



PROGRAMARE PROCEDURALĂ

Bogdan Alexe

bogdan.alexe@fmi.unibuc.ro

Secția Informatică, anul I,

2018-2019

Cursul 3

Organizare

- echipa de laboranți e completă la ambele serii ☺;
- Ibrahim Coskun – 1311 + 1312
- Mihai Ghidoveanu – 1321 + 1322
- Andreea Musat – 1331 + 1332
- George Glăvan – 1341 + 1342
- Lucian Bicsi – 1351 + 1352
- am stabilit modul de evaluare (după ce ne-am consultat cu laboranții).

Evaluarea la examenul din iarnă

$$Nota = \min(10, Curs + Laborator + Seminar)$$

5p 5p 1p

Seminar: nota pe activitate la seminar

Laborator: proiect = 3p + un test de laborator = 2p

- tema de proiect o primiți pe 20 noiembrie, aveți timp o lună să o implementați; este individuală (nu copiați de la unul la altul ☺);
- testul de laborator îl dați pe 15 sau 16 decembrie;
- trebuie să obțineți media de laborator minim 5 (echivalent cu 2.5 puncte din nota finală) pentru a putea intra în examenul final;
- dacă nu puteți da testul la timp, vă rog să mă contactați din timp (dacă e posibil - cu cel puțin o săptămână înainte de test);

Curs: examen scris (19 sau 20 ianuarie 2019);

- trebuie să obțineți minim nota 5 (echivalent cu 2.5 puncte din nota finală) pentru a promova examenul;
- rotunjirea notei finale: x.49 devine x, x.50 devine x+1.

Evaluarea la restanțe (iunie, septembrie)

$$Nota = \min(10, Curs + Laborator + Seminar)$$

5p 5p 1p

Seminar: se păstreaza nota de la activitatea la seminar

Laborator: test de laborator pe calculator (dacă nu l-ați luat din iarnă);

- trebuie să obțineți minim 5 (echivalent cu 2.5 puncte din nota finală) pentru a promova examenul (condiție necesară).

Curs: examen scris

- trebuie să obțineți minim nota 5 (echivalent cu 2.5 puncte din nota finală) pentru a promova examenul;
- rotunjirea notei finale: x.49 devine x, x.50 devine x+1.

Evaluarea la mărire (septembrie)

$$Nota = \min(10, Curs + Laborator + Seminar)$$

5p 5p 1p

Seminar: se păstreaza nota de la activitatea la seminar

Laborator: test de laborator pe calculator (îl dați din nou, obligatoriu)

Curs: examen scris (îl dați din nou, obligatoriu)

- Rotunjirea notei finale: x.49 devine x, x.50 devine x+1.

Evaluarea pentru 2019-2020

Refaceti seminarul + laboratorul si dati examenul final.

Va conformati eventualelor modificari din anul universitar urmator!

Recapitulare – cursul trecut

1. Introducere în limbajul C. Structura unui program C
2. Tipuri de date fundamentale
3. Variabile și constante
4. Expresii și operatori

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

Expresii și operatori

❑ expresii

- ❑ sunt formate din **operanzi** și **operatori**;
- ❑ arată modul de calcul al unor valori;
- ❑ cea mai simplă expresie este formată dintr-un operand;

❑ operatori

- ❑ elemente fundamentale ale expresiilor
- ❑ operatori aritmetici, relaționali, etc.
- ❑ C are foarte mulți operatori (46 în tabelul de la sfârșit)

❑ operanzi

- ❑ variabilă, o constantă
- ❑ apel de funcție
- ❑ expresie între paranteze
- ❑ etc.

Expresii aritmetice și operatori aritmetici

Se aplică asupra unui singur operand

| | | |
|---------------|-----------------------|---|
| Unari | | + (plus unar) - (minus unar) |
| | <i>Aditivi</i> | + (adunare) - (scădere) |
| Binari | | * (înmulțire) / (împărțire) % (restul împărțirii) |
| | <i>Multiplicativi</i> | |

Necesită doi operanzi

Expresii aritmetice și operatori aritmetici

□ exemple

```
int a, b, c = +3;          // operatorul unar +
b = -4;                   // operatorul unar -
a = b - c + 1;            // operatorul binar - și +      a este -6
a = a * b / 2;            // operatorul binar * și /      a este 12
c = a % 5;                // operatorul binar %          c este 2
```

- operatorii aritmetici se pot aplica asupra operanzilor
 - de tip **întreg** (int, char) sau
 - de tip **real** (float sau double)
- se pot **combina** aceste tipuri în aceeași expresie
 - **excepție**: % doar între întregi

Expresii aritmetice și operatori aritmetici

□ observații:

- operatorul / semnifică
 - împărțirea **întreagă** dacă ambii operanzi sunt întregi (int, char)
 - împărțirea **cu virgulă** dacă cel puțin unul dintre operanzi este de tip real (float, double)

```
int a = 5, b = 2;
float x = 5.0f;
a = a / b;      // a este 2
x = x / b;      // x este 2.5
x = 5 / b;      // x este 2.0
```

- împărțirea la zero !!
 - operatorii / și % nu pot avea operandul din dreapta 0
- trunchierea la împărțirea întreagă
 - C89 – dependent de implementare
 - C99 – trunchiere către 0

```
c = 7 / 5;          // c este 1 (trunchiat de la 1.4)
c = -7 / 5;         // c este -1 (trunchiat de la -1.4)
c = 9 / 5;          // c este 1 (trunchiat de la 1.8)
c = 9 / -5;         // c este -1 (trunchiat de la -1.8)
```

Evaluarea expresiilor

- introducem **principii fundamentale** pentru evaluarea oricăror expresii prin intermediul expresiilor aritmetice
 - mai ușor de înțeles astfel
- **precedență și asociativitatea** operatorilor
 - dacă într-o expresie apar mai mulți operatori, atunci evaluarea expresiei respectă **ordinea de precedență** a operatorilor
 - dacă într-o expresie apar mai mulți operatori de aceeași prioritate, atunci se aplică **regula de asociativitate** a operatorilor

