

PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2017-2018

Cursul 4

ACM - ICPC

University of Bucharest	Romania	Echipa Lacheta	ACCEPTED
University of Bucharest	Romania	Samsarienii	ACCEPTED
University of Bucharest	Romania	Too Simplex	ACCEPTED

Standings [5:00:00]

Last success: 4:58:52, UTCN_SnackOverflow, F.

Place	User	Region	A	B	C	D	E	F	G	H	I	J	K	L	Total	Penalty			
1	Echipa Lacheta		+ (0:26)	+2 (1:18)	+1 (3:14)	+	+ (4:09)	+ (0:53)	+		+ (4:41)	+2 (0:40)	+	(3:26)	10	1348			
2	UzhNU_push --force		+1 (0:53)	+	(4:50)	+	(1:58)	+	(3:48)	+	(0:15)	+	(0:29)	-3	+ (1:19)	+ (0:20)	-4	8	852
3	FMI_1		+1 (1:09)	+	(2:36)	+	(3:03)	-4	+2 (2:31)	+	(0:27)			+ (0:40)	+ (1:44)		7	790	
4	LNU Algotesters		+ (0:59)			+	(2:34)	-6	+ (1:38)	+1 (0:22)			+1 (3:53)	+	(1:25)	+4 (4:06)	7	1017	
5	KNU WakeUp		+1 (0:59)		-1	+	(3:35)	+30 (4:51)	+3 (1:54)	+	(0:39)		+2 (3:22)	+	(1:30)		7	1730	
6	TeamName		+ (0:17)			+1 (2:08)	-3	+	(0:48)	+1 (0:27)			-3	+	(0:32)	+2 (4:34)	6	606	
7	random variable		+1 (0:29)	-1	-2	+	(3:14)		+2 (1:36)	+	(0:39)			+	(1:04)	+ (3:03)	6	665	
8	NTU_KhPI_Morituri::Reborn		+3 (1:40)			+	(2:07)	-7	+1 (1:32)	+1 (1:02)			+	(2:37)	+	(0:33)	-5	6	671
14	Samsarienii		+1 (1:18)	+2 (4:51)		+	(3:28)		+3 (3:41)	+	(0:25)				+	(0:45)		6	988
15	ONPU_Compote		+3 (1:15)			+3 (3:12)		+1 (2:18)	+1 (0:52)			+2 (4:33)	+2 (0:37)				6	1007	
16	Team AUBG		+4 (2:53)			+	(2:07)		+3 (3:02)	+	(1:26)		+ (4:27)	+	(1:55)		6	1090	
17	KNU \$teamName		+3 (1:51)			+	(3:00)		+1 (2:11)	+4 (1:36)			+	(4:18)	+	(2:49)		6	1105
18	UzhNU_Storm		+5 (1:39)			+	(2:18)	-1	+9 (3:37)	+4 (1:52)			+15 (4:59)	+	(0:16)		6	1541	
19	Too Simplex		+1 (0:58)			+	(3:02)		+ (2:01)	+	(0:14)		-2	+	(0:35)		5	430	

Echipa Lacheta + antrenor



Recapitulare – cursul trecut

1. Expresii și operatori
2. Conversii
3. Tipuri derivate de date: pointeri, tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți

Programa cursului

□ Introducere

- Algoritmi
- Limbaje de programare.

□ Fundamentele limbajului C

- Introducere în limbajul C. Structura unui program C.
- Tipuri de date fundamentale. Variabile. Constante. Operatori. Expresii. Conversii.
- Tipuri derivate de date: tablouri, siruri de caractere, structuri, uniuni, câmpuri de biți, enumerări, pointeri
- Instrucțiuni de control
- Directive de preprocessare. Macrodefiniții.
- Funcții de citire/scriere.
- Etapele realizării unui program C.



□ Fișiere text

- Funcții specifice de manipulare.

□ Funcții (1)

- Declarație și definire. Apel. Metode de transmitere a parametrilor. Pointeri la funcții.

□ Tablouri și pointeri

- Legătura dintre tablouri și pointeri
- Aritmetică a pointerilor
- Alocarea dinamică a memoriei
- Clase de memorare

□ Siruri de caractere

- Funcții specifice de manipulare.

□ Fișiere binare

- Funcții specifice de manipulare.

□ Structuri de date complexe și autoreferite

- Definire și utilizare

□ Funcții (2)

- Funcții cu număr variabil de argumente.
- Preluarea argumentelor funcției main din linia de comandă.
- Programare generică.

□ Recursivitate

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Şiruri de caractere

- **un sir de caractere (string)** este o zonă de memorie ocupată cu caractere/char-uri (un char ocupă un octet) terminată cu un octet de valoare 0 (caracterul '\0' are codul ASCII egal cu 0).
- o variabilă care reprezintă un sir de caractere este un pointer (reține adresa) la primul octet.
- se poate reprezenta ca:
 - tablou de caractere (pointer constant):
 - `char sir1[10];`
 - `char sir2[10] = "exemplu";`
 - pointer la caractere:
 - `char *sir3;`
 - `char *sir4 = "exemplu";`

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

```
main.c
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int i;
6
7     for(i = 33; i<= 127; i++)
8     {
9         printf("%c ",i);
10        if ((i-2) % 10 == 0)
11            printf("\n");
12    }
13    return 0;
14
15
16 }
```

Ce afișează programul?

```
! " # $ % & ' ( ) *
+ , - . / 0 1 2 3 4
5 6 7 8 9 : ; < = >
? @ A B C D E F G H
I J K L M N O P Q R
S T U V W X Y Z [ \
] ^ _ ` a b c d e f
g h i j k l m n o p
q r s t u v w x y z
{ | } ~
```

```
Process returned 0 (0x0)    execution time :
Press ENTER to continue.
```

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

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	@	Ø	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		DEL

Codurile ASCII

- cod ASCII = reprezentarea numerică a unui caracter.
(ASCII – American Standard Code for Information Interchange)

