

Detalii de implementare.
Declararea variabilelor,
transmiterea parametrilor
catre subprograme.

Declararea variabilelor

- variabile globale -declarate in afara oricarei functii
- variabile locale -declarate intr-un bloc de instructiuni (ex: in corpul unei functii)
- variabile locale statice -precedate de static

- **variabile globale**
 - vizibile in orice functie
 - initializate implicit cu 0
- **variabile locale**
 - vizibile doar in bloc(existente doar in bloc)
 - NU sunt initializate automat
- **variabile locale statice**
 - durata de existenta -tot programul
 - vizibile doar in apelul functiei
 - initializarea - implicit cu 0, la primul apel, pentru oricate apeluri ale functiei
 - la un nou apel al functiei, variabila are valoarea din ultimul apel al functiei

```
int vg;                //initializata cu 0-implicit
```

```
void f()
```

```
{int vl; cout<<vl;    //valoare nedeterminata
```

```
static int vs1;        //initializata cu 0-implicit
```

```
static int vs2=0;
```

```
vs1++;vs2++;
```

```
cout<<vs1<<vs2;
```

```
}
```

```
int main()
```

```
{cout<<vg;
```

```
f();                //vs1=1;vs2=1;
```

```
f();                //vs1=2;vs2=2;
```

```
}
```

Tipuri de date complexe

```
int i;  
int *p;          //adresa de intreg  
int &r=i; //alt nume pentru variabila i (initializare obligatorie)
```

```
int main()  
{ p=&i;          // adresa variabilei i  
  *p=1;          //zona adresata de p  
  r=1;           // r=i=1  
  p=&r;           // adresa zonei referite =adresa lui i  
}
```

Declaratori complecsi

prioritate()=prioritate[]>prioritate*

int v[10]; // vector de intregi

int **aa; // adresa unei adrese de intreg

int * va[10]; // vector de adrese de intregi

int * fp(); //functie care intoarce adresa unui intreg

int (*pf)(); //adresa unei functii ce intoarce un intreg

Alocare dinamica

```
int *p;
```

```
p=new int; /* o zona int e ocupata -intoarce adresa de  
           inceput a zonei*/
```

```
*p=1; // accesarea zonei ocupate
```

```
p=new int(1); // alocare si initializeaza zona
```

```
p=new int[2]; /* ocupa o zona de 2 intregi si intoarce adresa  
              de inceput a zonei */
```

Eliberare zona dinamica

delete p; /* zona ocupata e considerata libera -> poate fi
realocata altei variabile*/

delete [] p; /* zona continua de mai multe elemente va fi
considerata neocupata*/

Obs.1. pot elibera doar zone alocate dinamic

Obs.2. Orice zona alocata dinamic ramane alocata pana
este apelat operatorul delete

Pointeri si referinte care nu modifica zona

```
int a,b;
```

```
const int * p; //pointer care nu modifica zona  
p=&a;          //poate primi valori dupa declarare  
a=0;          // variabila se poate modifica  
/*p=0;        // nu poate modifica variabila adresata
```

```
int * const pc=&a; // pointer constant- trebuie initializat  
*pc=0;           // poate modifica zona  
//pc=&b;         // nu poate adresa ALTA zona
```

```
const int &rc=a;      // trebuie initializata
    a=0;              // variabila se poate modifica
    //rc=0;           // rc nu poate modifica variabila referita
```

```
int &r=a;    /*orice referinta este constanta -
            nu poate fi un alt nume pentru ALTA variabila*/
    r=b;    /*copiază valoarea lui b in zona comuna a lui r si a*/
```

Transfer parametrilor

- prin valoare -parametrul formal este o COPIE -pe stiva a parametrului actual (modificarile din functie NU modifica parametrul actual); pot transmite o adresa prin valoare
- prin referinta-parametrul formal este un alt nume al parametrului actual (modificarile din functie se fac in zona de memorie a parametrului actual) ; parametrul actual e obligatoriu o zona de memorie

```
void f(int i, int*p, int &r)
{ i++; // se modifica doar copia
  (*p)++; // se modifica variabila de la adresa p
          /* transmite tot prin valoare -a adresei*/
  r++; // se modifica parametrul actual
}
```

```
int main()
{int x,y,z;
  x=y=z=0;
  f(x,&y,z);
}
```

```
int f1(){return 2;}    // se intoarce o copie
```

```
int * f2(int i, int *p, int &r)    //intoarce o adresa
```

```
{int l;
```

```
// return &l; return &i;    /* eroare adresa zona de pe stiva*/
```

```
return p; return &r;
```

```
int *pl=new int; return pl;}
```

```
int & f2(int i, int *p, int &r)    /*intoarce alt nume pt un int */
```

```
{int l;
```

```
//return l; return i;    /* eroare alt nume pt zona de pe stiva*/
```

```
return *p; return r;
```

```
int *pl=new int; return *pl;}
```

