**Date în C++**

**Tipurile principale simple de date în C++**

**Numere întregi**: **int** (4 bytes), unsigned int (4 bytes),

short int (2 bytes), unsigned short int (2 bytes),

(long==int – 4 bytes),

long long (8 bytes), unsigned long long (8 bytes),

char (1 byte), unsigned char (1 byte)

Diferenţele între ele constau în plaja de valori pe care o pot lua

Ex: int aparţine de la -2147483648 la 2147483647

orice depăşire în vreun sens, NU dă eroare, însă reţine alte valori

short int aparţine de la -32768..32767

char folosit ca tip întreg permite valori între -128..127

unsigned char folosit ca tip întreg permite valori între 0..255

tipul int ocupă 4 octeţi (bytes)

short int – 2 octeţi

char – 1 octet

**Numere reale**: float – memorează aprox. 7-8 cifre, ocupă 4 octeţi

**double** – memorează aprox. 15-16 cifre, ocupă 8 octeţi

**Caractere**: **char**, unsigned char

Un caracter se memorează prin codul său ASCII – un număr între 0..255.

Cele de bază sunt între 0..127

**Datele dintr-un program C++ se mai clasifică în constante şi variabile**

**Constantele în C++:**

**numerice întregi**: numerele ca atare, ca la mate

Ex: 0, 14, -5, 33

**numerice reale**:

• numerele ca atare, în care separatorul zecimal este punctul: 3.14, -8.23

• numerele în forma ştiinţifică, adică **ME**+**P**

**M** se numeşte mantisă

**P** se numeşte putere sau exponent

Aceste numere au valoarea aritmetică M\*10P

Ex: **1.234**E+**4** = 1.234\*104 = 12340

**1.234**E-**2** = 1.234\*10-2 = 0.01234

**caracter (unul singur):**

caracterele ca atare, scrise între apostrofuri: 'A', 'B', '2', '$', 'a'

Obs: există câteva caractere neafişabile, care au coduri. Aceste coduri încep cu \.

Ex: '\n' – trecere la rând nou (endline)

'\t' – tab

'\\' – backslash

'**\'**' – apostrof

'**\"**' – ghilimele

**şir de caractere (string):**

caracterele ca atare, scrise între ghilimele: "ana are copaci"

"ana este eleva la colegiul \"Unirea\"" "ana are\nmere"

**Variabile în C++**

O variabilă este zonă din memoria calculatorului capabilă să reţină (să memoreze) o valoare. În orice moment putem folosi valoarea memorată (operaţie care se numeşte "evaluarea variabilei") sau putem memora o altă valoare (caz în care valoarea precedentă se pierde)

O variabilă trebuie declarată înainte de a fi utilizată.

Acest lucru se face prin:

int a;

int x,y;

double q;

int x=3,y,z=x+1;

...

Orice variabilă are un nume (identificator) şi un tip de date.

**Constante literale în C++**

Pe lângă constantele de care am povestit mai sus (datele ca atare) există constante pe car ele putem folosi prin identificatori, la fel ca pe variabile, doar că atribuirea de valori către acestea nu este permisă (va da eroare)

Declararea şi iniţializarea lor se face prin

const tip identif\_constantă=valoare;

Ex:

const int a=12;

folosind a este ca şi cum am scrie 12. Orice încercare de a schimba valoarea lui a ne dă eroare.

**Expresii în C++**

O expresie implică efectuarea unor operaţii.

În C++ avem operaţii:

• aritmetice: +, -, \*, /, %

• relaţionale: <, <=, >, >=, ==, !=

• logice: ||, &&, !

**Operaţiile aritmetice** se efecutează ca la matematică. Avem câteva particularităţi:

- operatorii +, - se fac ultimii, ca la mate, în ordinea apariţiei

- operatorii \*, /, % au prioritate, ei se fac primii, în ordinea apariţiei

Ex: 12\*14%10 = 8 15/4\*4 = 12

12\*(14%10) = 48

**operatorul**

**/**  în C++ funcţionează diferit în funcţie de membrii săi.

Şi anume:

dacă ambii parametri sunt numere întregi, ne dă câtul întreg (FĂRĂ ZECIMALE!)

dacă cel puţin un parametru este real, ne dă câtul real, cu zecimale.

Forţarea unui membru pentru a fi văzut de un tip anume dorit de noi:

(tip)valoare

Ex:

cout<<10/3; → afişează 3

cout<<10/4; → afişează 2

cout<<10/4.0; → afişează 2.5

cout<<(double)10/4; → afişează 2.5

cout<<(int)'A'; → 65

cout<<(int)3.785; → 3

cout<<(int)3.185; → 3

cout<<(int)-3.185; → -3

cout<<(int)-3.785; → -3

**operatorul %**

calculează restul împărţirii întregi dintre membrii săi.