Extended ASCII Codes

128	Ç	144	É	160	á	176	¤	192	Ł	208	₩	224	α	240	≡
129	ü	145	æ	161	í	177	¤¤	193	ŁŁ	209	〒	225	฿	241	±
130	é	146	Æ	162	ó	178	¤¤¤	194	Ŧ	210	ŦŦ	226	Γ	242	≥
131	â	147	ð	163	ú	179	_	195	ŦŦ	211	₩₩	227	π	243	≤
132	ã	148	ö	164	ñ	180	-	196	-	212	₪	228	Σ	244	ƒ
133	à	149	ò	165	Ñ	181	-	197	+	213	₹	229	σ	245	Ј
134	å	150	û	166	º	182		198	₣	214	₲	230	μ	246	÷
135	ç	151	ù	167	º	183		199	₭	215	₱	231	τ	247	₪
136	è	152	ÿ	168	¸	184	_	200	₵	216	₭	232	Φ	248	°
137	ë	153	Ö	169	ѓ	185		201	₹	217	Ј	233	ଓ	249	.
138	è	154	Ü	170	ѓ	186		202	₩	218	₲	234	₪	250	.
139	ï	155	œ	171	½	187	_	203	₩	219	₪	235	฿	251	₩
140	î	156	€	172	¼	188	_	204	₣	220	₪	236	∞	252	¤
141	ì	157	⌘	173	¡	189	_	205	=	221	₪	237	◊	253	¤
142	Ä	158	฿	174	«	190	_	206	₪	222	₪	238	₪	254	₪
143	Å	159	ƒ	175	»	191	_	207	₩	223	₪	239	₪	255	₪

Tipuri de date structurate

Limbajul C permite creare tipurilor de date în 5 moduri:

- structura (**struct**) – grupează mai multe variabile sub același nume;
- câmpul de biți (variațiune a structurii) → acces ușor la bitii individuali
- uniunea (**union**) – face posibil ca aceleași zone de memorie să fie definite ca două sau mai multe tipuri diferite de variabile
- enumerarea (**enum**) – listă de constante întregi cu nume
- tipuri definite de utilizator (**typedef**) – definește un nou nume pentru un tip existent

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, câmpuri de biți, uniuni, enumerări, tipuri definite de utilizator (typedef).
2. Instrucțiuni de control.
3. Directive de procesare. Macrodefiniții.

Structuri

- ❑ variabile grupate sub același nume.
- ❑ sintaxa:

```
struct <nume> {
    < tip 1 >    <variabila 1>;
    < tip 2 >    <variabila 2>;
    -----
    < tip n >    <variabila n>;
} lista_identificatori_de_tip_struct;
```

- ❑ variabilele care fac parte din structură sunt denumite membri (elemente sau câmpuri) ai structurii.

Structuri

- **struct <nume> {**
 < tip 1 > <variabila 1>;
 < tip 2 > <variabila 2>;

 < tip n > <variabila n>;
} lista_identificatori_de_tip_struct;

□ observații:

- dacă numele structurii (<nume>) lipsește, structura se numește **anonimă**. Dacă lista identificatorilor declarați lipsește, se definește doar tipul structură. Cel puțin una dintre aceste specificații trebuie să existe.
- dacă <nume> este prezent → se pot declara noi variabile de tip structura **struct <nume> <lista noilor identificatori>**;
- referirea unui membru al unei variabile de tip structură se face cu operatorul de selecție punct **.** care precizează identificatorul variabilei și al câmpului.

Structuri

- ❑ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
    int cod;  
};
```

- ❑ numele **adrese** identifică aceasta structură de date particulară.
- ❑ **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Structura din memorie pentru variabila A de tip adrese

nume 30 octeti

strada 40 octeti

oras 20 octeti

judet 3

cod 4

Structuri

- ❑ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
    int cod;  
};
```

The screenshot shows a terminal window with the following content:

```
#include <stdio.h>  
#include <stdlib.h>  
  
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[3];  
        int cod;  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octeți \n", (int)sizeof(A.nume));  
    printf("A.cod se memorează pe %d octeți \n", (int)sizeof(A.cod));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

The terminal output is:

```
A.nume se memorează pe 30 octeți  
A.cod se memorează pe 4 octeți  
A se memorează pe 100 octeți
```

The line "A se memorează pe 100 octeți" is highlighted with a red rectangle.

Process returned 0 (0x0) execution time : 0.006 s
Press ENTER to continue.

Compilatorul alocă memorie în plus pentru aliniere (multiplu de 4)

Structuri

□ exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[6];  
    int cod;  
};
```

The screenshot shows a code editor with a syntax-highlighted C program. A red box highlights the variable 'judet[6]'. To the right, a terminal window displays the output of the program, which includes memory allocation details for each field.

```
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[6];  
        int cod;  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octet", (int)sizeof(A.nume));  
    printf("A.cod se memorează pe %d octeți \n", (int)sizeof(A.cod));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

A. nume se memorează pe 30 octeți
A.cod se memorează pe 4 octeți
A se memorează pe 100 octeți
Process returned 0 (0x0)
Press ENTER to continue.

- numele **adrese** identifică aceasta structură de date particulară.
- **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Structuri

- exemplu:

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], judet[3];  
};
```

The screenshot shows a code editor with the following C code:

```
int main(){  
    struct adrese{  
        char nume[30];  
        char strada[40];  
        char oras[20], judet[3];  
    };  
    struct adrese A;  
    printf("A.nume se memorează pe %d octeți \n", (int)sizeof(A.nume));  
    printf("A se memorează pe %d octeți \n", (int)sizeof(A));  
  
    return 0;  
}
```

The variable `judet[3]` is highlighted with a red box. The output window below shows the results of the `printf` statements:

```
A.nume se memorează pe 30 octeți  
A se memorează pe 93 octeți
```

- numele **adrese** identifică aceasta structură de date particulară.
- **struct adrese A;** declară o variabilă de tip adrese și îi aloca memorie

Compilatorul alocă memorie în plus pentru aliniere numai când avem tipuri de date diferite

Structuri

□ exemplu:

```
struct adresă {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A, B, C;
```

Definește un tip de structură numit **adresă** și declară ca fiind de acest tip variabilele **A, B, C**

```
struct {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A;
```

Declară o variabilă numită **A** definită de structura care o precede.