Ordinea de precedență

□ ordinea de precedență a operatorilor aritmetici

| | |
|---------------------|---|
| Prioritate crescută | + (plus unar) - (minus unar) |
| | * (înmulțire) / (împărțire) % (restul împărțirii) |
| Prioritate scăzută | + (adunare) - (scădere) |

□ exemple:

$$\begin{array}{l} a + b * c \\ -a * b - c \\ +a - b / c \end{array}$$

este echivalent cu
este echivalent cu
este echivalent cu

$$\begin{array}{l} a + (b * c) \\ ((-a) * b) - c \\ (+a) - (b / c) \end{array}$$

Regula de asociativitate

- regula de asociativitate a operatorilor aritmetici
 - un operator este **asociativ la stânga** dacă se grupează *de la stânga la dreapta*
 - exemple: toți operatorii aritmetici binari (+, -, *, /, %)

| | | |
|-------------|--------------------|---------------|
| $a + b - c$ | este echivalent cu | $(a + b) - c$ |
| $a * b / c$ | este echivalent cu | $(a * b) / c$ |

- un operator este **asociativ la dreapta** dacă se grupează *de la dreapta la stânga*
 - exemple: operatorii aritmetici unari (+, -)

| | | |
|---------|--------------------|----------|
| $- + a$ | este echivalent cu | $- (+a)$ |
| $+ - a$ | este echivalent cu | $+ (-a)$ |

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

Operatori de atribuire

□ operatorul de atribuire simplă =

- efect: evaluarea expresiei din dreapta operatorului și asignarea acestei valori la variabila din stânga operatorului

```
a = 10;           // a ia valoarea 10
b = a;            // b ia valoarea 10
c = a + (b-7) * 3; // c ia valoarea 19
```

□ valoarea unei atribuirii **var = expresie** este valoarea lui var după asignare

- expresia de atribuire poate apărea ca operand într-o altă expresie unde se așteaptă o valoare de tipul var

```
a = 3;
b = 5 - (c = a);      // c ia valoarea 3 care
                      // se scade din 5 și astfel b devine 2
```

- expresia devine greu de înțeles și poate introduce erori greu de depistat

Operatori de atribuire

- atribuirea formalizată: **expr1 = expr2**
 - expr1 este *lvalue* (valoare stânga)
 - trebuie să permită stocarea valorii lui expr2 în memorie
 - corect: $v[i+1] = 10$
 - incorrect: $10 = v[i+1]$
- dacă tipul lui **expr1** și **expr2** nu este același, atunci se aplică regula conversiei implice
 - valoarea lui **expr2** este convertită la tipul lui **expr1** în momentul asignării

```
int a;
float x;
a = 12.34f;           // a ia valoarea 12
x = 123;              // x ia valoarea 123.0
```

Operatori de atribuire

- ❑ regula de asociativitate
 - ❑ operatorul de atribuire este asociativ dreapta
 - ❑ atribuirile se pot înlántui

$$a = b = c = 0$$

- ❑ operatori de atribuire compuși (operator =)
 - ❑ exemplu : `+=`, `-+`, `*=`, `/=`, `%=`, și-md. (combinat cu operatori pe biți)
 - ❑ permit calcularea noii valori a variabilei folosind valoarea veche a acesteia

```
a += 1;           // a se incrementează cu 1: a = a + 1;  
b -= 3;           // asemănător cu b = b - 3;  
c *= 4;           // asemănător cu c = c * 4;
```

- dar nu este întotdeauna echivalent cu varianta descompusă
 - contează ordinea de precedență și efectele secundare

Operatori de incrementare și decrementare

- operatorii **++** și **--**
 - incrementarea/decrementarea unei variabile cu 1
 - specifici limbajelor de asamblare, mult mai rapizi
- forma **prefixă** (**++i** sau **--i**)
 - preincrementare/predecrementare
- forma **postfixă** (**i++** sau **i--**)
 - postincrementare/postdecrementare
- efect secundar: modificarea valorii operandului
- valoarea returnată
 - preincrementarea (**++a**) returnează valoarea **a+1**
 - postincrementarea (**a++**) returnează valoarea **a**

i++;

Exemplu echivalent:

i = i + 1;
i += 1;

Operatori de incrementare și decrementare

```
int a = 5, b = 2, c;  
c = a - ++b;           // ⇔ b = b+1;  c = a-b;  
                      // valorile a: 5, b: 3, c: 2  
c = ++a + b--;        // ⇔ a = a+1;  c = a + b;  b = b-1;  
                      // valorile a: 6, b: 2, c: 9
```

- operatorii de **preincrementare** și **predecrementare** au aceeași prioritate ca și operatorii unari + și - și sunt asociativi dreapta
- operatorii de **postincrementare** și **postdecrementare** au prioritate crescută în raport cu operatorii unari + și - și sunt asociativi stânga

Expresii logice

- ❑ expresiile logice se evaluatează la valori de tip *adevărat* sau *fals*
- ❑ sunt construite cu ajutorul a trei categorii de operatori
 - ❑ operatori **relaționali**
 - ❑ operatori de **egalitate**
 - ❑ operatori **logici**
- ❑ limbajul C tratează valorile *adevărat* și *fals* ca valori întregi
 - ❑ 0 înseamnă fals
 - ❑ orice altă valoare nenulă se interpretează ca adevărat

Operatori relaționali

- ❑ operatorii `<`, `>`, `<=`, `>=`
- ❑ rezultatul este o valoare logică, adică valoarea 0 (fals) sau 1 (adevărat)
- ❑ sunt mai puțin prioritari decât operatorii aritmetici și sunt asociativi stânga

```
5    < 10          // rezultat: 1
10   <  5          // rezultat: 0
3    > 2.5         // rezultat: 1
                  // se pot combina tipurile întreg și real
a + b <= c - 1  // este de fapt (a + b) <= (c - 1)
                  // respectând ordinea de precedență

a < b < c  // echivalent cu (a < b) < c
              // datorita asociativitatii stanga
```

