|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | Cristina  Mihailescu |  |  |

Programare Procedurala

Cuprins

1. [Introducere 2](#_Toc438569188)
2. [Operatori: 2](#_Toc438569189)
3. [Ordinea operatorilor: 2](#_Toc438569190)
4. [Functii de citire/scriere 3](#_Toc438569191)
5. [Specificatori de format 3](#_Toc438569192)
6. [Structura de decizie multipla 4](#_Toc438569193)
7. [Pointeri 5](#_Toc438569194)
8. [Functii pentru alocarea dinamica a memoriei 5](#_Toc438569195)
9. [Alocarea dinamica a tablourilor bidimensionale 6](#_Toc438569196)
10. [Functii pentru manipularea memoriei (<string.h>) 7](#_Toc438569197)

# Introducere

Note: Daca cel putin unul este real => impartirea este reala.

1. 7/2 = 3
2. 7/2.0 = 3.5

Corect:

1. **int** a,b;
2. **double** c;
3. c=(**double**)a/b // c=3.5, a -> double
4. c=a/**double**(b) // c=3.5

Gresit: c=(double)(a/b) // c=3.0

##### **Tip de date boolean:**

1. **int** a=1, b=7;
2. **int** r=(a<b); // r=1

|  |  |
| --- | --- |
|  | if(x) ⬄ if(x!=0)if(!=x) ⬄ if x<<n ⬄ x=x\*2^n  x>>n ⬄ x=x/2^n |

**x^y =** (x XOR y)

# Operatori:

* +, -, \*, /, % (aritmetici)
* <, >, <=, >=, ==, != (relationali)
* !, ||, && (logici)
* <<, >>, &, |, ^, ~ (de biti)
* (tip de date) expresie (cast; schimba temporar modul in care este interpretata expresia)
* sizeof (de dimensiune; ex: sizeof(int))
* = (atribuire)
* ++, -- (incrementare/decrementare)
* ?: (conditional; expresie de tip logic ? expresie\_caz\_true : expresie\_caz\_false)
* , (virgula; z=(x=1, y=2) => z=2 – z la valoarea ultimei expresii)

# Ordinea operatorilor:

1. **( ), [ ], ++, --** (postfixate), structuri (-> asociativitate)
2. **+, -** (operatori unari), **!, ~, ++, --** (prefixate), **sizeof, \*, &** (pointeri) (<- asociativitate)
3. **\*, /, %** (-> asociativitate)
4. **+, -** (binari; -> asociativitate)
5. **>>, <<** (-> asociativitate)
6. **<, <=, >, >=** (-> asociativitate)
7. **==, !=** (-> asociativitate)
8. **&** (biti; -> asociativitate)
9. **^** (biti; -> asociativitate)
10. **|** (biti; -> asociativitate)
11. **&&** (-> asociativitate)
12. **||** (-> asociativitate)
13. **?:** (<- asociativitate)
14. **=** (<- asociativitate)
15. **,** (-> asociativitate)

# Functii de citire/scriere

1. Citire: scanf (scanare cu format)

FG: scanf("%specificatori de format", &variabila1, &variabila 2, ...)

1. Afisare: printf

FG: printf("%specificatori de format", expresie1, expresie2, ...)

Secvente escape:

* \n linie noua
* \t tab
* \a bell
* %% pentru %

# Specificatori de format

Note: h=short, l=long, u=unsigned

* %c (char, caractere), %hhd (char, numere), %hhu(char, numere unsigned)
* %d / %i (int), %u (unsigned int)
* %ld / %li (long int), %lu (unsigned long int)
* %hd / %hi (short int), %hu (unsigned short int)
* %lld / %lli (long long int), %I64 (pe Windows), %llu (unsigned long long int, pe Linux), %I64 (unsigned long long int, pe Windows)
* %f (float)
* %lf (duble, scanf), %f (double, printf), %Lf (long double)
* %s (sir de caractere)
* %p (pointeri, printf)

1. scanf("%1d", &x); //se va citi o cifra
2. scanf("%d%\*c%d, &h, &m) // %\*c citeste un caracter, dar nu il pastreaza intr-o variabila
3. scanf("%s", sir) //citeste pana cand intalneste un spatiu
4. scanf("%[^\n]s", sir) //citire sir cu spatii

## Metode pentru citirea sirurilor

1. scanf – nu contine capatul sirului
2. **char** sir[100];
3. scanf("%99s, sir);
4. fgets (sir, nr\_maxim\_carac, stdin) – contine capatul sirului
5. **char** sir[100];
6. fgets(sir, 100, stdin);
7. sir[strlen(n)-1] = '\0'; // scapam de \n

# Structura de decizie multipla

1. **switch** (expresie)
2. {
3. **case** valoare1:
4. {instructiuni\_1; **break**;}
5. **case** valoare2:
6. {instructiuni\_2; **break**;}
7. ...
8. **default**: //optional
9. instructiuni\_0;
10. }

Nota: NAN (Not-A-Numer; constanta) – folosita pentru expresii care nu sunt definite (ex: x/0).