Structuri

- exemplu:
- accesul la membri structurii se face prin folosirea operatorului punct:
nume_variabila.nume_camp
- `scanf("%d", &C.cod);`
- atribuiri pentru variabile de tip structură: **B = A;**

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
} A, B, C;
```

Definește un tip de structură numit **adrese** și declară ca fiind de acest tip variabilele **A, B, C**

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, câmpuri de biți, uniuni, enumerări, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefiniții.

Câmpuri de biți

- tip special de membru al unei structuri care definește cât de lung trebuie să fie câmpul, în biți.
- permite accesul la un singur bit.
- putem stoca mai multe variabile boolene într-un singur octet.
- nu se poate obține adresa unui câmp de biți.
- **adaugă mai multă structurare.**

□ **Sintaxa:**

```
struct <nume> {  
    < tip 1 >    <variabila 1>: lungime;  
    < tip 2 >    <variabila 2>: lungime;  
    -----  
    < tip n >    <variabila n>: lungime;  
} lista_identificatori_de_tip_struct;
```

Câmpuri de biți

□ Sintaxa:

```
struct <nume> {
    < tip 1 >  <variabila 1>: lungime;
    < tip 2 >  <variabila 2>: lungime;
    -----
    < tip n >  <variabila n>: lungime;
} lista_identificatori_de_tip_struct;
```

□ Observații:

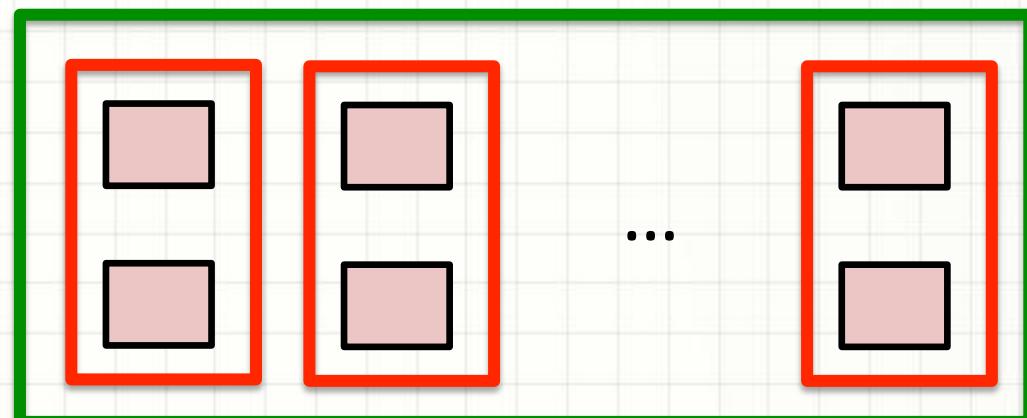
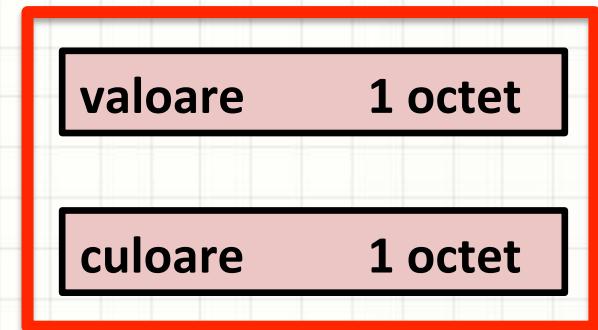
- tipul câmpului de biți poate fi doar: **int**, **unsigned** sau **signed**.
- câmpul de biți cu lungimea 1 → **unsigned** (un singur bit nu poate avea semn).
- unele compilatoare → doar unsigned.
- lungime → numărul de biți dintr-un câmp.

Câmpuri de biți

- exemplu: în cadrul unui joc de cărți reprezentăm o carte printr-o structură

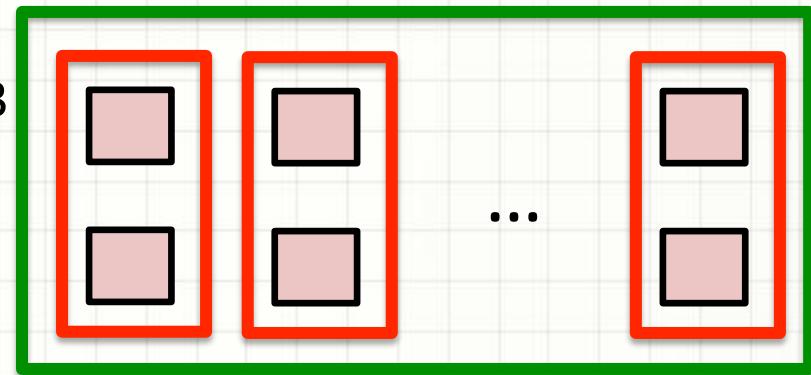
```
struct carteJoc {  
    unsigned char valoare; // valori între 1 și 13  
    unsigned char culoare; // valori între 1 și 4  
};
```

```
struct carteJoc PachetCartiJoc[52];
```



Câmpuri de biți

```
struct carteJoc {  
    unsigned char valoare; // valori între 1 și 13  
    unsigned char culoare; // valori între 1 și 4  
};  
struct carteJoc PachetCartiJoc[52];
```



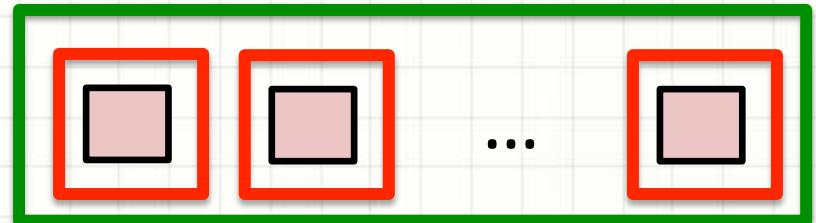
main.c

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4  
5 int main(){  
6     struct carteJoc{  
7         unsigned char valoare;  
8         unsigned char culoare;  
9     } A;  
10  
11     printf("dimensiune A este %d \n", sizeof(A));  
12     struct carteJoc pachetCartiJoc[52];  
13  
14     printf("dimensiune pachetCartiJoc este %d \n", sizeof(pachetCartiJoc));  
15  
16     return 0;  
17 }  
18  
19
```

dimensiune A este 2
dimensiune pachetCartiJoc este 104
Process returned 0 (0x0) execution time : 0
Press ENTER to continue.