Operatori de egalitate

- testează egalitatea dintre două valori
- **==** este operatorul "egal cu",
- **!=** este operatorul "diferit de"
- generează o valoare logică: 0 (fals) sau 1 (adevărat)
- sunt asociativi stânga
- în ordinea de precedență a operatorilor sunt mai puțin prioritari decât operatorii relaționali

```
a == 2           // returnează 1 dacă a este 2,  
                  // 0 în caz contrar  
a != b           // returnează 1 dacă a nu este egal cu b,  
                  // 0 dacă a și b au valori identice  
a < b == b < c  // este echivalent cu (a < b) == (b < c)  
                  // returnează 1 doar dacă expresiile au  
                  // aceeași valoare:  
                  // ambele sunt adevărate sau ambele false
```

Operatori logici

- limbajul C furnizează 3 operatori logici

- ! - operatorul unar, negare logică

```
!expr      // 1 dacă expr are valoarea logică 0 (fals)
           // 0 dacă expr are valoarea logică nenulă (adevărat)
```

- && - operator binar, **ȘI** logic

```
expr1 && expr2 // este 1 dacă expr1 și expr2 sunt nenule
```

- || - operator binar, **SAU** logic

```
expr1 || expr2 // este 1 dacă expr1 sau expr2 este nenulă
```

- generează o valoare logică: 0 (fals) sau 1 (adevărat)

Operatori logici

- evaluarea
 - dacă se poate deduce rezultatul global din evaluarea expresiei din stânga, atunci expresia din dreapta nu se mai evaluează

```
(a != 0) && (a % 4 == 0)
```

- operatorul ! (negare logică) are prioritate egală cu cea a operatorilor aritmetici unari (+ și -)
- operatorii && și || sunt mai puțin prioritari decât operatorii relaționali și cei de egalitate

Operatori pe biți

- două categorii
 - operatori **logici pe biți**
 - **& ȘI pe biți**, operator binar
 - **| SAU pe biți**, operator binar
 - **^ SAU EXCLUSIV pe biți**, operator binar
 - **~ complement față de 1**, operator unar
 - operatori de **deplasare pe biți**
 - **<< deplasare stânga pe biți**, operator binar
 - **>> deplasare dreapta pe biți**, operator binar
- operanzi de tip întreg (nu merg pe float, double)
- ordinea de precedență - în cadrul acestei categorii

| | |
|---------------------|---------------------------------|
| Prioritate crescută | \sim (complement față de unu) |
| | $<<$ (deplasare stânga) |
| | $>>$ (deplasare dreapta) |
| | $\&$ (și pe biți) |
| | $^$ (sau exclusiv pe biți) |
| Prioritate scăzută | $ $ (sau pe biți) |

Operatori pe biți

- $\&$ seamănă cu $\&\&$
- $|$ seamănă cu $||$
- au un rol similar, dar la nivelul fiecărei perechi de biți de pe poziții identice
- \sim este echivalentul operației $!$ dar aplicat la nivel de biți

| 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 ambele 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 |
| $\sim a$ | 0 | 1 | 0 | 1 | 1 unde bitul a fost 0 și 0 unde bitul a fost 1 |
| $\sim b$ | 1 | 0 | 0 | 0 | 1 unde bitul a fost 0 și 0 unde bitul a fost 1 |

Operatori de deplasare pe biți

- condiții:
 - operanzi întregi
 - al doilea operand cu valoare mai mică (nu negativ) decât numărul de biți pe care este reprezentat operandul din stânga
- deplasarea spre stânga \Leftrightarrow înmulțire cu 2 la puterea deplasamentului
(în anumite condiții)
- deplasarea spre dreapta \Leftrightarrow împărțire cu 2 la puterea deplasamentului
(în anumite condiții)

| 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 biți pe numere negative

Pagina 85 din standardul C99:

©ISO/IEC

ISO/IEC 9899:1999 (E)

- 4 The result of $\mathbf{E1} \ll \mathbf{E2}$ is $\mathbf{E1}$ left-shifted $\mathbf{E2}$ bit positions; vacated bits are filled with zeros. If $\mathbf{E1}$ has an unsigned type, the value of the result is $\mathbf{E1} \times 2^{\mathbf{E2}}$, reduced modulo one more than the maximum value representable in the result type. If $\mathbf{E1}$ has a signed type and nonnegative value, and $\mathbf{E1} \times 2^{\mathbf{E2}}$ is representable in the result type, then that is the resulting value; otherwise, the behavior is undefined.
- 5 The result of $\mathbf{E1} \gg \mathbf{E2}$ is $\mathbf{E1}$ right-shifted $\mathbf{E2}$ bit positions. If $\mathbf{E1}$ has an unsigned type or if $\mathbf{E1}$ has a signed type and a nonnegative value, the value of the result is the integral part of the quotient of $\mathbf{E1}$ divided by the quantity, 2 raised to the power $\mathbf{E2}$. If $\mathbf{E1}$ has a signed type and a negative value, the resulting value is implementation-defined.

Operatori pe biți pe numere negative

operatiiBitiNumereNegative1.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determină scrierea în baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u" , (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
19 int main()
20 {
21
22     int a = -1;
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = a<<2;
30     printf("%d \n",a);
31     afiseazaScriereBinara(a);
32
33     a = -1;
34     a = a>>1;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38     a = a>>2;
39     printf("%d \n",a);
40     afiseazaScriereBinara(a);
41
42
43     return 0;
44 }
```

Operatori pe biți pe numere negative

```
operatiiBitiNumereNegative1.c  x

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u" , (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
19 int main()
20 {
21
22     int a = -1;
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = a<<2;
30     printf("%d \n",a);
31     afiseazaScriereBinara(a);
32
33     a = -1;
34     a = a>>1;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38     a = a>>2;
39     printf("%d \n",a);
40     afiseazaScriereBinara(a);
41
42
43     return 0;
44 }
```

Operatori pe biți pe numere negative

