in caz contrar.
Variabilele care se vor folosi ca variabile logice vor fi declarate de tip intreg

O alta caracteristica aparte o constituie introducerea tipului **void**, pentru a desemna “nimic”. Se mai folosește la declararea explicită a funcțiilor care nu au parametrii.

Pentru a putea utiliza o variabilă într-un program C, este obligatorie declararea acesteia, astfel:
TIP_DE_DATE *lista_variabile*;
unde:

- TIP_DE_DATE este orice tip predefinit sau derivat;
- *lista_variabile* conține lista variabilelor de acel tip, despărțite prin virgulă;

Exemple:
```
int a,b,c;
char ch;
float x,y;
long int z;
int g=7;
```

Observatie: *La declararea variabilei intregi g s-a realizat si initializarea acesteia (=stabilirea unei valori initiale)*

.

Variabilele care sunt declarate in interiorul corpului unei functii se numesc variabile locale, iar cele declarate in afara oricarei functii din program se numesc variabile globale. Variabilele globale vor putea fi utilizate in toate functiile ce compun programul, iar variabilele locale doar în cadrul funcției în care au fost definite.

În cadrul *tipurilor derivate* avem:
- *tipuri structurate* –tablouri, structuri, uniuni, câmpuri de biţi, enumerări
- *tipuri funcţie* –caracterizate atât prin tipul rezultatului furnizat cât şi prin numărul şi tipul argumentelor necesare pentru obţinerea rezultatului.
- *tipuri pointer* –permit adresări indirecte la entităţi de diverse tipuri

Un tip specific limbajului C este *tipul void*. Acesta se foloseşte la declararea explicită a funcţiilor care nu returnează nici o valoare sau a funcţiilor care nu au parametri.

Tipul de date `int`

Valorile acestui tip sunt numere întregi, cuprinse în intervalul [-32767,32767] reprezentate în memorie pe 2 octeți, în cod complementar. Tipul de date int suportă modificatorii de tip unsigned (datele sunt numere normale) şi long (modifică dimensiunea reprezentării). Se obțin astfel următoarele tipuri de date întregi:

Tip	Reprezentare	Valori
`int`	2 octeti cu semn	[-32768, 32767]
`unsigned int`	2 octeti fara semn	[0, 65.535]
`long int`	4 octeti cu semn	[-2.147.483.647, 2.147.483.647]
`unsigned long int`	4 octeti fara semn	[0, 4.294.967.295]

Atunci când un tip de date nu este precizat implicit este considerat int. Prin urmare putem specifica numai modificatorul `long` sau `unsigned` sau `unsigned long`, tipul fiind implicit `int`.

Constantele întregi sunt numere întregi din intervalul corespunzător tipului. Ele pot fi precizate în baza 10 folosind notația uzuală, în baza 8 constanta fiind precedată de un 0 nesemnificativ și în baza 16 constanta având prefixul 0x sau 0X.

Exemple:
123
-12345678
01234
0x1a0
0XFFFFF

Explicitarea tipului de constantă numerică se poate face utilizând un sufix. Pentru tipul întreg, utilizarea sufixului U determină alocarea tipului `unsigned`. Analog, folosim sufixul L pentru tratarea unui întreg ca `long`. Dacă utilizăm sufixul F pentru tipul virgulă mobilă, numărul va fi tratat ca float iar dacă folosim sufixul L numărul va fi de tip `long double`.

Exemple:

100U	unsigned
-1234L	long int
123.45F	float
123456.789	long double

Tipul de date `char`

Tipul caracter în C se reprezintă pe un octet şi are ca valoare codul ASCII al caracterului respectiv. Constanta caracter grafic se scrie incluzând caracterul respectiv între caracterul apostrof.

Exemplu: 'A'
→ intern vom avea o reprezentare pe un octet ce conţine valoarea 65 (codul ASCII al literei A).

