Limbaje de Programare
Curs 2 – Decizii. Tipul caracter. Recursivitate

Dr. Casandra Holotescu

Universitatea Politehnica Timișoara

Ce discutăm azi...

1. Operatori, condiții și decizii
2. Tipul caracter
3. Recursivitate

Operatori aritmetici

Operator	Semnificație	Exemplu expresie
+	adunare	1 + 2
++	adună 1 la valoarea unei variabile	x++ sau ++x
-	scadere	5 - 3
- -	scade 1 din valoarea unei variabile	x - - sau - - x
*	înmulțire	3 * 4
/	împărțire (cu rest / exactă)	8 / 4
%	rest împărțire	5 % 3

Obs. : % se poate folosi doar pt. numere întregi

Precizări – operatori aritmetici

Împărțirea cu rest (modulo):

$\mathbf{7\%2}$ va fi $\mathbf{1}$, dar $-\mathbf{7\%2}$ va fi $-\mathbf{1}$
$\mathbf{7\% - 2}$ va fi $\mathbf{1}$, dar $-\mathbf{7\% - 2}$ va fi $-\mathbf{1}$
Restul are semnul deîmpărțitului.

Împărțirea întreagă vs. exactă:

$\mathbf{7/2}$ va fi $\mathbf{3}$ (câtul – împărțire întreagă)
$\mathbf{7.0/2.0}$ va fi $\mathbf{3.5}$ (împărțire reală)

dar...
$\mathbf{7/2.0}$ va fi $\mathbf{3.5}$ (7 este convertit implicit in 7.0)
$\mathbf{7.0/2}$ va fi $\mathbf{3.5}$ (2 este convertit implicit in 2.0)

Operatori aritmetici și conversii

Atunci când trebuie să împărțim exact un număr întreg, trebuie să îl convertim întâi în real.

Putem face asta în mod explicit:

```
double jumatate (int x){
 return ((double) x)/2; //conversie explicita
}
```

Sau în mod implicit:

```
double jumatate (int x){
 return (x*1.0)/2; //x*1.0 va fi double
}
```

Conversii între tipuri

Conversia **explicită**:

prin folosirea *operatorului de conversie*:

```
(tip) expresie
```

Conversia **implicită** a tipurilor numerice:

are loc la compilare, când doi operanzi au tipuri diferite

tipul comun = tipul care poate conține valorile **ambilor** operanzi

posibile conversii implicite:

int → unsigned int → long → unsigned long → long long

long long → float → double → long double

Operatori relaționali

Operator	Semnificație	Exemplu expresie
$==$	egalitate	$x == 2$
$!=$	inegalitate	$x != 1$
$>$	mai mare	$x > 0$
$<$	mai mic	$x < 3$
$>=$	mai mare sau egal	$x >= 3$
$<=$	mai mic sau egal	$x <= 0$

Atenție! Nu confundați operatorul de atribuire ($=$) cu cel relațional de egalitate ($==$)!

Operatorii relaționali produc expresii care pot fi **adevărate** sau **false**, si care pot fi folosite drept **condiții** în luarea de **decizii**.

Valori de adevăr în C

Noțiunea de **adevărat/fals** este foarte importantă în programare.

În multe limbaje de programare există tipul **boolean**, care este folosit pentru exprimarea acestei valori. În acest caz, tipul boolean are doar două valori: true (adevărat) și false (fals).

În ANSI C **nu** dispunem de un tip boolean **explicit**. Valorile de adevăr sunt exprimate cu ajutorul tipurilor **întregi**.

Astfel, o valoare întreagă este interpretată ca:

- **true/adevărat** dacă e **diferită de 0**
- **false/fals** dacă e **egală cu 0**

Exemple – valori de adevăr

Putem atribui valoarea rezultată din compararea a două numere (o valoare de adevăr), unei variabile întregi:

```
int compar (double x){
 int rezultat = (x>=2.3);
 return rezultat;
}
```

Dacă $x >= 2.3$, funcția va returna 1 (valoarea întreagă standard pentru codificarea valorii de adevăr true).

Dacă $x < 2.3$, funcția va returna 0.

Operatori logici

Operator	Semnificație	Exemplu expresie
!	negație	!(x == 2)
&&	și logic	(x != 1) && (x != 2)
||	sau logic	(x == 1) || (x == 2)

$e1$ && $e2$ && $e3$: $e1$ falsă $\Rightarrow$ $e1$ && $e2$ && $e3$ falsă.

$e1$ || $e2$ || $e3$: $e1$ adevărată $\Rightarrow$ $e1$ || $e2$ || $e3$ adevărată.

În ambele cazuri ar fi inutilă evaluarea $e2, e3$, etc.

