

Cuprins

1. Elemente de bază ale limbajului C/C++	11
1.1. Noțiuni introductive	11
<i>Evoluția limbajelor de programare</i>	11
1.2. Setul de caractere	12
1.3. Identificatori	12
1.4. Cuvinte rezervate	13
1.5. Comentarii	13
1.6. Separatori	14
1.7. Structura generală a unui program C/C++	15
1.8. Tipuri de date standard	16
<i>Tipul de date int</i>	16
<i>Tipul char</i>	17
<i>Tipuri reale</i>	19
<i>Tipul void</i>	19
1.9. Variabile	20
1.10. Preprocesare	21
1.11. Utilizarea funcțiilor din bibliotecile standard	22
1.12. Citirea/scrierea datelor	24
1.13. Citiri și scrieri în limbajul C++	24
1.14. Citiri și scrieri în limbajul C	26
<i>Citirea datelor cu format specificat</i>	26
<i>Citirea caracterelor</i>	29
<i>Afișarea datelor cu format</i>	29
1.15. Expresii	31
<i>Operatori aritmetici</i>	32
<i>Operatori de incrementare/decrementare</i>	32
<i>Operatori relaționali</i>	33
<i>Operatori de egalitate</i>	33
<i>Operatori logici globali</i>	34
<i>Operatori logici pe biți</i>	34
<i>Operatori de atribuire</i>	36
<i>Operatori condiționali</i>	37

Operatorul de referențiere (adresă)	37
Operatorul de conversie explicită	37
Operatorul de determinare a dimensiunii	38
Operatorul virgulă	38
Evaluarea expresiilor	39
Tabелul priorității operatorilor	40
1.16. Aplicații	40
Disc	40
Compus chimic	41
Triunghi	42
Schimb	43
Leul și iepurașul	44
Bazin	45
1.17. Probleme propuse	45
2. Instrucțiunile limbajului C/C++	50
2.1. Instrucțiunea expresie	50
2.2. Instrucțiunea compusă	50
2.3. Instrucțiunea if	51
2.4. Instrucțiunea switch	51
2.5. Instrucțiunea break	53
2.6. Instrucțiunea while	53
2.7. Instrucțiunea do-while	54
2.8. Instrucțiunea for	55
2.9. Aplicații	56
Modul	56
Paritate	57
Ecuația de gradul al II-lea	57
Sumă	58
Aria triunghiului	58
Secvență	59
Calcul	62
Numărare	64
Putere	66
Numere	67
Coduri și caractere	68
Secvență simetrică	69
Maxim	70
Media aritmetică	72
Suma cifrelor	73
Perechi	73

<i>Divizori</i>	74
<i>Număr prim</i>	75
<i>Descompunere în factori primi</i>	76
<i>Cel mai mare divizor comun</i>	77
<i>Gard</i>	78
<i>Exponent</i>	79
<i>Termen Fibonacci</i>	80
<i>Existență</i>	81
<i>Număr divizori primi</i>	82
<i>Vocale</i>	82
<i>Expresie</i>	83
<i>Radical</i>	84
<i>Relație</i>	84
<i>Sumă de două numere prime</i>	85
<i>Progres</i>	86
<i>Tabla adunării</i>	87
<i>Semne</i>	88
<i>Excursie</i>	88
2.10. Probleme propuse	90
3. Fișiere	102
3.1. Noțiuni introductive	102
3.2. Fișiere text în limbajul C++	103
<i>Declararea fișierelor</i>	103
<i>Deschiderea fișierelor</i>	103
<i>Citirea datelor dintr-un fișier</i>	105
<i>Scrierea datelor într-un fișier</i>	105
<i>Operații de test</i>	106
<i>Închiderea unui fișier</i>	107
3.3. Fișiere text în limbajul C	107
<i>Declararea fișierelor</i>	107
<i>Deschiderea fișierelor</i>	108
<i>Citirea datelor dintr-un fișier</i>	109
<i>Scrierea datelor într-un fișier</i>	109
<i>Închiderea fișierelor</i>	109
3.4. Probleme propuse	111
4. Tipuri structurate de date	112
4.1. Tablouri	112
4.2. Prelucrări elementare pe vectori	114
<i>Citirea unui vector</i>	114
<i>Afișarea unui vector</i>	114
<i>Copierea unui vector</i>	114