Tip	Reprezentare	Valori
`char`	1 octet cu semn	[-128, 127]
`unsigned char`	1 octet fara semn	[0, 255]

Contsantele de tip char (unsigned char) pot fi numere întregi din intervalul specificat care au codurile ASCII în intervalul specificat. Valorile de tip char par a avea o natură duală, caractere ASCII şi numere întregi. Caracterul A şi codul 65 au aceasi semnificaţie.
Caracterele grafice (coduri ASCII de la 32 la 127) se pot specifica încadrând caracterul respectiv între apostrofuri.
Exemplu:
'a', '9', '*'

Caracterele negrafice se pot specifica încadrând între apostrofuri o secvență de evitare (secvență escape). Secvențele escape sunt formate din caracterul \ (backslash) urmat de codul ASCII al caracterului exprimat în baza 8 sau în baza 16 precedat de un x.

Exemplu:

Secvență escape	Caracter
'\65'	'5'
'\x35'	'5' (exprimat in baza 16)
'\5'	Caracterul ♣
'\356'	Caracterul €

Unele caractere negrafice au asociate secvente escape speciale:

Secvenţa	Semnificaţia
\a	allert (bell)
\b	backspace
\f	form feed
\n	new line
\r	carriage return
\t	horizontal tab
\v	vertical tab

Constantele şir de caractere sunt constituite dintr-o succesiune de caractere încadrată între ghilimele.
Exemple:
"Acesta este un şir"
"Prima linie \n A doua linie"

Tipuri reale

Tipurile reale sunt float şi double. Tipul double acceptă şi modificatorul long.

Tipul	Reprezentare
float	4 octeti in virgulă mobilă
double	8 octeti in virgulă mobilă
long double	10 octeti in virgulă mobilă

Un număr flotant este un număr raţional care se compune din semn (dacă e cazul), o parte întreagă (care poate fi vidă), o parte fracţionară (care şi ea poate fi vidă) litera e sau E şi un exponent (care şi ele pot lipsi). Nu pot fi vide toate componentele. Partea întreagă este o succesiune de cifre hexagesimale. Partea fracţionară se compune dintr-un punct urmat de o succesiune de cifre zecimale, succesiune care poate fi vidă.
Se reprezintă în forma uzuală sau ştiinţifică ca exponent a lui 10, urmat de e.

Exemple:

-1
5e-5
2.e-4
12.4
-2.67e-10
-2.67E+8

5.3. Instructiuni

Instrucţiunile C implementează structurile de bază ale programării structurate, prezentate în cursul 1 şi identificabile in schemele logice şi pseudocod astfel:

-*structura secvenţială* (instrucţiunea compusă) ;
-*structura alternativă* (instrucţiunea if) ;
-*structura repetitivă* condiţionată anterior (instrucţiunea `while`) şi condiţionată posterior (instrucţiunea `do while`)

Toate instrucţiunile limbajului C se termină cu ';' excepţie făcând instrucţiunile care se termină cu } (instrucţiunea compusă şi instrucţiunea switch) după care nu se pune ';'.

Instrucţiunea vidă

Instrucțiunea vidă are formatul:

;

şi este folosită pentru a înlocui o instrucțiune care practic nu este, dar contextul (sintaxa) o cere.

Una din situații o constituie instrucțiunile de ciclare `while` şi `for` a căror sintaxă impune corp al instrucțiunii. Dacă acesta lipsește, atunci instrucțiunea vidă va ține loc de corp al instrucțiunii.

Exemplu1

```
. . . . . .
for (i=0;(i+1)*(i+1)<n;i++);
. . . . . .
```

Această instrucțiune de ciclare determină cel mai mare număr al cărui pătrat nu este mai mare decât numărul natural n adică [√n] (partea întreagă din radical din n). Se observă că tot ceea ce este de făcut

se face în linia instrucțiunii `for`, dar sintaxa lui `for` impune corp al instrucțiunii și în locul acestuia se pune instrucțiunea vidă ;.

Instrucţiunea compusă

Forma generală:

```
{
  declaratii_si_definitii;

  instructiune1;
  instructiune2;
…
  instructiune n;
}
```

Se foloseşte în situaţiile în care sintaxa impune o singură instrucţiune, dar codificarea impune prezenţa unei secvenţe de instrucţiuni. Blocul de instrucţiuni contează ca o singură instrucţiune.