```
operatiiBitiNumereNegative2.c ●
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u", (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
18 }
```

```
19 int main()
20 {
21
22     int a = -(int) pow(2,31);
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29
30     a = -(int) pow(2,31);
31     a = a>>1;
32     printf("%d \n",a);
33     afiseazaScriereBinara(a);
34
35     a = a>>2;
36     printf("%d \n",a);
37     afiseazaScriereBinara(a);
38
39
40
41 }
```

Operatori pe biți pe numere negative

operatiiBitiNumereNegative2.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5
6 void afiseazaScriereBinara(int x)
7 {
8     //determina scrierea in baza 2
9     printf("Scrierea binara a lui %d este \n",x);
10    unsigned long long m = 1ULL << (8*sizeof(int)-1);
11    unsigned char b;
12    for(b = 0; b < 8*sizeof(int); b++){
13        printf("%u", (x & m) != 0);
14        m = m >> 1;
15    }
16    printf("\n");
17 }
```

```
Scrierea binara a lui -2147483648 este
10000000000000000000000000000000
0
Scrierea binara a lui 0 este
00000000000000000000000000000000
-1073741824
Scrierea binara a lui -1073741824 este
11000000000000000000000000000000
-268435456
Scrierea binara a lui -268435456 este
11110000000000000000000000000000
```

```
19 int main()
20 {
21
22     int a = -(int) pow(2,31);
23     afiseazaScriereBinara(a);
24
25     a = a<<1;
26     printf("%d \n",a);
27     afiseazaScriereBinara(a);
28
29     a = -(int) pow(2,31);
30     a = a>>1;
31     printf("%d \n",a);
32     afiseazaScriereBinara(a);
33
34     a = a>>2;
35     printf("%d \n",a);
36     afiseazaScriereBinara(a);
37
38
39     return 0;
40 }
```

Alți operatori

- ❑ operatorul de acces la elementele tabloului []

```
int a[100];  
a[5] = 10;
```
- ❑ operatorul de apel de funcție (): b = f(a);
- ❑ operatorul **adresă** & și operatorul de **dereferețiere** *
 - ❑ strâns legat de pointeri (cursurile următoare)

```
int a, *p;           // p este un pointer la int  
p = &a;             // p este pointer la a  
*p = 3;             // valoarea lui a devine 3
```

- ❑ operatorul **sizeof**

```
sizeof(a)           // este numărul de octeți  
                      // ocupări în memorie de a
```
- ❑ operatorul de conversie explicită: (tip)

```
int a = 1, b = 2;  
float media;  
  
media = ( a + (float)b ) / 2;    // media devine 1.5  
media = ( a + b ) / 2;          // media devine 1.0 - incorect!
```

Alți operatori

❑ operatorul condițional ? :

- ❑ operator ternar, decizional
- ❑ similar cu instrucțiunea **if**
- ❑ **expresie1? expresie2 : expresie3**
- ❑ dacă **expresie1** e adevarată, execută **expresie2**, altfel execută **expresie3**

```
int a=3, b=5, max;  
max = a > b ? a : b;  
  
a % 2 ? printf("numar impar") : printf("numar par");
```

❑ operatorul virgulă

- ❑ evaluarea secvențială a expresiilor (de la stânga la dreapta)
- ❑ valoarea ultimei expresii din înlănțuire este valoarea expresiei compuse
- ❑ cel mai puțin priorită din lista de precedență

```
int i, n, s;  
printf("n = ");scanf("%d",&n);  
for(i = 1,s = 0;i <= n;s = s + i,i = i + 1);
```

Ordinea de precedență și asociativitate

| precedență | operatori | simbol | asociativitate |
|------------|-------------------------|------------------------|----------------|
| 1 | apel funcție / selecție | () [] . -> | SD |
| 2 | unari | * & - ! ~ ++ -- sizeof | DS |
| 3 | multiplicativi | * / % | SD |
| 4 | aditivi | + | SD |
| 5 | deplasări | << >> | SD |
| 6 | relaționali | < > <= >= | SD |
| 7 | egalitate / neegalitate | = != | SD |
| 8 | ȘI pe biți | & | SD |
| 9 | SAU exclusiv pe biți | ^ | SD |
| 10 | SAU inclusiv pe biți | | SD |
| 11 | ȘI logic | && | SD |
| 12 | SAU logic | | SD |
| 13 | condițional | ? : | DS |
| 14 | atribuire | = op= | DS |
| 15 | virgula | , | SD |

Laboratorul 1 – rezolvări

2. Se citesc trei numere întregi de la tastatură. Să se afișeze maximul dintre cele 3 numere folosind operatorul decizional.
3. Se citește un număr întreg n de la tastatură. Să se calculeze $n*8$, $n/4$ și $n*10$ folosind operatorii logici de deplasare la nivel de bit.
4. Se citește un număr întreg de la tastatură. Să se determine dacă acesta este par sau impar folosind doar operatorii logici la nivel de biți.

```
solutiiLaborator1.c  x
1 #include <stdio.h>
2
3 int main()
4 {
5     //problema 2
6     int x,y,z;
7     scanf("%d%d%d",&x,&y,&z);
8     x > y ? x > z ? printf("x = %d e cel mai mare \n", x) : printf("z = %d e cel mai mare\n",z) :
9         y > z? printf("y = %d e cel mai mare \n", y) : printf("z = %d e cel mai mare\n",z);
10
11    //problema 3
12    int n;
13    scanf ("%d",&n);
14    printf("%d*8 = %d\n%d/4 = %d\n%d*10=%d\n",n,n<<3,n,n>>2,n, (n<<3) + (n<<1));
15
16    //problema 4
17    scanf("%d",&n);
18    n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20    //rezolvare gresita
21    n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23    return 0;
24 }
```