**isnan(n) =**

**isfinite(n) =**

# **Pointeri**

|  |  |
| --- | --- |
|  | &variabila = adresa de memorie la care este stocata variabila (operator de referentiere)  \*pointer = valoarea aflata la aderesa memorata in pointer (operator de dereferentiere) |

1. **int** a, \*p;
2. p=&a;
3. a=10; //accesare directa
4. \*p=10; //acesare indirecta

**a=\*(&a)**

**if**(p!=NULL) // verificare daca p este alocat sau nu

Note: sizeof(pointer) = const = 4 octeti, indifferent de tipul de baza al pointerului

Aritmetica pointerilor are sens daca adresele implicate sunt adresele unor elemente dintr-un tablou.

|  |  |
| --- | --- |
|  | v[i] = \*(v+i) = \*(i+v) = i[v] |

# Functii pentru alocarea dinamica a memoriei

1. **int** \*v = (**int** \*) malloc (n\***sizeof**(**int**));
2. **if**(v!=NULL)
3. { ...instructiuni... }
5. **int** \*v = (**int** \*) calloc (n\***sizeof**(**int**)); //initializeaza intreaga zona de memorie cu 0
6. **if**(v!=NULL)
7. { ...instructiuni... }
9. aux = (**int** \*) realloc (v, n\***sizeof**(**int**)); //dimensiune noua;
10. **if**(aux!=NULL) /\*v = int realloc (v, n\*sizeof(int)); poate distruge vectorul daca nu exista spatiu pentru realocare\*/
11. {
12. v=aux;
13. ...instructiuni...
14. }

Exemplu:

1. #include <stdio.h>
2. #include <stdlib.h>
4. **int** main()
5. {
6. **int** \*v=NULL, \*aux=NULL, x, n=0, i;
7. **do**
8. {
9. scanf("%d", &x);
10. n++;
11. **if**(x!=0)
12. {
13. aux = (**int** \*) realloc(v,n\***sizeof**(**int**));
14. **if**(aux!=NULL)
15. {
16. v=aux;
17. v[n-1]=x;
18. }
19. **else**
20. {
21. printf("Eroare de compilare");
22. free(v);
24. }
25. }
26. }
27. **while**(x!=0);
28. **for**(i=0; i<n-1; i++)
29. printf("%d ", v[i]);
30. **return** 0;
31. }

# Alocarea dinamica a tablourilor bidimensionale

## Alocare necompacta

1. **int** \*\*a;
2. a = (**int** \*\*) malloc (n\***sizeof**(**int** \*));
3. //alocare
4. **for**(i=0; i<m; i++)  //n coloane
5. a[i] = (**int** \*) malloc (m\***sizeof**(**int**));
6. //dezalocare
7. **for**(i=0; i<n; i++)  //m linii
8. free(a[i]);
9. free(a);

## Alocare compacta

* **int** \*\*a;
* a = (**int** \*\*) malloc (n\***sizeof**(**int** \*));
* a[0] = (**int** \*) malloc (n\*m\***sizeof**(**int** \*));
* //alocare
* **for**(i=1; i<n; i++)
* a[i] = a[0] + i\*m;
* //dezalocare
* free(a[0]);
* free(a);

Pointer generic:

* poate memora adrese de orice tip
* nu au aritmetica (necesita conversie explicita)

Exemplu:

1. **int** a = 10;
2. **double** t = 3.14;
3. **void** \*p;
4. p = &a;
5. printf("%d", \*(**int** \*)p);
7. \*(**int** \*)p = 7;
8. p = &t;
9. printf("%lf", \*(**double** \*)p);
11. \*(**double** \*)p = 100.5;

# Functii pentru manipularea memoriei (<string.h>)

* memcpy (a nu se folosi atunci cand destinatia si sursa se suprapun)

1. **int** dest[100], surs[100];
2. memcpy (dest, surs, n\***sizeof**(**int**));

* memmove (se foloseste atunci cand destinatia si sursa se suprapun)

1. **int** v[100], v[100];
2. memcpy (v+i, v+j, (n-j)\***sizeof**(**int**));

* memset (seteaza nr\_octeti incepand de la adresa indicata la valoarea indicate ca parametru)

memset (\*adresa, val, nr\_octeti);

Exemplu: memset (v, 0, 100\***sizeof**(**int**)); //primele 100 de elemente ale lui  devin 0

* memchr (returneaza un pointer la prima aparitie a valorii indicate in block)

1. **char** \* pch;
2. **char** str[] = "Example string";
3. pch = (**char**\*) memchr (str, 'p', strlen(str));
4. **if** (pch!=NULL)
5. printf ("'p' found at position %d.\n", pch-str+1);
6. **else**
7. printf ("'p' not found.\n");

* memcmp

int r = memcmp (v, w, n);

# Fisiere. Fisiere text

La citire ‘\r’ dispare.

Ana are\r\n (9 octeti, 8 ch)

\r\n (2 octeti, 1 ch)

Mere si \r\n (10 octeti, 9 ch)

Pere. (5 octeti, 5 ch)

Deschiderea unui fisier: **FILE** \*fopen (**char** \*nume\_fisier, **char** \*mod)

Mod (fisier text) =

Mod (fisier binar) – rb, wb, ab.