Observaţii:
1. După acoladă închisă nu se pune ;.
2. Corpul unei funcţii are aceeaşi structură ca şi instrucţiunea compusă.
3. O instrucţiune compusă permite folosirea mai multor instrucţiuni acolo unde sintaxa cere o instrucţiune, aceasta fiind echivalentă sintactic cu o singură instrucţiune.
4. instrucțiunea compusă reprezintă un domeniu de vizibilitate. Astfel, declarațiile și definițiile din interiorul unei instrucțiuni compuse au valabilitate doar in interiorul acesteia.

Instrucţiunea if

În limbajul C, instrucţiunea condiţională de bază este instrucţiunea `if`.

Forma generală:

```
if (expresie)
  instructiune1;
else
  instructiune2;
```

Se evaluează expresia; dacă este diferită de 0 se execută `instrucţiune1` altfel `instrucţiune2`

O formă simplificată are instrucţiune2 vidă:

```
if (expresie)
  instructiune;
```

În problemele de clasificare se întâlnesc decizii de forma:

```
if (expr1)
  instr1;
else if (expr2)
  instr2;
…
else
  instrn;
```

Exemplu2: dorim să contorizăm caracterele citite pe categorii: litere mari, litere mici, cifre, linii şi altele:

```
if  (c == '\n')
  linii++;
```

```
else if (c>='a' &&  c<='z')
  lmici++;
else if (c>='A' && c<='Z')
  lmari++;
else if (c>='0' && c<='9')
  cifre++;
else
  altele++;
```

Observaţii: 1. Ramura `else` poate lipsi.
2. Întotdeauna una şi numai una dintre instrucţiuni se execută adică instrucțiunea de pe ramura if sau cea de pe else.
3. Este posibilă utilizarea mai multor instrucţiuni `if` imbricate.
4. Deoarece există posibilitatea prezenţei instrucţiunii `if` fără ramura `else`, rezultă că este necesară următoarea regulă: un `else` se pune în corespondenţă cu primul `if` care se află înaintea lui în textul sursă şi care nu are asociat un alt `else`.

Exemplu3:

```
if(x)
  if(y) printf("1");
  else printf("2");
```

În acest caz, ramura else care tipăreşte valoarea 2 este legată de instrucţiunea `if(y)`.

```
if(x)
  {if (y) printf("1"); }
else printf("2");
```

În acest caz, datorită instrucţiunii compuse, ramura else care tipăreşte valoarea 2 aparţine instrucţiunii `if(x)`.

Instrucţiunea while

Instrucţiunea while implementează structura repetitivă condiţionată anterior. Forma instrucțiunii while este:

```
while (expresie)
       instructiune;
```

Atât timp cât *expresie* este diferită de 0 se execută *instructiune*. Instrucțiunea *while* mai este cunoscută şi sub numele de ciclare cu test inițial. Bucla *while* este folosită atunci când numărul de iterații nu este cunoscut apriori.
Observații.
1.Instrucțiunea se repetă cât timp valoarea expresiei nu este nulă. Pentru ca ciclul să nu fie infinit, este obligatoriu ca instrucțiunea care se execută să modifice cel puțin una din variabilele care intervin în expresie, astfel încât aceasta să poată lua valoarea 0, sau să conțină o operație de ieșire necondiționată din ciclu (de exemplu `break`).
2.Dacă de la început expresia are valoarea 0, instrucțiunea nu se execută nici măcar o dată.
3.Sintaxa permite executarea în while a unei singure instrucțiuni, prin urmare, atunci când este necesare efectuarea mai multor operații , acestea se grupează într-o singură instrucțiune compusă.

Exemplu5: Programul care numără caracterele unui text. Introducerea se încheie cu *enter*.

```
#include <stdio.h>

int main ()
{
```

```
  int i=0;
  printf("Scrieti un text : \n");
  while (getch()!='\r')  /*se citeste textul fara afisare in consola */
     i++;
  printf("Numarul de caractere este %d\n",i);
  return 0;
}
```

Exemplu6: Program care calculează factorialul:

```
#include <stdio.h>
int main()
{
  long n, fact;
  printf("n=");  scanf("%ld", &n);
  fact=1;
  while (n>1)
  {
       fact = fact * n;
       n--;
  }
  printf("Factorialul este : %ld\n", fact);
}
```

Instrucţiunea do while

Instrucţiunea do while implementează structura repetitivă condiţionată posterior.