Laboratorul 1 – rezolvări

2. Se citesc trei numere întregi de la tastatură. Să se afișeze maximul dintre cele 3 numere folosind operatorul decizional.
3. Se citește un număr întreg n de la tastatură. Să se calculeze $n*8$, $n/4$ și $n*10$ folosind operatorii logici de deplasare la nivel de bit.
4. Se citește un număr întreg de la tastatură. Să se determine dacă acesta este par sau impar folosind doar operatorii logici la nivel de biți.

```
solutiiLaborator1.c  x
1 #include <stdio.h>
2
3 int main()
[Bogdan-Alexes-MacBook-Pro:diverse bogdan$ gcc solutiiLaborator1.c
solutiiLaborator1.c:21:4: warning: | has lower precedence than ==; == will be
      evaluated first [-Wparentheses]
      n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
           ^~~~~~
solutiiLaborator1.c:21:4: note: place parentheses around the '==' expression to
      silence this warning
      n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
           ^
16 //problema 4
17 scanf("%d",&n);
18 n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20 //rezolvare gresita
21 n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23 return 0;
24 }
```

Laboratorul 1 – rezolvări

```
solutiiLaborator1.c    x

1 #include <stdio.h>
2
3 int main()
4 {
5     //problema 2
6     int x,y,z;
7     scanf("%d%d%d",&x,&y,&z);
8     x > y ? x > z ? printf("x = %d e cel mai mare \n", x) : printf("z = %d e cel mai mare\n",z) :
9         y > z? printf("y = %d e cel mai mare \n", y) : printf("z = %d e cel mai mare\n",z);
10
11    //problema 3
12    int n;
13    scanf ("%d",&n);
14    printf("%d*8 = %d\n%d/4 = %d\n%d*10=%d\n",n,n<<3,n,n>>2,n, (n<<3) + (n<<1));
15
16    //problema 4
17    scanf("%d",&n);
18    n & 1 ? printf("%d e impar \n",n) : printf("%d e par \n",n);
19
20    //rezolvare gresita
21    n | 1 == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
22
23    //rezolvare corecta
24    (n | 1) == n ? printf("%d e impar \n",n) : printf("%d e par \n",n);
25
26    return 0;
27 }
```

Seminarul 2

- tema seminarului este operatori pe biți;
- are 9 pagini ☺;
- 7 probleme rezolvate (unele au și cod soluție);
- 9 probleme propuse studentilor (acestea se vor discuta la seminar);
- o să presupunem că ati citit problemele rezolvate ☺ (eventual puteti adresa intrebări legate de ele).

Cuprinsul cursului de azi

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

Conversii implicate și explicite

```
conversii.c
1 #include <stdio.h>
2
3 int main(){
4
5     int i;
6     i = 1.6 + 1 + 1.7;
7     printf("i = %d \n", i);
8     i = (int)1.6 + 1 + (int)1.7;
9     printf("i = %d \n", i);
10
11    char a = 30, b = 40, c = 10;
12    char d = (a*b)/c;
13    printf("%d \n",d);
14
15    unsigned int ui_one = 1;
16    int i_one = 1;
17    short sh_minus_one = -1;
18    if(sh_minus_one > ui_one)
19        printf("-1 > 1 \n");
20    if(sh_minus_one < i_one)
21        printf("-1 < 1 \n");
22
23    return 0;
24 }
```

Ce afișează
programul alăturat?

i = 4
i = 3
120
-1 > 1
-1 < 1

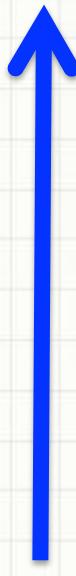
Conversii implicate

- context
 - este permisă combinarea mai multor operanzi de tipuri diferite într-o singură expresie
- problema
 - operatorii binari, care se aplică asupra a doi operanzi, cer ca **tipul operanzilor să fie același** pentru a putea efectua operația
- soluție: conversia implicită
 - compilatorul **convertește valorile operanzilor la același tip într-un mod transparent** programatorului înaintea generării codului mașină
 - există reguli de conversie implicită
- alternativă
 - conversii explicite: (tip)

Conversii implicate - reguli

- ❑ 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 doi operanzi, ambii operanzi sunt convertiți la tipul de date cel mai înalt în ierarhie
- ❑ ierarhia tipurilor de date:
(nu există char și short)
 - ❑ 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



Conversii implicite - reguli

□ conversii implicite la atribuire

- valoarea expresiei din dreapta se convertește la tipul expresiei din stânga
 - pot apărea pierderi – dacă tipul nu este suficient de încăpător

```
char c = 'a';
short sh = 140;
int a = 3, b;
unsigned int u = 1234567u;
long i = 300L;
float f = 80.13f;
double d = 5.75, g;
```

```
b = a + sh;    // val. lui sh convertita la int
a = sh - c;    // val. lui sh si c convertite la int
g = d + f;    // val. lui f convertita la double
f = i + u;    // cal lui u convertita la long
              // rezultatul convertit la float
```

Cuprinsul cursului de azi

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

Pointeri

- **pointer** = tip de date derivat folosit pentru manipularea adreselor de memorie
- **sintaxa**

tip *nume_variabilă;

 - **tip** = tipul de bază al variabilei de tip pointer nume_variabilă
 - ***** = operator de indirectare
 - **nume_variabila** = variabila de tip pointer care poate lua ca valori adrese de memorie
- **cel mai puternic mecanism de accesare a memoriei în C**

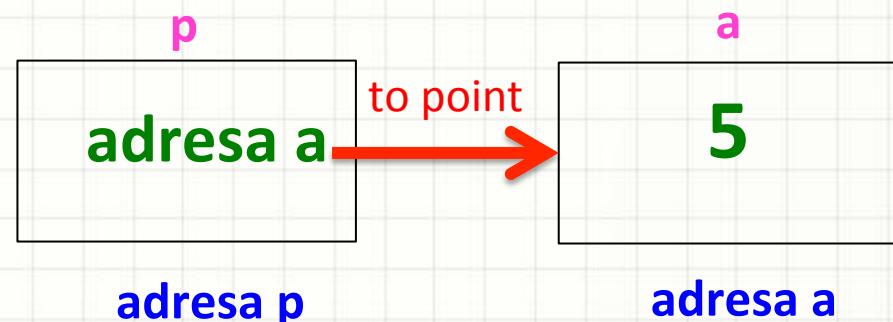
Pointeri

- pointer = tip de date derivat folosit pentru manipularea adreselor de memorie
- operatori pentru manipularea adreselor de memorie:
 - & - operatorul de referențiere
 - **&variabila** – furnizează adresa variabilei respective
 - * - operatorul de dereferențiere
 - ***variabila_de_tip_pointer** – furnizează valoarea aflată la adresa de memorie stocată în variabila_de_tip_pointer