Intoarcere rezultate prin parametrii

```
void fv(int copie){cin>>copie;} // citirea se face in copie  
void fr(int &rez){cin>>rez;}
```

```
int main()  
{int v=0;  
fv(v); // copie =0; citeste in copie; v ramane 0  
fr(v); /* rez va fi alt nume pentru v -citirea se va face in v*/  
}
```

```
void f1(int * & rezP)
{rezP=new int;}
```

```
int main()
{int *p=NULL;
f1(p);          /* rezP e alt nume pt variabila p
                 adresa zonei noi se va pune in p */
}
```

```
void f2(int **adRezP)
{ *adRezP=new int; }
```

```
int main()
{ int *p=NULL;
  f2(&p);      /* in zona adresata adica in variabila p se
                pune adresa zonei noi */
}
```


Diferente alocare statica /dinamica

`int v[10];`

- v=adresa primului element din vector
- v=pointer constant (nu va adresa alta zona)
- are zona alocata de dimensiune fixa
- sizeof(v) este 10×4

`int *p;`

- trebuie alocata zona ex: `p=new int[10];`
- pointer variabil ex: `p=v`
- dimensiune variabila a zonei ex: `p=new int[n];`
- sizeof(p) este 4

Vectori

```
int v[10];
```

```
// v e adresa primului element
```

```
//v+1 e adresa urmatorului element (int)
```

```
v[0] este *v;
```

```
v[1] este *(v+1);
```

```
v[i] este *(v+i);
```

Transmitere vectori

```
void f1(int *v)    {v[0]=0;};    /*v=0
```

```
void f2(int v[10]) {v[0]=0;};
```

```
void f3(int v[ ])  {v[0]=0;};
```

```
// transmite (echivalent) adresa primului element
```

```
int main()
```

```
{int a[10];
```

```
f1(a); f2(a); f3(a);
```

```
}
```

```
/* modifica valoarea de la adresa v (elementele din a) */
```

Matrici

```
int m[5][10];
```

```
/* vector cu 5 elemente de tip vector de 10 int ;  
   e zona continua de 50 int */
```

m e adresa primului vector de 10 int

m+1 e adresa vector 2 de 10 int

m+i e adresa vector i de 10 int

*(m+i) e vectorul i = adresa primului int din vector i

*(m+i) + j = adresa elem j din vectorul i

m[i][j] elem de pe linia i coloana j este $*(*(m+i)+j)$

```
int *m[5];    //vector cu 5 elemente de tip pointer la int;  
/*zonele nu sunt continue -trebuie alocate*/  
for(int i=0;i<lin;i++) m[i]= new int [col];  
/* vector  de  adrese de vectori de int */
```

m adresa primului pointer

m+1 e adresa urmatorului pointer

m+i e adresa pointer i

*(m+i) e adresa primului int din vectorul i

*(m+i)+j e adresa elem j din vectorul i

m[i][j] este (*(m+i)+j)

```
int ** m;
```

```
/*adresa unei adrese de int -nu are alocat spatiu pt  
elemente*/
```

```
m=new int *[lin];    //vector de adrese de int
```

```
for (int i=0;i<lin;i++) m[i]=new int[col];
```

```
/* vector de adrese de vectori de int */
```

```
m[i][j] este *(*m+i)+j)
```

Transmitere matrici

- trebuie specificata ultima dimensiune

($m+1$ =adresa urmatorului vector de 10 char = $m+10$ octeti)

```
void f4(char m[5][10]){}  
void f5(char m[][10]){}  
  
-nu trebuie specificata ultima dimensiune  
( $m+1$ = adresa urmatorului pointer la char = $m+4$  octeti)  
void f6(char *m[5]){}  
void f7(char **m){}
```

Sir sau vector de caractere

```
char v[10];
```

```
//vector de caractere
```

```
for(int i=0;i<10;i++) cin>>v[i];
```

```
//sir de caractere (terminat cu caracterul '\0')
```

```
cin>>v;
```

```
//ocupa un octet in plus -pt '\0'
```


Citire siruri de caractere

`cin>>v; // se opreste la primul caracter alb`

`cin.get(v,nr,c); /* citeste cel mult nr-1 caractere sau a fost
intalnit delimitatorul c (implicit '\n') */`