Operatorii && și || se evaluează în **scurt-circuit**:
evaluarea se **oprește** la prima valoare **false** (0) pt &&
și la prima valoare **true** ($! = 0$) pentru ||.

Exemple – operatori logici

```
int exem1 (double x){
 int rezultat = (x>=2.3) && (x<=5.5);
 return rezultat;
}
```

Dacă x e 2.1, se evaluează doar expresia $(x >= 2.3)$, iar $(x <= 5.5)$ **nu mai este evaluată**.

```
int exem2(double x){
 int rezultat = (x==1.0) || (x>=2.5) || (x<0.7);
 return rezultat;
}
```

Dacă x e 1.0, se evaluează doar prima expresie $(x == 1.0)$, iar $(x >= 2.5)$ și $(x < 0.7)$ **nu mai sunt evaluate**.

Operatorul condițional

O expresie condițională în C are sintaxa:

conditie? expr-true : expr-false

Valoarea expresiei condiționale e dată de:

- condiția e adevărată $\Rightarrow$ valoarea expresiei expr-true
- condiția e falsă $\Rightarrow$ valoarea expresiei expr-false

Dacă expr-true și expr-false au tipuri diferite, are loc conversia implicită la un tip comun.

Exemple – operatorul condițional

Funcția modul (*abs*, declarată în *stdlib.h*):

```
int abs(int x){
 return x >= 0? x : -x;
}
```

Se evaluează întâi condiția $x >= 0$.

Dacă valoarea primită ca parametru e $>= 0$, atunci se returnează x, altfel $-x$.

Precedența operatorilor

Știm deja că înmulțirea și împărțirea au precedența mai mică decât adunarea și scăderea (se execută înaintea acestora).

Precedența operatorilor învățați până acum:

Precedență	Operatori	Asociativitate
1	++, - - (postfix)	la stânga
2	++, - - (prefix), !, (tip)	la dreapta
3	*, /, %	la stânga
4	+, -	la stânga
6	<, <=, >, >=	la stânga
7	==, !=	la stânga
11	&&	la stânga
12	||	la stânga
13	? :	la dreapta
14	=	la dreapta

Expresii și apeluri de funcții

Fie un apel de funcție în care dăm ca parametri expresii:

```
abs(3*4+3);
medie(7+4, 7*7);
abs(medie(abs(-3-9), abs(-7*2)));
```

Funcția, pentru a se lansa în execuție, are nevoie să cunoască **valorile tuturor argumentelor** sale.

Astfel, **întâi se evaluează toate expresiile** date ca parametru, se obțin valorile corespunzătoare și abia **apoi se execută apelul** la funcție!

În C o funcție lucrează numai cu **valori**, nu cu expresii neevaluate!

Instrucțiunea condițională

Ne permite să efectuăm [o serie de] acțiuni diferite în funcție de rezultatul evaluării unei condiții:

```
if(conditie)
    instructiune_cond_adevarata;
 else
    instructiune_cond_falsa;
```

Funcția modul rescrisă cu instrucțiunea condițională:

```
int abs(int x){
 if (x >= 0)
    return x;
 else
    return -x;
}
```

Instrucțiunea condițională – precizări

Atenție: O funcție de tip **non-void** trebuie sa returneze o valoare in **toate cazurile**!

Programul *va compila* (cu avertismente...), dar valoarea returnata de funcție pe ramura respectivă rămâne **nedefinită**, poate fi **orice**!

```
int bad_abs(int x){
 if (x >= 0)
   return x;
 //incorect, dar va compila...
 //si daca x<0 ???
}
...
int a = bad_abs(-7); //in main
// ce valoarea va avea a??
```

Instrucțiunea switch

Când avem mai multe cazuri posibile (**valori constante**):

```
switch(expresie){ //se evalueaza expresia
  case valoare_1:
        instructiuni1;
        break; //altfel ar continua...

  case valoare_2: //se sare la valoarea care
        instructiuni2; // corespunde si se
        break;         //executa instructiunile
  ...
  default:  //nicio valoare nu se potriveste
        instructiuni_default;

  //cazul default poate lipsi
}
```

Exemple – instrucțiunea switch

```
int sgn(int x, int sign){
    int newx=0;
    switch(sign){
        case -1:{
            newx = -x;
            break; //fara break ar continua
        }     // cu urmatoarea instructiune

        case 0: // grupam cele 2 valori impreuna
        case 1: // aceeasi instructiune
            newx = x;
    }
    return newx;
}
```

Tipul caracter

Reprezentarea caracterelor în C: tipul **char**.

În C, constantele caracter se scriu între apostroafe:
'A', 'c', '9', '#', '!', etc.