Forma instructiunii do while este :

```
do
      instructiune;
while (expresie);
```

Instrucțiunea este asemănătoare cu instructiunea *while* cu deosebirea că, aici, testul se face la sfârșitul buclei. Bucla se va executa cel puțin o dată, chiar dacă testul nu este îndeplinit de la început. Instrucțiunea se execută cât timp expresia este diferită de 0.

Observaţie: Spre deosebire de instrucţiunea while, în cazul acestei instrucţiuni, corpul ei se execută cel puţin o dată, indiferent de valoarea iniţială de adevăr a expresiei.
Preferăm să utilizăm instrucțiunea while când este necesar să testăm o condiție înainte de efectuarea unor prelucrări.
Preferăm să utilizăm `do - while` când condiția depinde de la început de prelucrările din ciclu, prin urmare este necesar să o testăm după executarea intrucțiunii.

Exemplu8: Să se numere cifrele numărului natural memorat în variabila n. Corpul buclei se scrie ca mai jos.

```
#include<stdio.h>
int main()
{
     int n, nr;
     printf("introduceti un numar intreg: "); scanf("%d", &n);
     nr=0;
     do
     {
          n/=10;
```

```
            nr++;
      } while(n);
      printf("numarul de cifre este %d\n",  nr);
      return 0;
}
```

Instrucţiunea for

În cazul în care se cunoaşte numărul de iteraţii ale unui ciclu repetitiv, se poate uşor utiliza instrucţiunea for. *Formatul general* al acestei instrucţiuni este următorul:

```
for(exp1;exp2;exp3) instrucţiune;
```

Efectul acestei instrucţiuni este următorul: se evaluează exp1 după care se testează valoarea de adevăr a exp2. Dacă exp2 are valoarea 0, se termină execuţia instrucţiunii for, şi se continuă cu următoarea instrucţiune de după instrucţiunea for. Dacă exp2 are valoarea diferită de 0, se execută instrucţiune după care se evaluează exp3. Se revine la evaluarea lui exp2 şi se continuă ciclic până când exp2 va deveni, caz în care se continuă cu următoarea instrucţiune de după instrucţiunea for.

Instrucţiunea for este echivalentă cu următoarea secvenţă de instrucţiuni:

```
exp1;
while(exp2)
{
   instrucţiune;
   exp3;
}
```

Observaţie: Oricare din cele 3 expresii pot să lipsească. De asemenea instrucţiunea poate lipsi. Dacă lipsesc toate cele 3 expresii se obţine un ciclu infinit:

```
     for(;;) instrucţiune;
```

Exemplu9 :

```
#include <stdio.h>
int main()
{
       int  i;
       for( i = 1; i <= 10; i++ )
             printf("%d ", i );
       printf("\n");
       return 0;
}
```

Exemplu10: Să se scrie programul C care calculează n! unde ştim că n!=n*(n-1)! iar 0!=1.

```
#include<stdio.h>
int main()
{
  int n=6, fact=1, i;
  for(i=1;i<=n;i++) fact*=i;
  printf("%d! = %d\n", n, fact);
  return 0;
}
```

Instrucţiunea return

Instrucţiunea return este folosită pentru a returna rezultatul dintr-o funcţie.

Este considerată drept instrucţiune de salt întrucât determină executarea programului să revină (să execute un salt înapoi) la punctul în care s-a făcut apelarea funcţiei.
Dacă instrucţiunii return îi este asociată o valoare, acea valoare devine valoarea calculată prin acea funcţie. Dacă nu este specificată nici o valoare calculată a funcţiei, se presupune returnarea unei valori inutile (unele compilatoare C vor returna valoarea zero dacă nu este indicată nici o valoare).