Câmpuri de biți

```
struct carteJoc {  
    unsigned char valoare : 4; // 4 biți  
    unsigned char culoare : 2; // 2 biți  
};  
struct carteJoc PachetCartiJoc[52];
```



main.c

```
1 #include <stdio.h>  
2 #include <stdlib.h>  
3  
4  
5 int main(){  
6     struct carteJoc{  
7         unsigned char valoare : 4;  
8         unsigned char culoare : 2;  
9     } A;  
10  
11     printf("dimensiune A este %d \n", sizeof(A));  
12     struct carteJoc pachetCartiJoc[52];  
13  
14     printf("dimensiune pachetCartiJoc este %d \n", sizeof(pachetCartiJoc));  
15  
16     return 0;  
17 }  
18  
19
```

dimensiune A este 1
dimensiune pachetCartiJoc este 52
Process returned 0 (0x0) execution time : 0.00
Press ENTER to continue.

Câmpuri de biți

- ❑ exemplu: în aceeași structură putem combina câmpuri de biți și membri normali

```
struct adrese {  
    char nume[30];  
    char strada[40];  
    char oras[20], jud[3];  
    int cod;  
};
```

```
struct angajat {  
    struct adrese A;  
    float plata;  
    unsigned statut: 1; // activ sau intrerupt  
    unsigned plata_ora: 1; // plata cu ora  
    unsigned impozit: 3; // impozit rezultat  
};
```

- ❑ observație: definește o înregistrare despre salariat care foloseste doar un octet pentru a păstra 3 informații: statutul, daca este platit cu ora și impozitul.
 - ❑ fără câmpul de biti, aceste informații ar fi ocupat 3 octeți.

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, câmpuri de biți, **uniuni**, enumerări, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefiniții.

Uniuni

- ❑ tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie.
- ❑ membri unei uniuni au de obicei tipuri diferite

Sintaxa:

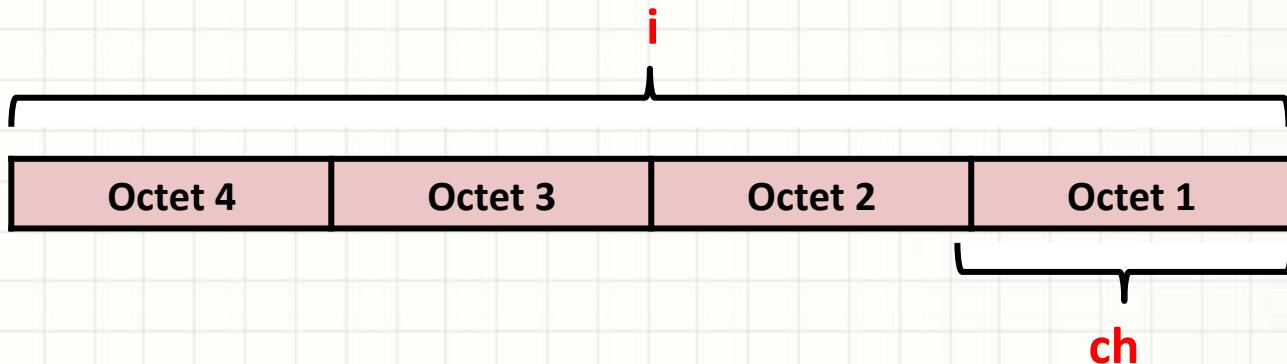
```
union <nume> {  
    < tip 1 >    <variabila 1>;  
    < tip 2 >    <variabila 2>;  
    -----  
    < tip n >    <variabila n>;  
} lista_identificatori_tip_union;
```

Uniuni

- tip special de structuri ai cărei membri folosesc la momente diferite aceeași locație în memorie.
- membri unei uniuni au de obicei tipuri diferite
- exemplu:

```
union tip_u {  
    int i;  
    char ch;  
};
```

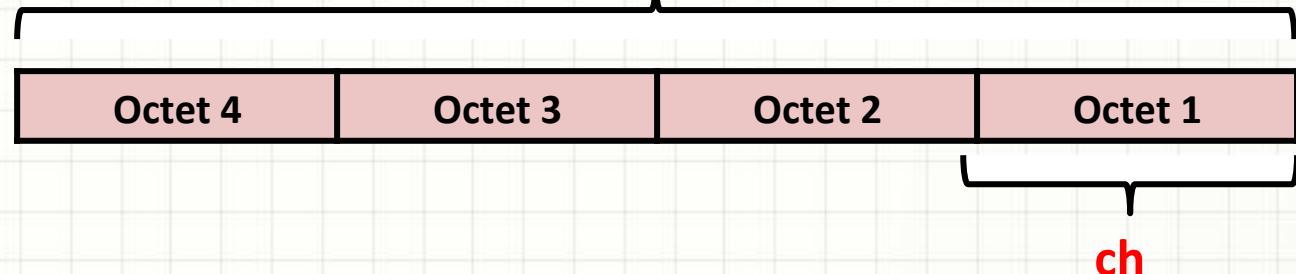
union tip_u A;



- când este declarată o variabilă de tip **union** compilatorul alocă automat memorie suficientă pentru a păstra cel mai mare membru al acesteia.