Fiecare caracter este de fapt un **cod numeric** în tabela ASCII (asociaza simbol – cod numeric) $\Rightarrow$ **char = tip întreg**!

De exemplu: 'a' == 97, 'b' == 98, 'A' == 65, etc.

Domeniul de valori al **char** (se poate memora pe **1 octet – 8 biți**):

- **unsigned char** 0 – 255
- **signed char** -128 – 127

Ambele sunt incluse în domeniul de valori al tipului int.

Tabela ASCII

ASCII = American Standard Code for Information Interchange

0	*NUL*	16	*DLE*	32	SPC	48	0	64	@	80	P	96	`	112	p
1	*SOH*	17	*DC1*	33	!	49	1	65	A	81	Q	97	a	113	q
2	*STX*	18	*DC2*	34	"	50	2	66	B	82	R	98	b	114	r
3	*ETX*	19	*DC3*	35	#	51	3	67	C	83	S	99	c	115	s
4	*EOT*	20	*DC4*	36	$	52	4	68	D	84	T	100	d	116	t
5	*ENQ*	21	*NAK*	37	%	53	5	69	E	85	U	101	e	117	u
6	*ACK*	22	*SYN*	38	&	54	6	70	F	86	V	102	f	118	v
7	*BEL*	23	*ETB*	39	'	55	7	71	G	87	W	103	g	119	w
8	*BS*	24	*CAN*	40	(	56	8	72	H	88	X	104	h	120	x
9	*HT*	25	*EM*	41	)	57	9	73	I	89	Y	105	i	121	y
10	*LF*	26	*SUB*	42	*	58	:	74	J	90	Z	106	j	122	z
11	*VT*	27	*ESC*	43	+	59	;	75	K	91	[	107	k	123	{
12	*FF*	28	*FS*	44	,	60	<	76	L	92	\	108	l	124	|
13	*CR*	29	*GS*	45	-	61	=	77	M	93	]	109	m	125	}
14	*SO*	30	*RS*	46	.	62	>	78	N	94	^	110	n	126	~
15	*SI*	31	*US*	47	/	63	?	79	O	95	_	111	o	127	DEL

Tipul caracter – continuare

Cifrele, literele mari/mici ocupă zone **continue** în tabela ASCII
⇒ literele și cifrele consecutive au coduri ASCII **consecutive**:
Ex.: 'a' == 97, 'b' == 98, 'c' == 99, etc.

În calcule, valorile de tip caracter sunt convertite automat la întreg.

'a' + 2 == 'c' 'P' - 'M' == 3 'F' + ('a'-'A') == 'f'
'0' + 7 == '7' '9' - '1' == 8

Reprezentări pentru caractere speciale:

'\0'	null	'\n'	linie nouă
'\a'	alarm	'\r'	carriage return
'\b'	backspace	'\f'	feed form
'\t'	tab	'\' '	apostrof
'\v'	vertical tab	'\\'	backslash

Exemple

```
//recunoaste literele mici
 int islower(char c){
  if (c >= 'a' && c <= 'z')
      return 1; //e litera mica
   return 0;     //nu e litera mica
 }


//valoarea numerica a cifrei
 int digitValue(char c){
  if (c >= '0' && c <= '9')
      return c - '0';
   return -1;   //sau -1 daca nu e cifra
 }
```

Citirea de caractere

Putem citi de la tastatură un caracter cu funcția **getchar()** declarată in **stdio.h** cu antetul **int getchar(void)**.

Un apel la **getchar()** poate returna:

- codul ASCII al caracterului citit ca **unsigned char**
- sau... **EOF** (end-of-file): sfârșit de fișier (constantă întreagă cu valoarea **-1**, poate fi introdusă de la tastatură: CTRL+D (Unix/Linux), CTRL+Z (Windows))

Pentru a putea exprima și valoarea EOF, nu doar codurile ASCII, **tipul returnat** de getchar() este **int**!

Obs.: Caracterele se citesc într-o zonă tampon, programul le preia doar după apăsarea tastei *Enter*.

Scrierea de caractere

Putem scrie la ieșire un caracter cu funcția **putchar(caracter)** declarată in **stdio.h** cu antetul **int putchar(int c)**.

Funcția scrie un **unsigned char** primit ca int și returnează valoarea pe care tocmai a scris-o.