Forma generală a instrucţiunii return permite 2 formate

```
return;  // este necesara daca tipul de return al functiei este void
return expresie;
```

Primul format se utilizeaza cand functia nu returneaza o valoare iar cel de-al doilea se utilizeaza atunci cand functia returneaza o valoare (functia returneaza valoarea expresiei specificate).

În cadrul unei funcţii putem folosi instrucţiuni return la discreţie. Cu toate acestea, o funcţie va fi stopată din execuţie la întâlnirea primei instrucţiuni return.
Acolada de la sfârşitul unei funcţii determină de asemenea revenirea din aceasta. Situaţia este analogă cu un return căruia nu i s-a precizat nici o valoare.

O funcţie declarată de tip void nu poate conţine o instrucţiune return care specifică o valoare. Din moment ce o funcţie de tip void nu are valoare calculată, este de bun simţ faptul că o instrucţiune return interioară acesteia nu poate returna o valoare

Tipul valorii returnate va fi cel specificat la tipul funcţiei care returnează valoarea (tipul int în cazul în care nu s-a specificat un tip pentru această funcţie). Funcţia apelantă poate ignora valoarea returnată.

O funcţie care nu returnează nimic se poate încheia şi fără prezenţa instrucţiunii return. Astfel, ea se încheie după executia ultimei instrucţiuni şi întâlnirea acoladei de sfârşit a funcţiei ‘}’.

Exemplu 12: Să se scrie o funcţie care returnează maximul a 2 numere primite ca parametru.

```
int max(int x, int y)
{
  if(x>y) return x;
  else return y;
}
```

Instrucţiunea switch

Permite realizarea unei structuri selective. Ea este o generalizare a instrucţiunii if care poate fi realizata prin structuri alternative imbricate.
Această instrucţiune determină transferul controlului unei instrucţiuni sau unui bloc de instrucţiuni în funcţie de valoarea unei expresii şi are următorul *format general*:

```
switch (expresie) {
  case c1: instr1;
  [case c2: instr2;]
 …
  [default: instr_default;]
}
```

Efectul acestei instrucţiuni este următorul: se evaluează expresia dintre paranteze. Dacă valoarea expresiei este egală cu c_i atunci se execută instrucţiunea corespunzătoare constantei c_i. Dacă valoarea expresiei este diferită de orice c_i indicat, atunci se execută instrucţiunea corespunzătoare clauzei default, dacă există această clauză după care se trece la următoarea instrucţiune de după instrucţiunea switch.

După ce se execută instrucţiunea corespunzătoare unei constante indicate, dacă nu există instrucţiunea break atunci necondiţionat se execută toate instrucţiunile de mai jos celei corespunzătoare constantei c_i, eventual până la prima instrucţiune break întâlnită.
Instrucţiunea switch este deseori folosită pentru a prelucra anumite comenzi de la tastatură, cum ar fi selecţia unei opţiuni dintr-un meniu.

Observaţii:

1. Valoarea expresiei dintre paranteze trebuie să fie de un tip compatibil cu int (char, enum sau orice variantă de int). Nu se pot utiliza numere reale, şiruri, pointeri sau structuri dar se pot folosi elemente de tip compatibil cu int din cadrul şirurilor sau structurilor. Este interzisă apariţia a două constante pentru case în aceeaşi instrucţiune switch cu aceeaşi valoare.
2. Standardul ANSI C prevede că instrucţiunea switch poate să aibă cel puţin 257 de instrucţiuni tip case. Există posibilitatea de a imbrica mai multe instrucţiuni switch una în alta, fără să apară conflicte chiar dacă unele constante case dintr-un switch interior şi unul exterior conţin valori comune.