Pointeri

□ exemplu:

```
int a=5;  
int *p;  
p = &a;
```

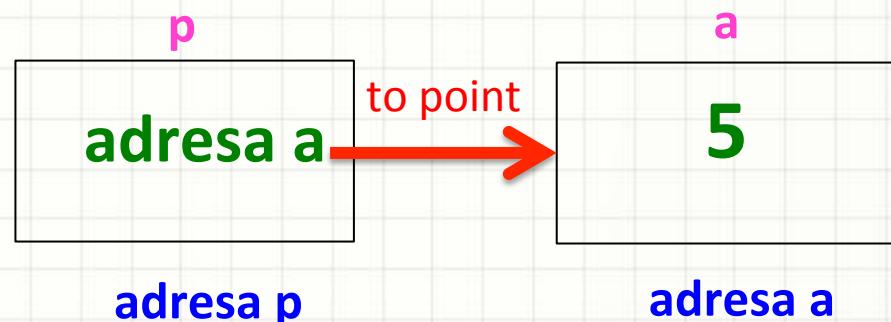


```
pointer1.c  
x  
1 #include <stdio.h>  
2  
3 int main() {  
4  
5     int a = 5, *p;  
6     p = &a;  
7     printf("Adresa lui a este: %p \n", &a);  
8     printf("Valoarea stocata la adresa lui a este: %d \n",a);  
9  
10    printf("Adresa lui p este: %p \n",&p);  
11    printf("Valoarea stocata la adresa lui p este: %p \n",p);  
12    printf("Valoarea variabilei a carei adresa e stocata in p este: %d \n",*p);  
13  
14    return 0;  
15 }
```

Pointeri

□ exemplu:

```
int a=5  
int *p;  
p = &a;
```



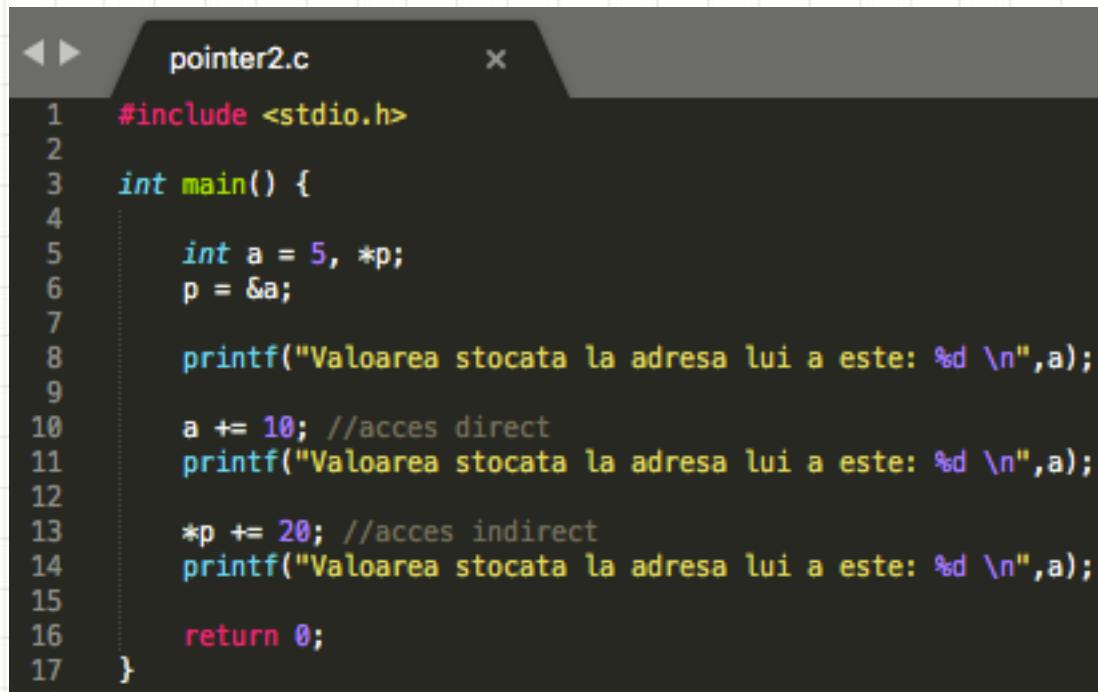
```
pointer1.c
```

```
1 #include <stdio.h>  
2  
3 int main() {  
4  
5     int a = 5, *p;  
6     p = &a;  
7     printf("Adresa lui a este: %p \n", &a);  
8     printf("Valoarea stocata la adresa lui a este: %d \n",a);  
9  
10    printf("Adresa lui p este: %p \n",&p);  
11    printf("Valoarea stocata la adresa lui p este: %p \n",p);  
12    printf("Valoarea variabilei a carei adresa e stocata in p este: %d \n",*p);  
13  
14    return 0;  
15 }
```

```
Adresa lui a este: 0x7fff5a2eab58  
Valoarea stocata la adresa lui a este: 5  
Adresa lui p este: 0x7fff5a2eab50  
Valoarea stocata la adresa lui p este: 0x7fff5a2eab58  
Valoarea variabilei a carei adresa e stocata in p este: 5
```