<i>Determinarea elementului maxim/minim dintr-un vector</i>	114
<i>Media aritmetică a elementelor strict pozitive</i>	115
<i>Inversarea ordinii elementelor din vector</i>	115
<i>Verificarea unei proprietăți</i>	116
<i>Căutarea unui element într-un vector</i>	117
<i>Sortare</i>	118
<i>Interclasare</i>	120
4.3. Prelucrări elementare cu matrice	121
<i>Citirea unei matrice</i>	121
<i>Afișarea unei matrice</i>	121
<i>Parcurgerea unei matrice pe linii</i>	122
<i>Parcurgerea unei matrice pe coloane</i>	122
4.4. Prelucrări elementare specifice matricelor pătrate	122
<i>Parcurgerea diagonalelor</i>	122
<i>Parcurgerea elementelor de sub diagonala principală</i>	123
4.5. Șiruri de caractere	123
4.6. Citirea și afișarea șirurilor de caractere	124
<i>Citirea unui șir de caractere în limbajul C++</i>	124
<i>Citirea unui șir de caractere în limbajul C</i>	125
<i>Afișarea unui șir de caractere în limbajul C++</i>	126
<i>Afișarea unui șir de caractere în limbajul C</i>	126
4.7. Tipul pointer	127
<i>Declararea unui pointer de date</i>	127
<i>Operații cu pointeri</i>	128
4.8. Legătura dintre pointeri și tablouri	130
<i>Utilizarea funcțiilor standard de lucru cu șiruri de caractere</i>	131
4.9. Tipul de date struct	137
<i>Declararea unui tip struct</i>	137
<i>Referirea la un câmp al unei structuri</i>	138
<i>Inițializarea unei structuri</i>	139
<i>Citirea și afișarea structurilor</i>	139
<i>Operații cu structuri</i>	139
4.10. Asocierea unui nume pentru un tip de date	139
4.11. Aplicații	140
<i>Temperaturi</i>	140
<i>Inserare medii</i>	141
<i>Combinare vectori</i>	141
<i>Ciurul lui Eratostene</i>	143
<i>Copii</i>	144
<i>Perechi</i>	145
<i>Repetiție</i>	146

<i>Eliminare</i>	148
<i>Permutare ciclică</i>	149
<i>Generarea mulțimii</i>	150
<i>Schema lui Horner</i>	152
<i>Subsecvență de sumă maximă</i>	153
<i>Cărți</i>	154
<i>Marcare</i>	155
<i>Depozit</i>	156
<i>Problema celebrității</i>	157
<i>Cadran</i>	158
<i>Matrice</i>	159
<i>Situație școlară</i>	161
<i>Pseudodiagonale</i>	164
<i>Submulțimi cu sume egale</i>	167
<i>Dreptunghi de sumă maximă</i>	168
<i>Secvențe</i>	170
<i>Problema spectacolelor</i>	172
<i>Problema rucsacului</i>	173
<i>Generare de submulțimi</i>	175
<i>Generare elemente produs cartezian</i>	176
<i>Expresie</i>	177
<i>Reactivi</i>	178
<i>Furnici</i>	181
<i>Apariții cifră</i>	183
<i>Examen de capacitate</i>	184
<i>Aparițiile unei cuvânt</i>	185
<i>Propoziție</i>	186
4.12. Probleme propuse	187
5. Stiva și coada	201
5.1. Stiva	201
<i>Care este utilitatea stivelor?</i>	202
<i>Cum implementăm o stivă?</i>	202
5.2. Coada	204
<i>Care este utilitatea unei cozi?</i>	205
<i>Cum implementăm o coadă?</i>	205
5.3. Aplicații	207
<i>Depou</i>	207
<i>Paranteze</i>	208
<i>Manna-Pmelî</i>	210
<i>Caroiaj</i>	211
5.4. Probleme propuse	213