Uniuni

exemplu



```
main.c X
1 #include <stdio.h>
2 #include <stdlib.h>
3
4
5 int main(){
6     union tip_u{
7         int i;
8         unsigned char ch;
9     };
10
11     union tip_u A;
12     printf("Dimensiune A = %d \n", sizeof(A));
13     A.ch = 10;
14     printf("A.ch = %d \n", A.ch);
15     printf("A.i = %d \n", A.i);
16
17     A.i = 300;
18     printf("A.ch = %d \n", A.ch);
19     printf("A.i = %d \n", A.i);
20
21
22
23     return 0;
24 }
```

```
Dimensiune A = 4
A.ch = 10
A.i = 10
A.ch = 44
A.i = 300
```

```
Process returned 0 (0x0)    execution time ...
Press ENTER to continue.
```

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, câmpuri de biți, uniuni, **enumerări**, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefiniții.

Enumerări

- ❑ multime de constante de tip întreg care specifică toate valorile permise pe care le poate avea o variabilă de acel tip
- ❑ atât numele generic al enumerării cât și lista de variabile sunt optionale
- ❑ constanta unui element al enumerării este fie asociată implicit, fie explicit. Implicit, primul element are asociată valoarea 0, iar pentru restul este valoarea precedentă+1.

❑ **sintaxa:**

```
enum <nume> {  
    lista enumerarilor  
} lista variabile;
```

Enumerări

❑ exemplu

enum {a, b, c, d}; → a = 0, b = 1, c = 2, d = 3

enum {a, b, c=7, d}; → a = 0, b = 1, c = 7, d = 8

enum {a=4, b=-3, c=9, d=-8}; enum {false,true};

```
main.c ✘
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      enum zile{
8          luni,
9          marti,
10         miercuri,
11         joi,
12         vineri,
13         sambata,
14         duminica};
15
16         printf("Azi e ziua a %d-a a saptamanii \n", joi);
17
18     return 0;
19 }
```

Azi e ziua a 3-a a saptamanii
Process returned 0 (0x0) execution time : 0.00
Press ENTER to continue.

Enumerări

□ exemplu

enum {a, b, c, d}; → a = 0, b = 1, c = 2, d = 3

enum {a, b, c=7, d}; → a = 0, b = 1, c = 7, d = 8

enum {a=4, b=-3, c=9, d=-8}; enum {false,true};

```
main.c x
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     enum zile{
6         luni=1,
7         marti,
8         miercuri,
9         joi,
10        vineri,
11        sambata,
12        duminica};
13
14     printf("Azi e ziua a %d-a a saptamanii \n",joi);
15
16
17
18     return 0;
19 }
```

Azi e ziua a 4-a a saptamanii
Process returned 0 (0x0) execution time :
Press ENTER to continue.

Enumerări

❑ exemplu

main.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4
5  int main(){
6
7      enum zile{
8          luni=1,
9          marti,
10         miercuri,
11         joi,
12         vineri,
13         sambata,
14         duminica} azi,maine;
15
16
17         azi = joi;
18         printf("Azi e ziua a %d-a a saptamanii \n",azi);
19         maine = azi+1;
20         printf("Maine va fi a %d-a a saptamanii \n",maine);
21
22
23     return 0;
24 }
```

Azi e ziua a 4-a a saptamanii
Maine va fi a 5-a a saptamanii

Process returned 0 (0x0) execution t
Press ENTER to continue.

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, câmpuri de biți, uniuni, enumerări, tipuri definite de utilizator (typedef).
2. Instrucțiuni de control.
3. Directive de procesare. Macrodefiniții.

Specificatorul `typedef`

- definește explicit noi tipuri de date.
- nu se declară o variabilă sau o funcție de un anumit tip, ci se asociază un nume (sinonimul) tipului de date.
- **sintaxa:**
`typedef <definiție tip> <identificator>;`
- **exemple:**

```
typedef unsigned int natural;  
typedef long double tablouNumereReale [100] ;  
tablouNumereReale a, b, c;  
natural m,n,i;
```

Specificatorul `typedef`

```
1 #include <stdio.h>
2
3 int main()
4 {
5     typedef long double tablouNumereReale[100];
6     tablouNumereReale a;
7     printf("dimensiunea lui a este %d\n", sizeof(a));
8     long double d;
9     printf("dimensiunea lui d este %d\n", sizeof(d));
10
11     return 0;
12 }
```

.....

dimensiunea lui a este 1600
dimensiunea lui d este 16
Pascal Alexeas MacBook Pro: ~ baza>

Exemplu subiect de examen

- definim regulile unui joc de cărți
- definim câteva tipuri de date structurate care să vă ajute în implementare (cărți, jucători, etc.)
- vă dăm main-ul/prototipul funcțiilor
- trebuie să implementați funcțiile

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4
5 struct carteJoc {
6     unsigned char valoare : 4;
7     unsigned char culoare : 2;
8 };
9
10 typedef struct {
11     struct carteJoc cartiJucator[4];
12     struct carteJoc cartiLuate[32];
13     int nrPuncte;
14 } jucator;
15
16 typedef struct {
17     struct carteJoc cartiPachet[32];
18     int carteCurenta;
19     int nrTotalCarti;
20 } pachetCarti;
21
22 void citesteCartiPachet(pachetCarti *p, char *numeFisier);
23 void amestecaPachet(pachetCarti *p);
24 void imparteCartiDinPachet(pachetCarti *p ,jucator *j1,jucator *j2);
25 void afiseazaCarti(jucator j,char* numeJucator);
26 int joacaORunda(jucator* j1, jucator* j2, int* nrCartiJucate);
27 void numaraPunctele(jucator *j1, jucator* j2);
28
```

```
28
29 int main()
30 {
31     pachetCarti pachet;
32     char* numeFisier ="carti.in";
33     citesteCartiPachet(&pachet,numeFisier);
34     amestecaPachet(&pachet);
35     jucator Player, CPU;
36     //imparte 4 carti din pachet
37     imparteCartiDinPachet(&pachet,&Player,&CPU);
38     afiseazaCarti(Player,"Player");
39     afiseazaCarti(CPU,"CPU");
40
41
42     int cateCartiAuFostJucate=0;
43     jucator *j1=&Player,*j2=&CPU;
44     //se joaca cat timp mai sunt carti in pachet
45     while(pachet.carteCurenta < pachet.nrTotalCarti)
46     {
47         int castigatorMana = joacaO Runda(j1,j2,&cateCartiAuFostJucate);
48         //cine a castigat mana
49         if(castigatorMana == 1) //a castigat j1
50             imparteCartiDinPachet(&pachet,j1,j2);
51         else
52             imparteCartiDinPachet(&pachet,j2,j1);
53     }
54     numaraPunctele(j1,j2);
55     return 0;
56
57 }
```