Exemplu 20:

```
…
switch (luna) {
// februarie
case 2: zile=28; break;
// aprilie, iunie,..., noiembrie
case 4: case 6: case 9: case 11: zile =30; break;
// ianuarie, martie, mai,.. decembrie
default: zile=31; break;
}
…
```

Instrucţiunea break

Instrucţiunea break este utilizată pentru ieşirea forţată dintr-o buclă, înainte de sfârşitul acesteia. În general, aceste instrucţiuni de ieşire forţată se aplică unor algoritmi nestructuraţi. Se recomandă evitarea acestora, ştiind fiind faptul că orice algoritm nestructurat poate fi transformat într-un algoritm structurat.
Dacă este necesar, corpul unei instrucţiuni de ciclare poate să conţină mai multe instrucţiuni break, corespunzătoare mai multor condiţii de ieşire forţată. Bineînţeles, pot exista şi ieşiri normale din aceste bucle, chiar dacă au fost prevăzute mai multe instrucţiuni break.
Instrucțiunea permite ieșirea dintr-un singur ciclu, nu și din eventualele cicluri care ar conține instrucțiunea repetitivă în care s-a executat instrucțiunea `break`.

Exemplu15:

```
#include <stdio.h>
#define PROMPT ':'

int main()
{
     float a, b, result;
     char oper, eroare;
     while (putchar(PROMPT),(scanf("%f%c%f",&a, &oper, &b)!=EOF)){
           eroare=0;
           switch(oper){
                  case '+': result=a+b; break;
                  case '-': result=a-b; break;
                  case '*': result=a*b; break;
                  case '/': if(b)result=a/b;
                            else
```

```
                                        {
                                    puts("*** Impartire la 0 ***");
                                    eroare=1;
                                }
                                break;
                    default : printf("*** Operator ilegal %c ***\n", oper);
                                eroare=1;
            }//switch
            if(!eroare)
                    printf("rezultatul e %f\n", result);
      }//while
}//main
```

Exemplu16:

```
char c; int a, b, r;
printf("Scrieti o operatie intre doi intregi: ");
if (scanf("%d %c %d", &a, &c, &b) == 3) { /* toate 3 corect */
     switch (c) {
          case '+': r = a + b; break; //iese din corpul switch
          case '-': r = a - b; break; // idem
          default: c = '\0'; break; // fanion caracter eronat
          case 'x': c = '*'; //'x' e tot ?nmultire, continua
          case '*': r = a * b; break; //ca si pt. apoi iese
          case '/': r = a / b; //la sfarsit nu trebuie break
     }
     if (c)
          printf("Rezultatul: %d %c %d = %d\n", a, c, b, r);
     else
          printf("Operatie necunoscuta\n");
     }
else printf("Format eronat\n");
```

Instrucţiunea continue

Se utilizează în interiorul ciclurilor şi permite saltul la începutul secvenţei de instrucţiuni care formează corpul ciclului respectiv continuând cu următoarea iteraţie a ciclului, deci nu se părăseşte bucla. Ea se aseamănă cu instrucţiunea break, dar în loc să forţeze încheierea buclării, instrucţiunea continue forţează trecerea la următoarea iteraţie a buclei.
Pentru bucla for, instrucţiunea continue determină execuţia secvenţei de incrementare şi a testului de condiţionare, iar pentru buclele while şi do-while controlul programului este trecut testului de condiţionare.

Exemplu17:

```
while (1)
{
scanf("%lf", &x);
if (x < 0.0)
 break; //iesim din while cand x este negativ
 printf("%lf\n", sqrt(x));
}
while (contor < n)
{
   scanf("%lf", &x);
   if (x > -0.01 && x < =0.01)
     continue; //valorile mici nu se iau in considerare
   ++contor;
```

```
    suma += x;
}
```

5.12.Instrucţiunea goto

Instrucţiunea de salt goto permite saltul la o anumită instrucţiune din cadrul funcţiei, instrucţiune precedată de o etichetă. Formal, instrucţiunea goto nu este necesară niciodată, în cazul programării structurate. Nu se recomanda utilizarea acestei instructiuni; programarea care utilizeaza salturi neconditionate se numeste programare in stil "spaghetti".

Formatul instrucţiunii goto este:

```
goto etichetă;
```

unde etichetă este un nume care identifică, sub forma:

```
etichetă: instrucţiune;
```

instrucţiunea cu care se continuă execuţia programului.
Aceeaşi etichetă poate fi referită de mai multe instrucţiuni goto, dar o etichetă nu poate identifica decât o singură instrucţiune. Unicul scop pentru care se defineşte o etichetă este de a fi referită de o instrucţiune goto.