`cin.get(); /* mai trebuie citit un caracter (delimitatorul) */`

-biblioteca string -functii ce lucreaza cu siruri sfarsite cu delimitator :

strlen(sir), strcpy(dest,sursa), strcmp(sir1,sir2)

-copiere

```
char dest[10],sursa[10];
```

```
//dest=sursa; /* ar schimba zona adresata-nu face copierea*/
```

```
strcpy(dest,sursa);
```

-comparare:

```
char s1[10],s2[10];
```

```
//s1<s2; // compara adresele de inceput ale vectorilor
```

```
strcmp(s1,s2); // compara lexicografic sirurile (<0),0,(>0)
```

Structuri

```
struct data
```

```
{int zi,luna,an;
```

```
};
```

```
data azi,ieri,*pd; //data este un tip de date
```

```
    pd=&azi;    // *pd este zona adresata adica azi
```

```
    azi.zi=17;
```

```
(*pd).zi=17; /*echivalent cu*/
```

```
pd->zi=17;
```

```
azi=ieri;      /* copiaza bit cu bit informatia
```

```
eroare in cazul campurilor alocate dinamic*/
```

Structuri recursive

```
struct nod  
{int inf;  
  nod * a_urm;  
};  
nod el,*p;
```

```
  p=&el;      // *p este zona adresata adica el  
  el.inf=0;  
(*p).inf=0;      // echivalent cu  
p->inf=0;
```

```
p= new nod ; /* alocă zona pentru un nod  
                și întoarce adresa nodului*/  
// *p este zona adresată = nodul alocat
```

```
(*p).el =0; // echivalent cu  
p->el=0;
```

```
delete p; // consideră nodul ca fiind zona liberă
```

Instructioni

```
if (cond1) instr1;  
if (cond2) instr2;  
else instr2;
```

/* else se leaga de cel mai apropiat if
cod echivalent cu : */

```
if (cond1) instr1;  
{if (cond2) instr2;  
else instr2;}
```

```
int x=0;  
cout<<x++; // se afiseaza 0 si x devine 1  
int y=0;  
cout << ++y; // y devine 1 si se afiseaza 1
```

== difera de =

```
int i=0,j=0;  
if(i=1) instr1; /* i primeste val 1 , rezultatul atribuirii este 1  
                (adevarat) deci executa instr1*/  
if(j==1)instr1; /* conditie falsa -nu se executa instr1*/
```

break- iese din switch sau ciclare (for, while, do while)

```
/* daca switch nu are break -executa la rand instructiunile  
   pana la sfarsit/break;*/
```

```
int i = 1;
```

```
switch (i) { case 1:cout<<"1";  
             case 2: cout << "2"; break  
             case 3: cout<<"3";  
           } // afiseaza 1 2
```


continue - sare restul instructiunii de ciclare

```
while (/* ... */)
```

```
{ // ...
```

```
    continue; // echivalent cu goto contin;
```

```
// ...
```

```
    contin::;
```

```
}
```

```
do { // ...  
  continue; //echivalent cu goto contin;  
  // ...  
contin::  
} while (/* ... */);
```

```
for (/* ... */) {  
  // ...  
  continue; // echivalent cu goto contin;  
  // ...  
contin::  
}
```

Tipuri de date standard

Modificatori de tip

•signed

- implicit pentru toate tipurile de date;
- bitul cel mai semnificativ este bitul de semn;

•unsigned

- doar valori pozitive;
- bitul de semn participă la reprezentarea valorilor -> domeniul de valori mai mare;

•short

- reduce dimensiunea tipului de date int la jumătate;
- se aplică doar pe int;

•long

- se aplică doar pe int sau double;
- int – la compilatoarele noi: **int = long int = 4octeti;**

•long long

- se aplica doar pe int.

Tip de date	Memorie	Domeniu de valori	Observații
char	≥ 1 bytes		<ul style="list-style-type: none"> - poate fi signed/unsigned în funcție de compilator/sistem de operare/arhitectură; - folosit când vrem să memorăm caractere.
signed char		Cel puțin [-128,+127] ($[2^7, 2^7-1]$)	- folosit când vrem să stocăm un întreg.
unsigned char		Cel puțin [0, 255]	

dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch	dec	oct	hex	ch
0	0	00	NUL (null)	32	40	20	(space)	64	100	40	@	96	140	60	`
1	1	01	SOH (start of header)	33	41	21	!	65	101	41	A	97	141	61	a
2	2	02	STX (start of text)	34	42	22	"	66	102	42	B	98	142	62	b
3	3	03	ETX (end of text)	35	43	23	#	67	103	43	C	99	143	63	c
4	4	04	EOT (end of transmission)	36	44	24	\$	68	104	44	D	100	144	64	d
5	5	05	ENQ (enquiry)	37	45	25	%	69	105	45	E	101	145	65	e
6	6	06	ACK (acknowledge)	38	46	26	&	70	106	46	F	102	146	66	f
7	7	07	BEL (bell)	39	47	27	'	71	107	47	G	103	147	67	g
8	10	08	BS (backspace)	40	50	28	(72	110	48	H	104	150	68	h
9	11	09	HT (horizontal tab)	41	51	29)	73	111	49	I	105	151	69	i
10	12	0a	LF (line feed - new line)	42	52	2a	*	74	112	4a	J	106	152	6a	j
11	13	0b	VT (vertical tab)	43	53	2b	+	75	113	4b	K	107	153	6b	k
12	14	0c	FF (form feed - new page)	44	54	2c	,	76	114	4c	L	108	154	6c	l
13	15	0d	CR (carriage return)	45	55	2d	-	77	115	4d	M	109	155	6d	m
14	16	0e	SO (shift out)	46	56	2e	.	78	116	4e	N	110	156	6e	n
15	17	0f	SI (shift in)	47	57	2f	/	79	117	4f	O	111	157	6f	o
16	20	10	DLE (data link escape)	48	60	30	0	80	120	50	P	112	160	70	p
17	21	11	DC1 (device control 1)	49	61	31	1	81	121	51	Q	113	161	71	q
18	22	12	DC2 (device control 2)	50	62	32	2	82	122	52	R	114	162	72	r
19	23	13	DC3 (device control 3)	51	63	33	3	83	123	53	S	115	163	73	s
20	24	14	DC4 (device control 4)	52	64	34	4	84	124	54	T	116	164	74	t
21	25	15	NAK (negative acknowledge)	53	65	35	5	85	125	55	U	117	165	75	u
22	26	16	SYN (synchronous idle)	54	66	36	6	86	126	56	V	118	166	76	v
23	27	17	ETB (end of transmission block)	55	67	37	7	87	127	57	W	119	167	77	w
24	30	18	CAN (cancel)	56	70	38	8	88	130	58	X	120	170	78	x
25	31	19	EM (end of medium)	57	71	39	9	89	131	59	Y	121	171	79	y
26	32	1a	SUB (substitute)	58	72	3a	:	90	132	5a	Z	122	172	7a	z
27	33	1b	ESC (escape)	59	73	3b	;	91	133	5b	[123	173	7b	{
28	34	1c	FS (file separator)	60	74	3c	<	92	134	5c	\	124	174	7c	
29	35	1d	GS (group separator)	61	75	3d	=	93	135	5d]	125	175	7d	}
30	36	1e	RS (record separator)	62	76	3e	>	94	136	5e	^	126	176	7e	~
31	37	1f	US (unit separator)	63	77	3f	?	95	137	5f	_	127	177	7f	DEL (delete)

short short int signed short signed short int	≥ 2 bytes	Cel puțin [-32768, +32767] (mai rar [-32767, +32767])	
unsigned short unsigned short int		Cel puțin [0, +65535]	- short int pozitiv
int signed signed int		Cel puțin [-32768, +32767]	- deseori echivalent cu long (long int), implicit domeniul de valori este [-2147483648,+2147483647]
unsigned unsigned int		Cel puțin [0, +65535]	- int pozitiv - deseori echivalent cu unsigned long (unsigned long int), implicit domeniul de valori este [0,+4294967295]

long long int signed long signed long int	≥ 4 bytes	Cel puțin [-2147483648, +2147483647]	
unsigned long unsigned long int		Cel puțin [0,+4294967295]	- long pozitiv

long long long long int signed long long signed long long int	≥ 8 bytes	Cel puțin $[-2^{63}, +2^{63}-1]$	
unsigned long long unsigned long long int		$[0, +2^{64}-1]$	- long long pozitiv

float	≥ 4 bytes	Aproximativ (pe pozitiv) [-1.2E+38, 3.4e+38]	- cel puțin 6 zecimale exacte
double	≥ 8 bytes	Aproximativ (pe pozitiv) [-2.3E+308, 1.7e+308]	- cel puțin 15 zecimale exacte
long double	≥ 10 bytes	Aproximativ (pe pozitiv) [-3.4E+4932, 1.1E+4932]	- cel puțin 19 zecimale exacte

- Operatori

1. Operatori aritmetici
2. Operatorul de atribuire
3. Operatori de incrementare și decrementare
4. Operatori de egalitate, logici și relaționali
5. Operatori pe biți
6. Alți operatori:

de acces la elemente unui tablou, de apel de funcție, de adresa, de referențiere, sizeof, de conversie explicită, condițional, virgulă.