1. **while**(1)
2. {
3. citeste x;
4. **if**(feof(x)) **break**;
5. prelucreaza x;
6. }

# Functii de citire/scriere pentru fluxuri de tip text

1. La nivel de caracter

CITIRE:

1. **int** fgetc(**FILE** \*f)
2. //returneaza codul ASCII al caracterului citit
3. //eroare => EOF

SCRIERE:

1. **int** fputc(**int** c, **FILE** \*f)
2. //returneaza codul ASCII al caracterului citit
3. //eroare => EOF

Exemplu:

1. **int** main()
2. {
3. **FILE** \* in, \* out;
4. **int** c;
5. in = fopen("sursa.txt", "r");
6. out = fopen("destinatie.txt", "w");
7. **if**(!in || !out)
8. {
9. printf("Eroare la deschiderea fisierelor!");
10. exit(0);
11. }
12. **while**((c = fgetc(in))!=EOF)
13. fputc(c, out);
14. **if**((fclose(in)!=0) || (fclose(out)!=0))
15. {
16. printf("Eroare la inchiderea fisierelor!");
17. exit(0);
18. }
19. **return** 0;
20. }
21. La nivel de linie

CITIRE:

1. **char** \*fgets(**char** \*linie, **int** nr\_max\_char, **FILE** \*f)
2. //returneaza sirul linie
3. //eroare => NULL
4. **char** linie[100];
5. fgets (linie, 100, f) // linie = "Ana are\n\0"
6. linie[strlen(linie)-1) = '\0'; // pentru ca citeste si caracterul \n !!

SCRIERE:

1. **char** \*fputs(**char** \*sir, **FILE** \*f)
2. //returneaza > 0
3. //eroare => EOF

|  |  |
| --- | --- |
|  | rewind(in) //adduce pointerul la inceputul fisierului |

Exemplu: Vezi cursul de pe data de 18.11.2015, pagina 6.

1. Cu format

CITIRE: **int** fscanf(**FILE** \*f, "%..", &variabile)

SCRIERE: **int** fprintf(**FILE** \*f, "%..", variabile)

27\n

1. **int** v;
2. **char** s[100];
3. **FILE** \*f = fopen("sursa.txt", "r");
4. fscanf(f, "%d", &v);
5. fgetc(f); // citeste in gol '/n'
6. printf("%d", v);

|  |  |
| --- | --- |
|  | **while** (fscanf(f, "%d %d", &x, &y) == 2) //cat timp se citesc doua numere pe o linie  **while** (fscanf(f, "%d %d", &x, &y) != EOF)  //sunt echivalente |

1. freopen ("exemplu.txt", "w", stdout); //stdin pentru citire
2. printf("Text"); // afiseaza "Text" in fisierul exemplu.txt

# Fisiere binare

1. **int** fwrite (**void** \*adresa, **int** dim\_elem, **int** nr\_elem, **FILE** \*f);
2. **int** fread (..);
3. //urmatoarele sunt echivalente
4. **for**(i=0; i<n; i++)
5. fwrite (v+i, **sizeof**(**int**), 1, f);
6. fwrite (v, **sizeof**(**int**), n, f);
7. fwrite (v, n\***sizeof**(**int**), 1, f);
8. fwrite (v, 100\***sizeof**(**int**), 1, f);

# Functii pentru fisiere binare

1. fseek
2. **int** fseek (**FILE** \*f, **int** nr\_octeti, **int** origine) // 0 = succes, !=0 = eroare
3. /\*
4. origine:
5. SEEK\_SET = inceputul fisierului
6. SEEK\_END = sfarsitul fisierului
7. SEEK\_CUR = octetul curent
8. \*/
9. ftell
10. **int** ftell (**FILE** \*f)
11. //returneaza numarul de ordine al octetului curent
12. //altfel -1 (EOF)
13. remove
14. **int** remove (**char** \*cale\_fisier) //sterge fisierul de pe hard disk
15. //0 = succes, !=0 = eroare
16. rename
17. **int** rename (**char** \*nume\_vechi, **char** \*nume\_nou) //redenumeste fisierul
18. //0 = succes, !=0 = eroare
19. tmpfile

**FILE** \*tmpfile ()//creeaza un fisier temporar cu nume unic si il deschide cu wb+

1. tmpnam
2. **char** \*tmpnam(**char** \*nf) //creeaza un nume de fisier temporar unic
3. //Exemplu:
4. **char** \*nf = tmpnam(NULL);

# Functii

1. **int** suma(**int**, **int**)
3. **int** main()
4. {
5. ...
6. suma(a, b);
7. ...
8. }
10. **int** suma(**int** x, **int** y)
11. {
12. **return** x+y;
13. }

Nota: Vezi fisiere binare.

|  |  |
| --- | --- |
| Data | Stare |
| 7 | Done |
| 14 | Done |
| 21 | Done |
| 28 | Done |
| 4.11 | Done |
| 11.11 | Done |
| 18.11 | Done |
| 25.11 | Done |
| 2.12 | Processing |
| 9.12 | In queue |
| 16.12 | In queue |