6. Funcții	218
6.1. Subprograme în limbajul C/C++	219
6.2. Definiția unei funcții	219
6.3. Declararea funcțiilor	221
<i>Utilitatea declarațiilor</i>	222
<i>Biblioteci de funcții și fișiere antet</i>	222
6.4. Apelul funcțiilor	223
6.5. Transferul parametrilor prin referință	226
<i>Utilizarea pointerilor ca parametri</i>	226
<i>Tipul referință</i>	227
6.6. Variabile globale și variabile locale	231
<i>Variabilele globale</i>	232
<i>Variabilele locale</i>	232
<i>Regula de omonimie</i>	232
<i>Operatorul de rezoluție</i>	233
6.7. Specificatori de clasă de memorare	233
6.8. Parametrii funcției main ()	234
6.9. Caracteristici specifice funcțiilor C++	236
<i>Funcții inline</i>	236
<i>Funcții cu parametri implicați</i>	236
<i>Supraincărcarea funcțiilor</i>	237
<i>Funcții șablon</i>	238
6.10. Proiecte	240
6.11. Aplicații	242
<i>Expresie</i>	242
<i>Find & Replace</i>	243
<i>Cel mai mare divizor comun cu descompunere în factori primi</i>	245
<i>Incluziune</i>	247
<i>Operații cu numere naturale mari</i>	248
<i>Secvență regulată</i>	252
6.12. Probleme propuse	255
Anexe	264
1. Tabela codurilor ASCII	264
2. Cuvintele cheie C/C++	265
3. Sisteme de numerație	266
<i>Operații de conversie</i>	266
4. Organizarea logică a memoriei	268
5. Reprezentarea datelor în memorie	269
6. Etapele dezvoltării programelor	270

Soluții și indicații	272
1. Elemente de bază ale limbajului C/C++	272
2. Instrucțiunile limbajului C/C++	272
3. Fișiere	278
4. Tipuri structurate de date	278
5. Stiva și coada	285
6. Funcții	286
Bibliografie	293

2. Instrucțiunile limbajului C/C++

2.1. Instrucțiunea expresie

| expresie;

Efect

Se evaluează expresia.

Exemple

```
| i++; //se incrementeaza valoarea variabilei i  
| p*=2; //se inmulteste cu 2 valoarea variabilei p  
| clrscr(); //se sterge ecranul
```

Observații

1. Ultima expresie este constituită dintr-un apel de funcție. Pentru a apela funcția standard `clrscr()`, trebuie să includem la începutul programului fișierul antet `conio.h`, în care se găsește prototipul (declarația) funcției:

| `void clrscr(void);`

ceea ce înseamnă că funcția nu are nici un parametru și nu întoarce nici un rezultat.

2. Este permis ca expresia să fie vidă. Instrucțiunea devine `;` (instrucțiunea vidă).
3. Este permisă utilizarea oricărei expresii sintactice corecte, chiar și atunci când instrucțiunea nu generează nici un efect, de exemplu: `2+5;`
4. Orice instrucțiune din limbajul C/C++ se termină cu caracterul `;`.

2.2. Instrucțiunea compusă

```
| {  
|   declaratii  
|   instructiune_1  
|   instructiune_2  
|   ...  
|   instructiune_n  
| }
```

Efect

Se execută în ordine instrucțiunile specificate.

Observații

1. Utilizarea instrucțiunilor compuse este necesară atunci când sintaxa permite executarea unei singure instrucțiuni, dar este necesară efectuarea mai multor operații (de exemplu, într-o instrucțiune *if*, *for*, *while* sau *do-while*).
2. Declarațiile care apar într-o instrucțiune compusă sunt **locale** instrucțiunii. Mai exact, ele sunt valabile numai în corpul instrucțiunii compuse, din momentul declarării lor până la sfârșitul instrucțiunii.

2.3. Instrucțiunea *if*

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

Efect

Se evaluează expresia. Dacă valoarea expresiei este diferită de 0, se execută *instructiune_1*, altfel se execută *instructiune_2*.

Exemplu

```
if (a) cout << a << " este nenul" << endl;
    else cout << a << " este nul" << endl;
```

Observații

1. Expresia se încadrează obligatoriu între paranteze rotunde.
2. Dacă *instructiune_2* este vidă, ramura *else* poate să lipsească, obținându-se o formă simplificată a instrucțiunii *if*:

```
if (expresie) instructiune
```

Exemplu

Să calculăm în variabila *max* maximul dintre *a* și *b*:

```
max=a;
if (max<b) max=b;
```

2.4. Instrucțiunea *switch*

```
switch (expresie)
{ case expresie-constanta1: secventa-instructiuni1
  case expresie-constanta2: secventa-instructiuni2
  ...
  case expresie-constantan: secventa-instructiunin
  default: secventa-instructiunin+1
}
```

Efect

Se evaluează expresia.