- Shiftare

Expresie	Reprezentare binară	Observație
a = 12	0000 0000 0000 1100	
b = 3600	0000 1110 0001 0000	
a << 1	0000 0000 0001 1000	Valoarea rezultată este $24 = 12 * 2^1$
a << 2	0000 0000 0011 0000	Valoarea rezultată este $48 = 12 * 2^2$
a << 5	0000 0001 1000 0000	Valoarea rezultată este $384 = 12 * 2^5$
a >> 1	0000 0000 0000 0110	Valoarea rezultată este $6 = 12 / 2^1$
a >> 2	0000 0000 0000 0011	Valoarea rezultată este $3 = 12 / 2^2$
b >> 4	0000 0000 1110 0001	Valoarea rezultată este $225 = 3600 / 2^4$

- operatori pe biti

Expresie	Reprezentare pe 4 biți				Observație
a = 10	1	0	1	0	
b = 7	0	1	1	1	
a & b	0	0	1	0	1 dacă ambi biți sunt 1, 0 în rest
a b	1	1	1	1	1 dacă cel puțin unul din cei doi biți este 1, 0 în rest
a ^ b	1	1	0	1	1 dacă doar unul din cei doi biți este 1, 0 în rest
~ a	0	1	0	1	1 unde bitul a fost 0 și 0 unde bitul a fost 1
~ b	1	0	0	0	1 unde bitul a fost 0 și 0 unde bitul a fost 1


Ordinea de precedență și asociativitate

Precedence	Operator	Description	Associativity
1	++ --	Suffix/postfix increment and decrement	Left-to-right
	()	Function call	
	[]	Array subscripting	
	.	Structure and union member access	
	->	Structure and union member access through pointer	
	(type){list}	Compound literal ^[CS9]	
2	++ --	Prefix increment and decrement	Right-to-left
	+ -	Unary plus and minus	
	! ~	Logical NOT and bitwise NOT	
	(type)	Type cast	
	*	Indirection (dereference)	
	&	Address-of	
	sizeof _Alignof	Size-of ^[note 1] Alignment requirement ^(C11)	
3	* / %	Multiplication, division, and remainder	Left-to-right
4	+ -	Addition and subtraction	
5	<< >>	Bitwise left shift and right shift	
6	< <=	For relational operators < and ≤ respectively	
	> >=	For relational operators > and ≥ respectively	
7	== !=	For relational = and ≠ respectively	
8	&	Bitwise AND	
9	^	Bitwise XOR (exclusive or)	
10		Bitwise OR (inclusive or)	
11	&&	Logical AND	
12		Logical OR	
13 ^[note 2]	?:	Ternary conditional ^[note 3]	Right-to-Left
14	=	Simple assignment	
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= =	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

http://en.cppreference.com/w/c/language/operator_precedence


Conversii implicite – reguli (1)

- conversii implicite **la atribuire**
 - valoarea expresiei din dreapta se convertește la tipul expresiei din stânga
 - **pot apare pierderi** – dacă tipul nu este suficient de încăpător nu poate efectua conversia
- conversia implicită a **tipurilor aritmetice**
 - convesia operanzilor se face la cel mai apropiat tip care poate reprezenta valorile ambilor operanzi



long long
long
int
short
char

Ierarhia tipurilor întregi standard



long double
double
float


Ierarhia tipurilor reale standard

Conversii implicite – reguli (2)

- când apar într-o expresie tipurile de date **char** și **short** (atât signed și unsigned) sunt convertite la tipul int (**promovarea întregilor**)
- În orice operație între două operanzi, ambii operanzi sunt convertiți la tipul de date **mai înalt** în ierarhie:

- tipul care se reprezintă pe un **număr mai mare de octeți** are un rang mai mare în ierarhie
- pentru același tip, varianta **fără semn** are rang mai mare decât cea cu semn
- tipurile **reale** au rang mai mare decât tipurile întregi

long double
double
float
unsigned long long int
long long int
unsigned long int
long int
unsigned int
int
short int
char
_Bool | bool



Probleme propuse

1. Diferenta dintre doua date calendaristice (calculul nr de zile fata de inceputul anului)
2. Ordonarea unei matrici crescator pe fiecare linie si pe fiecare coloana (copierea matricii intr-un vector, ordonarea acestuia si redistribuirea liniilor in matrice)

