Introducere în Programarea calculatoarelor
Laborator 3

Operatori și instructiuni decizionale

Operatori aritmetici

Operatorii aritmetici folosiţi în limbajul C sunt:

- \+ adunarea a două numere;
- \- scăderea a două numere;
- * înmulţirea a două numere;
- / împărţirea a două numere (rezultatul împărţirii pentru numere reale, câtul împărţirii pentru numere întregi);
- % modulo (restul împărţirii a două numere întregi);
- ++ incrementarea (mărirea unei valori cu o unitate);
- -- decrementarea (micşorarea unei valori cu o unitate);

Exemplu pentru folosirea lui modulo**:**

```
#include <stdio.h>
 int main (void)
{
      int a = 54;
      int mod;
      mod = a % 10;
      printf ("Restul impartirii lui %d la 10 este %d\n",a, mod);
  return 0;
}
```

Operatorii de incrementare - decrementare sunt folosiţi pentru mărirea respectiv micşorarea unei valori cu o unitate.

Sunt de forma:

v++ - incrementare;
--v - decrementare;

Se pot folosi şi instrucţiuni de pre/post incrementare/decrementare:

- *post-incrementare*: x = a++; - este echivalentă cu x = a;
  a = a + 1;
- *pre-incremenatare*: x = ++a; - este echivalentă cu a = a + 1;
  x = a;

Pentru decrementare se procedează în mod analog.

Operatori relaționali și logici

Termenul „relaţional” se referă la relaţiile care se pot stabilii între diverse valori. Termenul logic se referă la felul în care se pot lega relaţiile existente. Întrucât operatorii relaţionali şi logici sunt deseori folosiţi împreună, îi vom trata împreună.

Introducere în Programarea calculatoarelor
Laborator 3

Ideea de adevărat sau fals stă la baza conceptelor de operatori logici şi relaţionali. În C adevărat înseamnă orice valoare deferită de 0, iar fals este 0. Expresiile care folosesc operatori relaţionali şi logici returnează 0 pentru fals şi 1 pentru adevărat.

Operator	Actiune
>	Mai mare decât
>=	Mai mare sau egal
<	Mai mic
<=	Mai mic sau Egal
= =	Egal
!=	Diferit

Operatori relaționali

Operator	Actiune
&&	SI
||	SAU
!	NU

Operatori logici

Tabelul cu valorile de adevăr pentru operatorii logici este prezentat folosind cifrele 1 şi 0.

a	b	a && b	a || b	!a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Valorile de adevăr a operatorilor logici

Exemplu:

```
int a=3,b=4,c=0,rez;
 rez=!  (a<c)  &&  (a<b)  ||  (b<=c);
printf(”%d\n”, rez);
```

Operatori la nivel de bit

Limbajul C pune la dispoziţie şase operatori la nivel de bit, furnizând astfel facilităţi specifice de obicei limbajelor de asamblare. Aceşti operatori permit scrierea unor programe care lucrează îndeaproape cu sistemul hardware al calculatorului.

Operatorii la nivel de bit se aplică unor valori întregi asociate cu tipurile *char, int, long, cu / fără semn* - iar rezultatele obţinute sunt, de asemenea, întregi. Faţă de ceilalţi operatori ai limbajului, aceştia operează asupra fiecărui bit din reprezentarea internă a operanzilor, tratându-l independent de valorile celorlalţi biţi.

Operatorii la nivel de bit sunt următorii.

- & - *AND* (şi logic);

- | - *OR* (sau logic);
- ^ - *XOR* (sau exclusiv);
- << - deplasare la stânga;
- >> - deplasare la dreapta;
- ~ - negare la nivel de bit (operator unar);

Întreg A	**Întreg B**	**Rezultatul**		
		&	^	|
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	1

Operatori pe biți

Indiferent de scopul în care sunt utilizaţi operatorii, maniera în care sunt folosiţi este în general aceeaşi. În cele ce urmează, se vor descrie pe scurt principiile de bază ale utilizării lor.
Observaţie: Operatorii la nivel de bit nu modifică valoarea operanzilor (se comportă ca şi operatorii aritmetici obişnuiţi), astfel o operaţie n << 2 va rezulta într-un nou număr, fără a modifica valoarea variabilei n. Pentru a modifica valoarea variabilei n trebuie să avem o atribuire:

```
n = n << 2;
```

Operatorii de deplasare (<< şi >>) realizează deplasarea biţilor din operandul stâng cu atâtea poziţii câte sunt indicate de operandul drept.