Ex.: putchar('a') va returna caracterul 'a' (adică 97)

```
#include <stdio.h>

int main(void) {
    putchar('A'); putchar(':'); // scrie A apoi :
    putchar(getchar()); // scrie caracterul citit
    return 0;
}
```

Exemple

```
#include <stdio.h>

int main(void) {
  int c = getchar(); //citim caracterul urmator

  if(c != EOF){    //daca nu e sfarsit de fisier
      putchar(c); //tiparim caracterul

   }else{
      putchar('E');
      putchar('O');
      putchar('F');
   }
  return 0;
 }
```

Câteva funcții din ctype.h

int isalnum(int c) – recunoaște litere și cifre
int isalpha(int c) – recunoaște litere
int isblank(int c) – recunoaște spațiu și tab
int isdigit(int c) – recunoaște cifre
int islower(int c) – recunoaște litere mici
int ispunct(int c) – recunoaște semne de punctuație
int isspace(int c) – recunoaște spațiile albe
int isupper(int c) – recunoaște literele mari
int isxdigit(int c) – recunoaște cifre hexazecimale
int toupper(int c) – transformă literele în lit. mari
int tolower(int c) – transformă litere în lit. mici

Exemple

```
#include <stdio.h>
#include <ctype.h>
int main(void) {
  int c = getchar(); //citim caracterul urmator
  if(c != EOF){
    if(isalpha(c)){ //e litera?
       putchar(toupper(c)); //scrie ca majuscula

    }else if (isdigit(c)){ //e cifra?
       printf("%d\n", c);  //codul ASCII
       printf("%d\n", c-'0'); // valoare intreaga
      }
   }
  return 0;
 }
```

Recurența

Să ne amintim șiruri **recurente**:

progresie aritmetică: $x_0 = 1,\ x_n = x_{n-1} + 3$

progresie geometrică: $y_0 = 4,\ y_n = y_{n-1} * 5$

fibonacci: $t_0 = 0,\ t_1 = 1,\ t_n = t_{n-1} + t_{n-2}$

Termenul **curent** al șirului (x_n) este definit cu ajutorul unui termen **precedent** (de ex. x_{n-1}).

Există cel puțin **un caz de bază** (x_0).

Funcții recursive

O funcție recursivă este o funcție care **se regăsește în propria ei definiție** (f recursivă: cel puțin o valoare f(x) este definită în funcție de altă valoare f(y), și x $\neq$ y).

$f : \mathbb{N} \to \mathbb{N}$
$f(0) = 1$
$f(x) = 3 + f(x-1)$ pt. $x > 0$

$fib : \mathbb{N} \to \mathbb{N}$
$fib(0) = 0,\ fib(1) = 1$
$fib(x) = fib(x-1) + fib(x-2)$ pt. $x > 1$

$factorial : \mathbb{N} \to \mathbb{N}$
$factorial(0) = 1$
$factorial(x) = x * f(x-1)$ pt. $x > 0$

Funcții recursive în C

O funcție recursivă este o funcție al cărei corp conține cel puțin un **apel la ea însăși**.

$f : \mathbb{N} \to \mathbb{N}$
$f(0) = 1$
$f(x) = 3 + f(x-1)$ pt. $x > 0$

```
unsigned int f(unsigned int x){
  if (x==0){
        return 1; //caz de baza
    }else{
        return 3 + f(x-1); //caz general
    }
 }
```

Exemple

```
//fibonacci
unsigned int fib(unsigned int x){
  if (x==0 || x==1){
        return x;
    }else{
        return fib(x-1) + fib(x-2);
    }
 }

//factorial: folosim tipul mai mare, long
unsigned long fact(unsigned long x){
  if (x==0){
        return 1;
    }else{
        return x * fact(x-1);
    }
 }
```

Exemple: citire/scriere & recursivitate

```
//citeste toate caracterele pana la EOF
//void -- nu returneaza nimic

void citeste(int c){
  if (c != EOF){
        int nextc = getchar();
        citeste(nextc);
        putchar(c);
        //se vor tipari in ordine inversa
    }
 }

   ...
   citeste(getchar()); //apelul functiei
```

Exemple: citire/scriere & recursivitate

```
//citeste pana la EOF, numara doar cifrele
int cifre(int c){
  if (c == EOF){
        return 0;
  }else{
        int nextc = getchar();
        if(isdigit(c))
              return 1 + cifre(nextc);
          else
              return cifre(nextc);
    }
 }


   ...
   cifre(getchar()); //apelul functiei
```

Exemple: citire/scriere & recursivitate

```
//citirea unui numar natural cu rezultat partial

unsigned int numar(int c, int nr){
  if (!isdigit(c)){
        return nr;  //numarul calculat deja
    }else{
        int nextc = getchar();
        return numar(nextc, nr * 10 + c - '0');

        //la apelul urmator valoarea lui nr
        //va fi nr*10 + valoarea cifrei c
    }
 }


   ...
   numar(getchar(), 0); //apelul functiei
```