Stilul de codare al studentilor...

```
void PLAYERvsCPU()
{
    int alegere, cartip[4], carticpu[4], i, vizitatp[4], vizitatcpu[4];
    int incepe=0, l1=0, l2=0, s, k=8;
    //initializare frecventa cu 0.
    for(i=0;i<4;i++) {vizitatp[i]=0; vizitatcpu[i]=0;}
    //impartirea cartilor
    printf("\nBun venit!\n\nCartile tale sunt: ");
    for(i=0;i<4;i++)
    {
        cartip[i]=pachet[i];
        carticpu[i]=pachet[i+4];
    }
    for(i=0;i<4;i++)
        printf("%d ", cartip[i]);
joc:
i=0;
if(incepe==0)
    { intoarcere: printf("\nAlege cartea pe care sa o joci:");
      exista: scanf("%d", &alegere);
      if(vizitatp[alegere]==1) goto exista;
      vizitatp[alegere]=1;
      printf("\nAi ales cartea :%d ", cartip[alegere]);
      //verific daca are o carte asemanatoare sau un 7
      while(i<4)
      { //daca are cel ptin o carte in mana ...
        if((carticpu[i]==cartip[alegere] || carticpu[i]==7) && vizitatcpu[i]==0)
        { printf("\nCPU a ales cartea : %d ",carticpu[i]);
          vizitatcpu[i]=1;
          if(continuare()==1) {i++; goto intoarcere;}
          else { printf("\nCPU castiga aceasta mana\n");
            incene=1;
        }
      }
    }
}
```

Cuprinsul cursului de azi

1. Tipuri derivate de date: siruri de caractere, structuri, campuri de biti, uniuni, enumerari, tipuri definite de utilizator (typedef).
2. Instructiuni de control.
3. Directive de procesare. Macrodefinitii.

Instrucțiuni de control

- ❑ reprezintă:
 - ❑ elementele fundamentale ale funcțiilor
 - ❑ comenzi date calculatorului
 - ❑ determină fluxul de control al programului (ordinea de execuție a operațiilor din program)
- ❑ **instructiuni de bază**
 - ❑ instructiunea expresie
 - ❑ instructiunea vidă
 - ❑ instructiuni sevențiale/liniare
 - ❑ instructiuni decizionale/selective simple sau multiple
 - ❑ instructiuni repetitive/ciclice/iterative
 - ❑ instructiuni de salt condiționat/necondiționat
 - ❑ instructiunea return

Instructiuni de control

- ❑ **instructiuni compuse**
 - ❑ create prin combinarea instructiunilor de bază
- ❑ **programare structurată**
 - ❑ Teorema Böhm-Jacopini: fluxul de control poate fi exprimat folosind doar trei tipuri de **instructiuni de control**:
 - ❑ instructiuni secentiale
 - ❑ instructiuni decizionale
 - ❑ instructiuni repetitive

Instrucțiunea expresie

- ❑ formată dintr-o expresie urmată de semnul ;
 - ❑ expresie;
- ❑ cele mai frecvente
 - ❑ se bazează pe expresii de atribuire, aritmetice și de incrementare / decrementare
 - ❑ adică expresii care au efecte secundare: schimbă valoarea unui operand

Exemple:

```
a = 123;  
b = a + 5;  
b++;
```

- ❑ expresie vs. instrucțiune

Expresie

i++

a=a-5

Instrucțiune

i++;

a=a-5;

Instrucțiunea vidă

- ❑ o instrucțiune care constă doar din caracterul ;
 - ❑ folosită în locurile în care limbajul impune existența unei instrucțiuni, dar programul nu trebuie să execute nimic
- ❑ cel mai adesea instrucțiunea vidă apare în combinație cu instrucțiunile repetitive
 - ❑ vezi instrucțiunea for

Instrucțiunea compusă

- ❑ numită și instrucțiune bloc
- ❑ alcătuită prin gruparea mai multor instrucțiuni și declarații
 - ❑ folosite în locurile în care sintaxa limbajului presupune o singură instrucțiune, dar programul trebuie să efectueze mai multe instrucțiuni
 - ❑ gruparea
 - ❑ includerea instrucțiunilor între accolade, {}
 - ❑ astfel compilatorul va trata secvența de instrucțiuni ca pe o singură instrucțiune
 - ❑ *{secvență de declarații și instrucțiuni }*

Instrucțiuni decizionale/selective

- ❑ ramifică fluxul de control în funcție de valoarea de adevăr a expresiei evaluate

- ❑ limbajul C furnizează două instrucțiuni decizionale
 - ❑ instrucțiunea **if** – instrucțiune decizională simplă
 - ❑ instrucțiunea **switch** - instrucțiune decizională multiplă

Instrucțiunea IF

- ❑ instrucțiunea selectivă fundamentală
 - ❑ permite selectarea uneia dintre două alternative în funcție de valoarea de adevăr a expresiei testate
- ❑ forma generală:

```
if (expresie)
    {bloc de instructiuni 1};
else
    {bloc de instructiuni 2};
```
- ❑ valoarea expresiei incluse între paranteze rotunde trebuie să fie un scalar
 - ❑ dacă e nenulă se execută *blocul de instrucțiuni 1 (instrucțiunea compusă)*, altfel se execută *blocul de instrucțiuni 2*
- ❑ ramura else poate lipsi