Se compară succesiv valoarea expresiei cu valorile expresiilor constante care etichetează alternativele `case`. Dacă se întâlnește o alternativă `case` etichetată cu valoarea expresiei, se execută secvența de instrucțiuni corespunzătoare și toate secvențele de instrucțiuni care urmează, până la întâlnirea instrucțiunii `break` sau până la întâlnirea acoladei închise, care marchează sfârșitul instrucțiunii `switch`. Dacă nici una dintre valorile etichetelor alternativelor `case` nu coincide cu valoarea expresiei, se execută secvența de instrucțiuni de pe ramura `default`¹³.

Exemplu

În funcție de valoarea variabilei de tip `char c` ('+', '-', '*', sau '/'), vom efectua operația corespunzătoare între variabilele `x` și `y`. Dacă variabila `c` are valoarea '~' sau '!', vom da mesajul "Nu e operator binar!", iar dacă `c` are orice altă valoare, vom da mesajul "Eroare".

```
switch (c)
{
    case '+':  x *= y; break;
    case '/':  x /= y; break;
    case '+':  x += y; break;
    case '-':  x -= y; break;
    case '~':
    case '!':  cout << "Nu e operator binar!"; break;
    default :  cout << "Eroare!";
}
```

Observații

1. Expresia se încadrează obligatoriu între paranteze rotunde.
2. Pe fiecare alternativă `case` este permisă executarea mai multor instrucțiuni.
3. Dacă secvența-instrucțiuni_{0..n} este vidă, ramura `default` poate lipsi (similar ramurii `else`).
4. Instrucțiunea `switch` este o generalizare a instrucțiunii `if`. Spre deosebire de `if`, care permite selectarea unei alternative din maximum două posibile, `switch` permite selectarea unei alternative din maximum `n+1` posibile. O altă diferență majoră constă în faptul că în `if` se execută instrucțiunea corespunzătoare valorii expresiei și atât, în timp ce în `switch` se execută și toate secvențele de instrucțiuni ale alternativelor `case` următoare.

13. În limba engleză, *default* înseamnă lipsă. Deci este alternativa care se alege în lipsă de altceva. Noi o vom numi alternativă implicită.

2.5. Instrucțiunea `break`

break;

Efect

Determină ieșirea necondiționată din instrucțiunea `switch`, `while`, `for` sau `do-while` în care apare.

2.6. Instrucțiunea `while`

while (expresie)
instrucțiune

Efect

Pas 1: se evaluează expresia;

Pas 2: dacă valoarea expresiei este 0, se iese din instrucțiunea `while`;

dacă valoarea expresiei este diferită de 0, se execută instrucțiunea, apoi se revine la Pas 1.

Observații

1. Instrucțiunea se execută repetat cât timp valoarea expresiei este nenulă. Pentru ca ciclul să nu fie infinit, este obligatoriu ca instrucțiunea care se execută să modifice cel puțin una dintre variabilele care intervin în expresie, astfel încât aceasta să poată lua valoarea 0, sau să conțină o operație de ieșire necondiționată din ciclu (de exemplu, `break`);
2. Dacă expresia are de la început valoarea 0, instrucțiunea nu se execută nici măcar o dată.
3. Sintaxa permite executarea în `while` a unei singure instrucțiuni, prin urmare, atunci când este necesară efectuarea mai multor operații, acestea se grupează într-o singură instrucțiune compusă.