Deplasarea la stânga (<<) completează la dreapta cu 0 (biţii din stânga se vor pierde), iar la dreapta numărului se completează cu 0.
Exemplu:

```
int a=255;          /* 0000 0000 1111 1111*/
a=a<<3;             /* 0000 0111 1111 1000*/
```

În cazul *deplasării la dreapta (>>)*, biţii din dreapta se pierd, iar în stânga se introduc biţi de 0, în caz că numărul era de tip fără semn (unsigned) sau biţi egali cu bitul de semn (cel mai din stânga bit) în cazul numerelor de tip cu semn.
Operatorul de negare ~ realizează complementarea tuturor biţilor unui număr (bitii de 0 se pun pe 1, iar cei de 1 se pun pe 0)

Exemplu:

```
int a = 5171;       /* 0001 0100 0011 0011*/
a=~a;              /*1110 1011 1100 1100*/
```

Întrucât întregul a este cu semn, prin complementare bitul de semn a luat valoarea 1 şi s-a obţinut un număr negativ.

Introducere în Programarea calculatoarelor
Laborator 3

Declararea constantelor

În limbajul C, constantele se pot declara în două moduri:

- folosind cuvântul cheie **const**;
- folosind directiva **#define**;

Cuvântul cheie const

Declararea unei valori constante se poate face folosind cuvântul cheie **const**, astfel:

```
const int c = 14;
```

Prin această declarare, variabila ’c’ ia valoarea 14 şi NU i se mai poate modifica această valoare. O variabilă declarată folosind *const* nu se poate folosi pentru precizarea lungimii unui şir.

Directiva #define

Directiva **define** este o directivă preprocesor, utilizată pentru a face un program mai uşor de realizat/parcurs.

Declararea unei directive preprocesor începe cu simbolul # şi nu se terminată cu ';'. Ea poate apărea oriunde într-un program, dar afectează numai liniile care urmează declarării ei. În mod uzual, directivele preprocesor sunt scrise la începutul programului.
În general, directivele preprocesor sunt scrise cu majuscule.

Considerăm următoarele exemple:

```
#define TRUE    1
#define FALSE   0
#define NULL    0
#define AND     &
#define OR      |
#define EQUALS  ==
gata = TRUE;
```

Directivele preprocesor sunt prelucrate de către compilator înainte de compilarea programului. Secvenţa este următoarea: se procesează directivele, apoi simbolurile ce apar în cadrul programului sunt înlocuite cu valoarea lor, iar ulterior programul este compilat.

Avantajele folosirii directivei #define

- lizibilitate mărită a programului - citirea si înţelegerea rapidă a fişierului sursă;
- schimbările ulterioare ale unor valori constante se pot face foarte uşor;

Exemplu: modificarea valorii lui LIMIT de la 100 la 10.000 – se face prin înlocuirea liniei:

```
#define LIMIT  100
```

cu
linia:

```
#define LIMIT  10000
```

Daca nu s-ar fi folosit acest mod de definire a constantei LIMIT, atunci ar fi trebuit sa se modifice peste tot in program 100 cu 10000.

Macroinstrucțiuni

Identificatorul definit prin #define poate fi substituit cu o instrucțiune.

Exemplu:

```
# define afiseaza(a) printf ("%.2f \n",a)
int main (void)
{
   float b = 3.12;
   afiseaza (b);
   return 0;
}
```

În acest program preprocesorul va înlocui expresia **afiseaza (b)** cu **printf ("%.2f \n",n1)**;

De asemenea, **#define** poate fi folosita pentru a declara constante simbolice.

```
# define RAD2 1.41
float x;
...
x = RAD2 * 15;
...
```

Ulterior, în program, nu se pot atribui valori constantelor simbolice.