Pointeri

- exemplu: modificarea valorii unei variabile prin dereferențiere

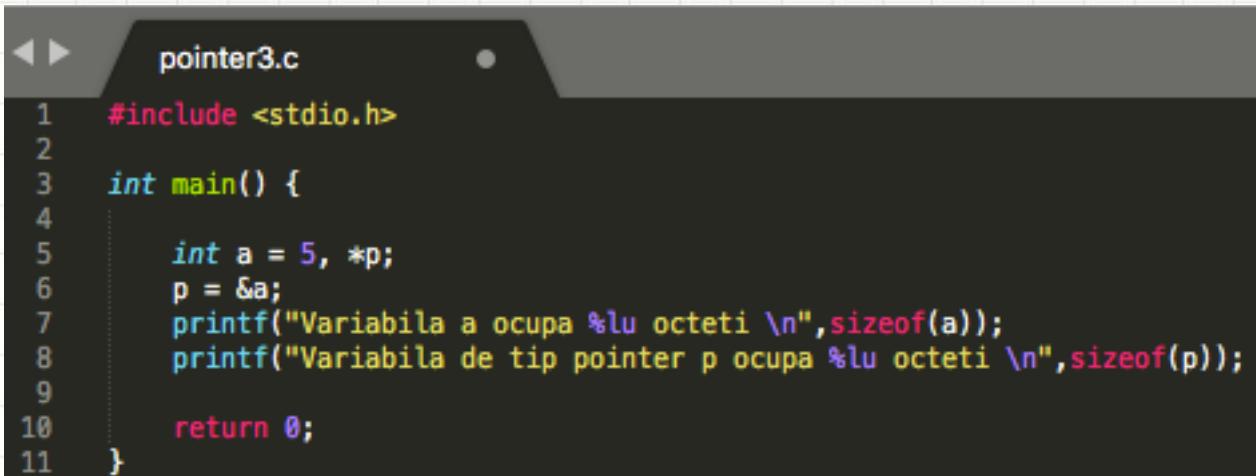


```
pointer2.c
1 #include <stdio.h>
2
3 int main() {
4
5     int a = 5, *p;
6     p = &a;
7
8     printf("Valoarea stocata la adresa lui a este: %d \n",a);
9
10    a += 10; //acces direct
11    printf("Valoarea stocata la adresa lui a este: %d \n",a);
12
13    *p += 20; //acces indirect
14    printf("Valoarea stocata la adresa lui a este: %d \n",a);
15
16    return 0;
17 }
```

```
Valoarea stocata la adresa lui a este: 5
Valoarea stocata la adresa lui a este: 15
Valoarea stocata la adresa lui a este: 35
```

Pointeri

- exemplu: spațiul de memorie ocupat de o variabilă de tip pointer



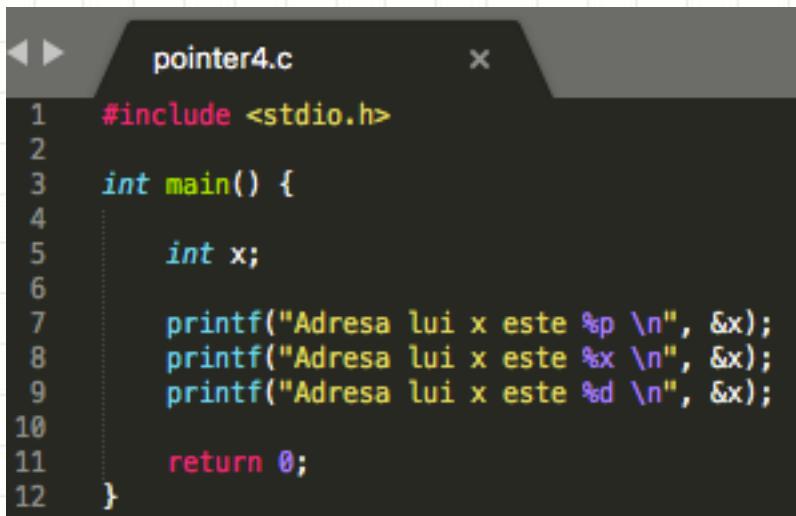
```
pointer3.c
1 #include <stdio.h>
2
3 int main() {
4
5     int a = 5, *p;
6     p = &a;
7     printf("Variabila a ocupa %lu octeti \n",sizeof(a));
8     printf("Variabila de tip pointer p ocupa %lu octeti \n",sizeof(p));
9
10    return 0;
11 }
```

Variabila a ocupa 4 octeti
Variabila de tip pointer p ocupa 8 octeti

Pointeri

□ adrese de memorie

- În C există un specificator de format special (%p) pentru tipărirea valorilor reprezentând adresele de memorie.



```
#include <stdio.h>
int main() {
    int x;
    printf("Adresa lui x este %p \n", &x);
    printf("Adresa lui x este %x \n", &x);
    printf("Adresa lui x este %d \n", &x);
    return 0;
}
```

Adresa lui x este 0xffff5cca4b58
Adresa lui x este 5cca4b58
Adresa lui x este 1556761432

Pointeri

□ adrese de memorie

- În C există un specificator de format special (%p) pentru tipărirea valorilor reprezentând adresele de memorie.

The screenshot shows a browser window with the title "Base-10 to Base-16 Conve X". The address bar contains the URL "www.unitconversion.org/numbers/base-10-to-base-16-conversion.html". Below the address bar, there are two input fields. The first input field is labeled "base-10:" and contains the value "1556761432". The second input field is labeled "base-16:" and contains the value "5CCA4B58".