Exemplu

Să calculăm răsturnatul numărului natural n . Răsturnatul unui număr se obține considerând cifrele numărului în ordine inversă (de la dreapta la stânga).

```
r=0; // in r vom obtine rasturnatul lui n
while (n)
{ r=r*10+n%10; // "lăpesc" ultima cifra din n la r
  n/=10; // elimin ultima cifra din n
}
```

Să urmărim pas cu pas execuția acestei secvențe de instrucțiuni, pentru $n=237$:

Linia	Efect	n	x
1	Se inițializează variabila x cu 0.	237	0
2	Se testează dacă $n \neq 0$, apoi se execută instrucțiunea compusă.	237	0
3	Se adaugă la sfârșitul lui x ultima cifră din n .	237	7
4	Se elimină ultima cifră din n .	23	7
2	Se testează dacă $n \neq 0$, apoi se execută instrucțiunea compusă.	23	7
3	Se adaugă la sfârșitul lui x ultima cifră din n .	23	73
4	Se elimină ultima cifră din n .	2	73
2	Se testează dacă $n \neq 0$, apoi se execută instrucțiunea compusă.	2	73
3	Se adaugă la sfârșitul lui x ultima cifră din n .	2	732
4	Se elimină ultima cifră din n .	0	732
2	Se testează dacă $n \neq 0$; se iese din <code>while</code> .	0	732

2.7. Instrucțiunea `do-while`

```
do
    instructiune
while (expresie);
```

Efect

Pas 1: se execută instrucțiunea;

Pas 2: se evaluează expresia;

Pas 3: dacă valoarea expresiei este 0, se iese din instrucțiunea repetitivă;
dacă valoarea expresiei este diferită de 0, se revine la Pas 1.

Observație

Spre deosebire de `while`, instrucțiunea `do-while` execută instrucțiunea specificată cel puțin o dată, chiar dacă de la început valoarea expresiei este 0, deoarece evaluarea expresiei se face după execuția instrucțiunii.

În afară de momentul evaluării expresiei (înainte sau după executarea instrucțiunii), alte diferențe între cele două instrucțiuni repetitive nu există. Prin urmare, preferăm să utilizăm instrucțiunea `while` când este necesar să testăm o condiție înainte de efectuarea unor prelucrări. Preferăm să utilizăm `do-while` când condiția depinde de la început de prelucrările din ciclu și, prin urmare, este necesar să o testăm după executarea instrucțiunii.

Exemplu

Să numărăm cifrele numărului natural memorat în variabila n :

```
nr=0;           //in nr vom obtine numarul de cifre
do
  {n/=10;      //elimin ultima cifra din n
  nr++;}       //o numar
while (n);     //actiunea se repeta cat timp n mai are cifre
```

Dacă în locul instrucțiunii `do-while` am fi utilizat o instrucțiune `while`, algoritmul nu ar fi funcționat și pentru $n=0$, deoarece testând condiția la început, nu s-ar fi executat nici o dată instrucțiunea din ciclu, deci valoarea variabilei `nr` ar fi rămas 0 (greșit! numărul 0 are o cifră!).

2.8. Instrucțiunea `for`

```
for (expresieinitalizare; expresiecontinuare; expresiereinitializare)
  instructiune
```

Efect

Pas 1: se evaluează expresia de inițializare;

Pas 2: se evaluează expresia de continuare;

Pas 3: dacă valoarea expresiei de continuare este 0, se iese din instrucțiunea repetitivă `for`;

dacă valoarea expresiei de continuare este diferită de 0:

- se execută instrucțiunea;
- se evaluează expresia de reinițializare;
- se revine la Pas 2.

Observații

Instrucțiunea `for` este tot o instrucțiune repetitivă, ca și `while` și `do-while`. Nu este o instrucțiune strict necesară, ea poate fi simulată cu instrucțiunea `while` astfel:

```
expresieinitalizare;
while (expresiecontinuare)
  { instructiune
  expresiereinitializare; }
```

Totuși, majoritatea programatorilor preferă să utilizeze instrucțiunea `for`, deoarece este un instrument puternic, flexibil, care ne permite să exprimăm foarte concis prelucrări de natură repetitivă.

Exemplu

Fie n un număr natural ($n < 10$) citit de la tastatură. Să se calculeze:

$n! = 1 \cdot 2 \cdot \dots \cdot n$ (n factorial). Prin definiție, $0! = 1$.

Observăm că pentru orice $n > 0$ are loc relația:

$n! = 1 \cdot 2 \cdot \dots \cdot (n-1) \cdot n = (1 \cdot 2 \cdot \dots \cdot (n-1)) \cdot n = (n-1)! \cdot n$