Introducere în Programarea calculatoarelor
Laborator 3

Instrucțiunile decizionale ale limbajului C

Instrucțiunea *if*

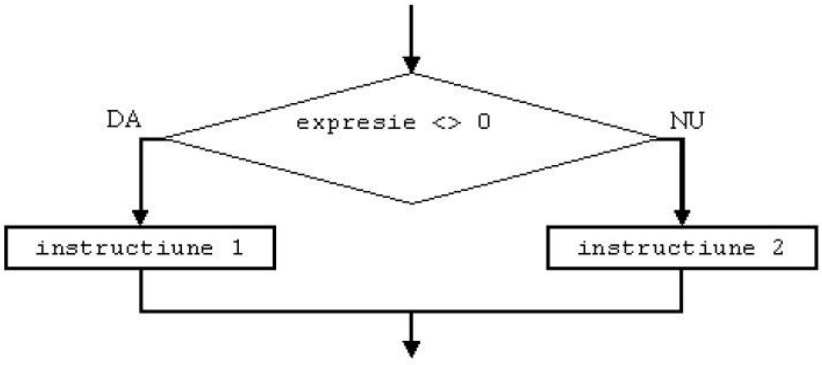
Instrucțiunea decizionala *if* permite executarea condiționată a unei secvențe de program în funcție de valoarea unei expresii. Sintaxa acestei instrucțiuni este următoarea:

```
if (expresie)
instructiune_1;
else
instructiune_2;
```

Dacă expresia are valoare adevărata, se executa *instructiune_1*. Altfel, se executa *instructiune_2*. Ramura *else* este opționala si poate sa lipsească.

Limbajul C nu dispune de un tip de data specializat pentru valori logice (*adevărat* sau *fals*). Orice expresie numerica *diferita de zero* se considera o valoare *adevărata*. Orice expresie numerica *egala cu zero* se considera *fals*a.

Schema logica aferenta secvenței de mai sus arata în felul următor:



Schemă logică - instrucțiunea if

Daca se dorește introducerea mai multor instrucțiuni pe o ramura, acestea trebuie încadrate între acolade:

```
if (expresie)
{
instructiune_1_1; instructiune_1_2;
}
else
{
instructiune_2_1; instructiune_2_2;
}
```

În construcția expresiilor logice, se pot folosi următorii operatori:

Operator	Descriere	Precedenta
<	mai mic	7
<=	mai mic sau egal	7
>	mai mare	7
>=	mai mare sau egal	7
!=	diferit	8
==	egal	8
&&	SI logic	12
||	SAU logic	13
!	NEGARE logica	2

Când se evaluează o expresie, operatorii se evaluează în ordinea crescătoare a precedentei (primul operator evaluat este cel cu cea mai mica valoare pe coloana precedenta). Trebuie făcuta distincție între operatorii logici && (SI), || (SAU) si operatorii pe biți & (SI la nivel de biți), | (SAU la nivel de biți).

Secvența următoare de cod ilustrează acest lucru:

```
int a = 1; /* 01 in binar */
int b = 2; /* 10 in binar */
int x = a & b; /* SI la nivel de bit, x ia valoarea 0 */
int y = a && b; /* SI logic, y ia o valoare diferita de 0 */
/* expresia se interpretează in felul următor: y ia o valoare diferita de zero daca a este diferit de
zero SI b este diferit de zero. */
```

Exemplu: sa se scrie un program în C care sa rezolve o ecuație de gradul I, cu coeficienții citiți de la tastatura:

```
#include <stdio.h>
int main (void)
{ float a, b, x;
        printf ("Coeficientul lui x=");
        scanf ("%f", &a);
        printf ("Termenul liber=");
        scanf ("%f", &b);
        if (a == 0)
        {
          if (b != 0)
                printf ("Nu are solutii \n");
          else
                printf ("X este orice numar real \n");
         }
        else
        {
          x = -b / a;
         printf ("X = %f", x);
        }
return 0;
}
```

Exemplu: Sa se scrie un program C care sa verifice daca trei numere reale citite de la tastatura pot reprezenta lungimile laturilor unui triunghi:

```
#include <stdio.h> int
main (void)
{ float a, b, c;
 printf ("Introduceti cele 3 lungimi ale laturilor: \n");
 scanf ("%f %f %f", &a, &b, &c);
 if ((a < (b + c))&&(b < (a + c))&&(c < (a + b)) )
          printf ("Cele trei segmente pot forma un triunghi ! \n");
 else
          printf ("Cele trei segmente nu pot forma triunghi ! \n");
 return 0;
}
```

Introducere în Programarea calculatoarelor
Laborator 3

Operatorul conditional

Operatorul condițional ?: permite simplificarea scrierii unor expresii condiționale în care se selectează una din doua valori, în funcție de o condiție logica. Sintaxa acestuia este următoarea:

(conditie) **?** val_1 **:** val_2

Aceasta expresie va returna valoarea val_1 în cazul în care *condiție* este o expresie adevărata (diferita de zero) si val_2 în cazul în care *condiție* este o expresie falsa (egala cu zero).