Adresa lui x este 0xffff5cca4b58
Adresa lui x este 5cca4b58
Adresa lui x este 1556761432

Cuprinsul cursului de azi

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

Tablouri unidimensionale

- **sintaxă:**

tip nume_tablou [dimensiune];

- **exemple:**

double v[100];

| | | | | | | | |
|-----|------|----|-----|-----|-----|------|----|
| 0.3 | -1.2 | 10 | 5.7 | ... | 0.2 | -1.5 | 1 |
| 0 | 1 | 2 | 3 | | 97 | 98 | 99 |

v[3] = 5.7;

int a[5];

| | | | | |
|---|-----|----|---|---|
| 3 | -12 | 10 | 7 | 1 |
| 0 | 1 | 2 | 3 | 4 |

a[0] = 3;

char c[34];

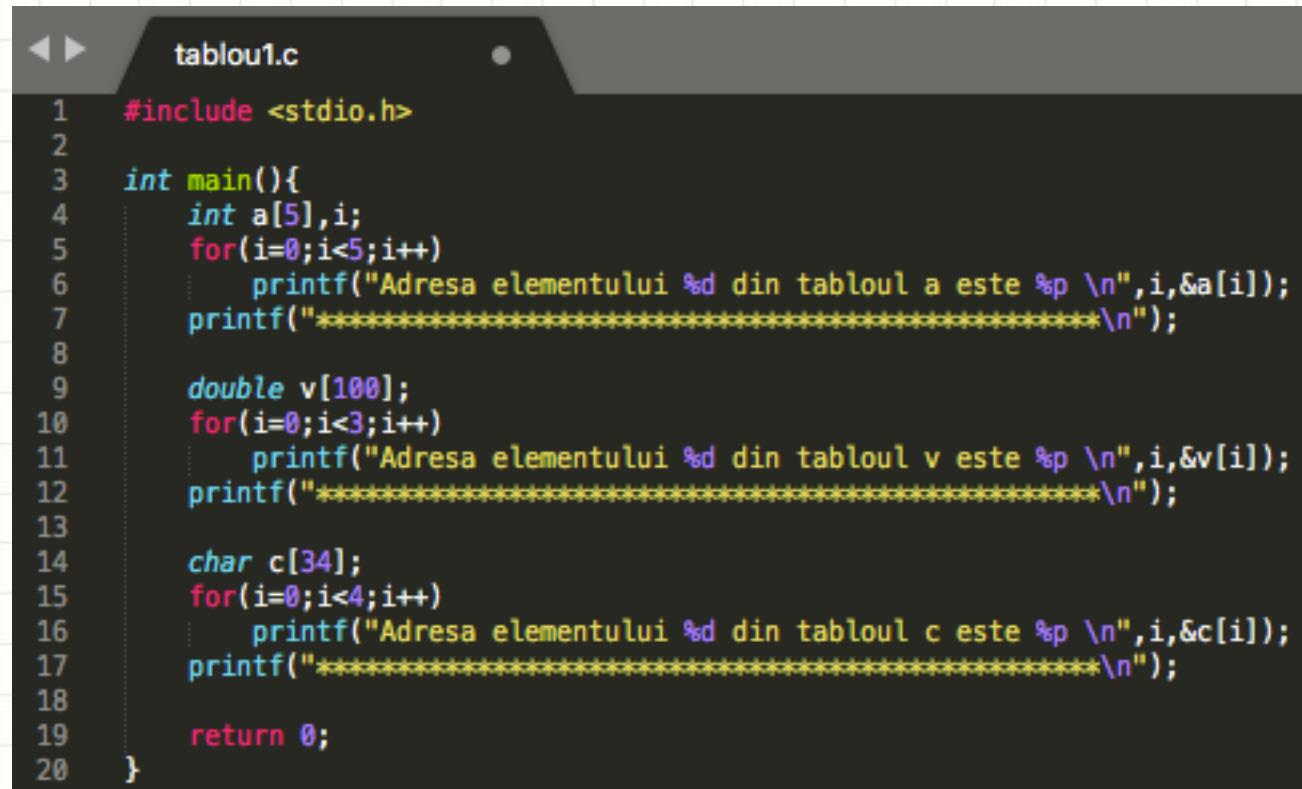
| | | | | | | | |
|---|---|---|---|-----|----|----|----|
| A | & | * | + | ... | c | M | # |
| 0 | 1 | 2 | 3 | | 31 | 32 | 33 |

c[1] = '&';

- În C, primul element al unui tablou are indicele 0.

Tablouri unidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.



```
tablou1.c
1 #include <stdio.h>
2
3 int main(){
4     int a[5],i;
5     for(i=0;i<5;i++)
6         printf("Adresa elementului %d din tabloul a este %p \n",i,&a[i]);
7     printf("*****\n");
8
9     double v[100];
10    for(i=0;i<3;i++)
11        printf("Adresa elementului %d din tabloul v este %p \n",i,&v[i]);
12    printf("*****\n");
13
14    char c[34];
15    for(i=0;i<4;i++)
16        printf("Adresa elementului %d din tabloul c este %p \n",i,&c[i]);
17    printf("*****\n");
18
19    return 0;
20 }
```

Tablouri unidimensionale

- definiție: set de valori de același tip memorat la adrese succesive de memorie.

```
tablou1.c
1 #include <stdio.h>
2
3 int main(){
4     int a[5],i;
5     for(i=0;i<5;i++)
6         printf("Adresa elementului %d din tabloul a este %p \n",i,&a[i]);
7     printf("*****\n");
8
9     double v[100];
10    for(i=0;i<3;i++)
11        printf("Adresa elementului %d din tabloul v a este %p \n",i,&v[i]);
12    printf("*****\n");
13
14     char c[34];
15     for(i=0;i<4;i++)
16         printf("Adresa elementului %d din tabloul c a este %p \n",i,&c[i]);
17     printf("*****\n");
18
19     return 0;
20 }
```

Adresa elementului 0 din tabloul a este 0xffff52d5bb40
Adresa elementului 1 din tabloul a este 0xffff52d5bb44
Adresa elementului 2 din tabloul a este 0xffff52d5bb48
Adresa elementului 3 din tabloul a este 0xffff52d5bb4c
Adresa elementului 4 din tabloul a este 0xffff52d5bb50

Adresa elementului 0 din tabloul v este 0xffff52d5b820
Adresa elementului 1 din tabloul v este 0xffff52d5b828
Adresa elementului 2 din tabloul v este 0xffff52d5b830

Adresa elementului 0 din tabloul c este 0xffff52d5b7f0
Adresa elementului 1 din tabloul c este 0xffff52d5b7f1
Adresa elementului 2 din tabloul c este 0xffff52d5b7f2
Adresa elementului 3 din tabloul c este 0xffff52d5b7f3