Instrucțiunea IF

- se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

```
seminar1.c X
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a,b;
7     scanf("%d %d",&a,&b);
8     if (a==0)
9     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     printf("%d \n", (int)pow(a%10,b%4+4) % 10);
24
25     return 0;
26 }
27
```

Instructiunea IF

- erori frecvente:
 - nu includem acoladele

```
if (a>b)
    a = a+b;
    b = a;
```

- Întotdeauna se va executa instrucțiunea `b=a`;
- confundarea operatorul de egalitate `==`, cu operatorul de atribuire `=`

Exemple comparative:

```
a = 2;
if ( a == 10 )
    printf("a este 10 \n");
```

```
a = 2;
if ( a = 10 )
    printf("a este 10 \n");
```

- | | |
|--|---|
| <ul style="list-style-type: none">□ mesajul <code>a este 10</code> nu va fi afișat<ul style="list-style-type: none">□ după testarea egalității folosind operatorul <code>==</code> se returnează 0<ul style="list-style-type: none">□ (2 nefiind egal cu 10) | <ul style="list-style-type: none">□ mesajul <code>a este 10</code> va fi afișat întotdeauna<ul style="list-style-type: none">□ expresia <code>a = 10</code><ul style="list-style-type: none">□ a ia valoarea 10□ se evaluatează adevărat la executarea instrucțiunii <code>printf</code> |
|--|---|

Instrucțiunea IF

- instrucțiuni IF imbricate
 - pe oricare ramură poate conține alte instrucțiuni if
- forma generală:

```
if (expresie1)
    if (expresie2) {bloc de instructiuni 1};
    else {bloc de instructiuni 2};
else
    {bloc de instructiuni 2};
```

Exemplu:

```
int a, b;
// ...
if ( a <= b )
    if ( a == b )
        printf("a = b");
    else
        printf("a < b");
else
    printf("a > b");
```

Instructiunea IF

- instructiuni IF în cascadă
 - testează succesiv mai multe condiții implementând o variantă de selecție multiplă
- forma generală:

```
if (expresie1) {bloc de instructiuni 1};  
else if (expresie2) {bloc de instructiuni 2};  
else if (expresie3) {bloc de instructiuni 3};  
...  
else {bloc de instructiuni N};
```

```
#include <stdio.h>  
  
int main() {  
    float nota;  
  
    printf("Introduceti o nota in intervalul [1, 10]: ");  
    scanf("%f", &nota);  
  
    if ( nota > 9 && nota <= 10)  
        printf("Calificativul este: EXCELENȚA");  
    else if ( nota > 8 && nota <= 9)  
        printf("Calificativul este: Foarte bine");  
    else if ( nota > 7 && nota <= 8)  
        printf("Calificativul este: Bine");  
    else if ( nota > 5 && nota <= 7)  
        printf("Calificativul este: Suficient");  
    else printf("Calificativul este: Insuficient");  
  
    return 0;  
}
```

Instrucțiunea SWITCH

- ❑ efectuează selecția multiplă
 - ❑ util când expresia de evaluat are mai multe valori posibile
- ❑ forma generală

```
switch (expresie){  
    case val_const_1: {bloc de instructiuni 1};  
    case val_const_2: {bloc de instructiuni 2};  
    ....  
    case val_const_n: {bloc de instructiuni N};  
    default: {bloc de instructiuni D};  
}
```

Instrucțiunea SWITCH

- ❑ poate fi întotdeauna reprezentată prin instrucțiunea IF
 - ❑ de regulă prin instrucțiuni IF cascade
- ❑ în cazul instrucțiunii switch fluxul de control sare direct la instrucțiunea corespunzătoare valorii expresiei testate
- ❑ switch este mai rapid și codul rezultat mai ușor de înțeles

Instructiunea SWITCH

```
#include <stdio.h>

int main()
{
    int nr1, nr2, rez;
    char op;

    printf("Introduceti o expresie aritmetica sub forma: nr1 operator nr2: ");
    scanf("%d %c %d", &nr1, &op, &nr2);

    switch (op)
    {
        case '+': rez = nr1 + nr2; break;
        case '-': rez = nr1 - nr2; break;
        case '*': rez = nr1 * nr2; break;
        case '/': rez = nr1 / nr2; break;
        case '%': rez = nr1 % nr2; break;
    }

    printf("Valoarea expresiei aritmetice introduse este: %d \n", rez);

    return 0;
}
```

Rezultatul unei rulări a acestui program este:

```
Introduceti o expresie aritmetica sub forma: nr1 operator nr2: 4 + 7
Valoarea expresiei aritmetice introduse este: 11
```

Instrucțiunea SWITCH

- se citesc numerele naturale a și b de la tastatură. Să se afișeze ultima cifră a numărului a^b .

The screenshot shows a code editor window titled "seminar1_2.c". The code is written in C and calculates the last digit of a^b . It includes #include directives for stdio.h and math.h, defines main(), and uses scanf to read integers a and b. It handles cases where a or b are zero. A switch statement is used to determine the last digit based on the value of b modulo 4.

```
1  #include <stdio.h>
2  #include <math.h>
3
4  int main()
5  {
6      int a,b;
7      scanf("%d %d",&a,&b);
8      if (a==0)
9      {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     a = a%10;
24     switch (b%4)
25     {
26         case 0: printf("%d\n",a*a*a*a % 10); break;
27         case 1: printf("%d\n",a); break;
28         case 2: printf("%d\n",a*a % 10); break;
29         case 3: printf("%d\n",a*a*a % 10); break,
30     }
31     return 0;
32 }
```

Instrucțiunea SWITCH

- ❑ efectuează selecția multiplă
 - ❑ util când expresia de evaluat are mai multe valori posibile
- ❑ forma generală

```
switch (expresie){  
    case val_const_1: bloc de instructiuni 1;  
    case val_const_2: bloc de instructiuni 2;  
    ....  
    case val_const_n: bloc de instructiuni N;  
    default: bloc de instructiuni D;  
}
```