Exemplu: Sa se determine maximul dintre 3 variabile:

```
int a = 2;
int b = 5;
int c = 3; int
m;
m = (a > b) ? a : b;
m = (m > c) ? m : c;
printf ("maximul este: %d", m);
```

Utilizând macrodefiniţiile, același exemplu se poate scrie într-o forma mai eleganta:

```
#define max(a, b) ((a > b) ? a : b) int a = 2;
int b = 5;
int c = 3;
int m;
m = max (a, b);
m = max (m, c);
printf ("maximul este: %d", m);
```

Instrucțiunea *switch*

Instrucțiunea *switch* permite selectarea unei alternative din mai multe posibilităţi în funcție de valoarea unei variabile sau expresii de tip întreg. Sintaxa este următoarea:

```
switch (expresie)
{
case valoare_1: instructiune_11;
                instructiune_12;
                 instructiune_13;
break;
case valoare_2: instructiune_21;
```

```
            instructiune_22;
break;
case valoare_N: instructiune_N;
break;
default: instructiune_M;
break;
}
```

Selectorul *expresie* trebuie sa fie de tip întreg. Toate valorile din care se face selecția, *valoare_1* până la *valoare_N* trebuie sa fie întregi. Ele trebuie sa fie si distincte.

Daca *expresie* este egala cu *valoare_1*, execuția continua cu instructiunile care apar pe aceasta ramura (*instructiune_11, instructiune_12, instructiune_13*) si continua pâna la întâlnirea unei instrucțiuni *break* sau până când se încheie blocul *switch*.

Daca *expresie* este egala cu *valoare_2*, se executa *instructiune_21, instructiune_22* etc.
Daca expresie nu este egala cu nici una dintre valorile *valoare_1* până la *valoare_N*, se executa instrucțiunile de sub eticheta *default*. Aceasta eticheta este opțională.

Exemplu: Programul următor afișează un calificativ în funcție de nota unui student citita de la tastatura.

```
#include <stdio.h>
int main (void)
{
  int nota;
 printf ("Nota= "); scanf ("%d", &nota);
switch (nota)
{
```

```
case 1:
case 2:
case 3:
case 4: printf ("Repetent
\n"); break;

case 5:
case 6:
case 7:
        printf ("Slab \n");
break; case 8:
case 9: printf ("Bine \n"); break;
case 10: printf ("Foarte bine \n"); break;
default: printf ("Valoarea introdusa nu este o nota corecta!");
break;
} retun 0;
}
```

Probleme propuse

1. Să se scrie un program în C care să rezolve o ecuație de gradul 2, cu coeficienții a, b, c reali introduși de la tastatura.

2. Să se scrie un program în C care citește 2 numere de la tastatură și verfică dacă sunt divizile cu 3. Dacă sunt divizibile se va face suma dintre ele, dacă nu produsul.

3. Să se realizeze un program care citește două numere reale de la tastaură și apoi afișează un meniu cu următoarele opțiuni:
   a. Suma numerelor introduse
   b. Diferenta numerelor introduse
   c. Produsul numerelor introdue
   d. Raportul numerelor introduse
   e. Iesire

   Opțiunea dvs:

4. Să se scrie un program în C care să rezolve ecuația de gradul II, ax2+bx+c=0, pentru a,b,c numere întregi citite de la tastatură.

Introducere în Programarea calculatoarelor
Laborator 3

5. Se citesc 3 variabile reale a, b si c. Sa se calculeze valoare expresiei:

$$f = \begin{cases} \sqrt{a+b}, \text{dacă } c > 0 \\ \frac{a}{b}, \text{dacă } c = 0 \\ a * b, \quad \text{dacă } c < 0 \end{cases}$$

5. Se citesc de la tastatura 3 numere reale pozitive, a b si c. Scrieti un program în C care să verifice dacă numerele citite pot constitui laturile unui triunghi. În caz afirmativ, determinați tipul triunghiului (echilateral, isoscel sau dreptunghic) și afișați aria sa.

6. Fie a și b două numere întregi citite de la tastatură. Scrieți un algoritm care să verifice dacă a și b sunt numere consecutive.