Instrucțiunea SWITCH

- ❑ mod de funcționare și constrângeri:
 - ❑ expresie se evaluează o singură dată la intrarea în instrucțiunea switch
 - ❑ expresie trebuie să rezulte într-o valoare întreagă (poate fi inclusiv caracter, dar nu valori reale sau siruri de caractere)
 - ❑ valorile din ramurile **case** notate **val_ const_i** (numite și etichete) trebuie să fie constante întregi (sau caracter), reprezentând o singură valoare
 - ❑ nu se poate reprezenta un interval de valori
 - ❑ instrucțiunile care urmează după etichetele **case** nu trebuie incluse între accolade, deși pot fi mai multe instrucțiuni, iar ultima instrucțiune este de regulă instrucțiunea **break**

Instrucțiunea SWITCH

- mod de funcționare și constrângeri:
 - dacă valoarea expresiei se potrivește cu vreuna din valorile constante din ramurile case, atunci se vor executa instrucțiunile corespunzătoare acelei ramuri, altfel se execută instrucțiunea de pe ramura **default** (dacă există)
 - dacă nu s-a întâlnit **break** la finalul instrucțiunilor de pe ramura pe care s-a intrat, atunci se continuă execuția instrucțiunilor de pe ramurile consecutive (fără verificarea etichetei) până când se ajunge la **break** sau la sfârșitul instrucțiunii **switch**, moment în care se ieșe din instrucțiunea **switch** și se trece la execuția instrucțiunii imediat următoare
 - ramura **default** este opțională iar poziția relativă a acesteia printre celelalte ramuri nu este relevantă
 - dacă nici o etichetă nu se potrivește cu valoarea expresiei testate și nu există ramura **default**, atunci instrucțiunea **switch** nu are nici un efect

Instrucțiunea SWITCH

- omiterea instrucțiunii break de la finalul unei ramuri case
 - accidentală - este o eroare frecventă
 - deliberată - permite fluxului de execuție să intre și pe ramura case următoare

```
...
switch (luna)
{
    case 1:
    case 3:
    case 5:
    case 7:
    case 8:
    case 10:
    case 12: nr_zile = 31;
               break;
    case 4:
    case 6:
    case 9:
    case 11: nr_zile = 30;
               break;
    case 2: if (bisect == 1) nr_zile = 29;
              else nr_zile = 28;
              break;
    default: printf("Luna trebuie sa fie in intervalul [1, 12] !");
}
```

Instructiunea SWITCH

seminar1_2.c

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main()
5 {
6     int a,b;
7     scanf("%d %d",&a,&b);
8     if (a==0)
9     {
10         if (b==0)
11         {
12             printf("0 la puterea 0, caz de nedeterminare \n");
13             return 0;
14         }
15     }
16
17     if(b==0)
18     {
19         printf("1\n");
20         return 0;
21     }
22
23     a = a%10;
24     switch (b%4)
25     {
26         case 0: printf("%d\n",a*a*a*a % 10);
27         case 1: printf("%d\n",a);
28         case 2: printf("%d\n",a*a % 10);
29         case 3: printf("%d\n",a*a*a % 10);
30     }
31     return 0;
32 }
```

12 33
2
4
8

Instrucțiuni repetitive

- ❑ sunt numite și instrucțiuni iterative sau ciclice
- ❑ efectuează o serie de instrucțiuni în mod repetat fiind condiționate de o expresie de control care este evaluată la fiecare iterare
- ❑ instrucțiunile iterative furnizate de limbajul C sunt:
 - ❑ instrucțiunea repetitivă cu testare inițială **while**
 - ❑ instrucțiunea repetitivă cu testare finală **do-while**
 - ❑ instrucțiunea repetitivă cu testare inițială **for**

Instrucțiunea WHILE

- execută în mod repetat o instrucțiune atât timp cât expresia de control este evaluată la adevărat
- evaluarea se efectuează la începutul instrucțiunii
 - dacă rezultatul corespunde valorii logice adevărat
 - se execută corpului instrucțiunii, după care se revine la testarea expresiei de control
 - acești pași se repetă până când expresia va fi evaluată la fals
 - acesta va determina ieșirea din instrucțiune și trecerea la instrucțiunea imediat următoare
- forma generală: **while** (*expresie*) {bloc de instrucțiuni}

Instructiunea WHILE

A programmer heads out to
the store. His wife says
"while you're out, get some
milk."

He never came home.

Instrucțiunea WHILE

```
sumaNumere.c ✘
1 #include <stdio.h>
2
3 int main()
4 {
5     int nr, i , suma;
6     printf("Introduceti un numar intreg: ");
7     scanf("%d",&nr);
8
9     i = 0; suma = 0;
10    while (i<=nr)
11    {
12        suma += i;
13        i++;
14    }
15    printf("Suma numerelor mai mici sau egale decat %d este: %d\n", nr, suma);
16
17    return 0;
18
19 }
20
```

Observații

- valorile care participă în expresia de control să fie inițializate înainte
- evitare ciclului infinit

Instrucțiunea WHILE

sumaNumere.c

```
1 #include <stdio.h>
2
3 int main()
4 {
5     int nr, i , suma;
6     printf("Introduceti un numar intreg: ");
7     scanf("%d",&nr);
8
9     i = 0; suma = 0;
10    while (((i=i+1) && (i<=nr) && (suma+i) );
11        printf("Suma numerelor mai mici sau egale decat %d este: %d\n", nr, suma);
12
13    return 0;
14
15 }
16
```

Observații

- Dacă o expresie nu mai este adevărată nu se mai continuă cu evaluarea expresiilor următoare

Instrucțiunea WHILE

```
unsigned int i=3;  
while (i>=0){  
    printf("%d\n",i);  
    i--;  
}
```

Ce afișează sevența de cod alăturată?

CICLEAZA!!!