E obligatoriu ca ambii membri să fie întregi. Dacă nu e aşa, va da eroare !

Ex:

10%3 → 1 13%3 → 1 16%3 → 1

11%3 → 2 14%3 → 2

12%3 → 0 15%3 → 0

(O împărţire cu rest arată aşa:

Deîmpărţit | Împărţitor Deîmpărţit:Împărţitor=Cât

+------------ ...

========== | Câtul ======

Restul Restul

Teorema împărţirii cu rest zice că: D = C\*I+R

iar restul poate fi cuprins între 0 şi I-1

)

Obs: La info nu e chiar aşa, mai precis la numerele negative, câtul va respecta regula semnelor.

Ex: 13/-5 → -2 -13/5 → -2 -13/-5 → 2

De asemenea, la info restul NU e ca la mate: se calculează făcând abstracţie de semne, iar apoi preia semnul deîmpărţitului

Ex: 13%-5 → 3 -13%5 → -3 -13%-5 → -3

Chiar şi aşa (cu cele de mai sus) se păstrează egalitatea teoremei împărţirii cu rest.

Expresii aritmetice uzuale:

Dacă x este număr natural, atunci:

x%10 reprezintă ultima cifră (a unităţilor)

x/10 reprezintă numărul de la care s-a "şters" ultima cifră

x%100 reprezintă numărul format de ultimele două cifre ale valorii date

x/100 reprezintă numărul de la care s-au şters ultimele două cifre

x/10%10 reprezintă cifra zecilor numărului dat

Există o serie de funcţii, aflate în biblioteca <cmath> care ajută la efectuarea unor operaţii matematice.

Daca utilizam <math.h> in loc de cmath, în acesta nu găsim decât fabs, pentru abs trebuie

să includem biblioteca <stdlib.h>

Cele mai folosite:

sqrt(x) – rădăcina pătrată (dă rezultat real)

abs(x) – modul (valoarea absolută) – pentru numere **întregi**

**f**abs(x) – modul (valoarea absolută) – pentru numere **reale**

**Expresiile relaţionale**

Au ca scop verificarea valorii de adevăr a unei comparaţii (relaţie) dintre două valori.

O astfel de expresie are ca rezultat 0 (FALS) sau 1 (ADEVĂRAT)

Ex:

1<=5 → are valoarea 1

1>5 → are valoarea 0

Expresii relaţionale uzuale:

a%2==0 → expresia este adevărată **dacă şi numai dacă** numărul a este par

a%2!=0 → expresia este adevărată **dacă şi numai dacă** numărul a este impar

a%b==0 → expresia este adevărată dacă şi numai dacă numărul a se divide la b.

sqrt(x)==(int)sqrt(x) → expresia este adevărată dacă şi numai dacă numărul x este

un pătrat perfect

**Expresiile logice**

Au ca scop compunerea mai multor valori de adevăr (0 şi 1) pentru a verifica dacă anumite lucruri se întâmplă simultan sau cel puţin unul este adevărat.

operatorii logici sunt: || (sau-ul logic, conjuncţia) – se face ultima

&& (şi-ul logic, disjuncţia) – se face înainte de ||

! (negaţia) – se face prima

Tabelele de adevăr sunt:

|  |  |  |
| --- | --- | --- |
| a | b | a||b |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| a | b | a&&b |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

|  |  |
| --- | --- |
| a | !a |
| 0 | 1 |
| 1 | 0 |

Negaţia unei expresii relaţionale schimbă semnul, ţinând cont şi de egalitate:

!(a<b) ⇔ a>=b

!(a<=b) ⇔ a>b

!(a>b) ⇔ a<=b

!(a>=b) ⇔ a<b

!(a==b) ⇔ a!=b

!(a!=b) ⇔ a==b

Negaţia expresiilor compuse:

!(E1 || E2) = !E1 && !E2

!(E1 && E2) = !E1 || !E2

**Limbajul pseudocod**

Foloseşte la descrierea pur teoretică a unor algoritmi, de regulă nu foarte laborioşi.

Este exprimat în limba celui care-l utilizează şi nu are reguli foarte stricte.

Astfel, în pseudocod este permisă scrierea mai apropiată de matematică, folosirea unor operatori diferiţi dacă ei sunt sugestivi, şi un oarecare grad de libertate.

În pseudocod, operaţia de împărţire NU este ca în C++, adică întotdeauna se consideră că dă rezultat real.

Din acest motiv, dacă în pseudocod întâlnim [a/b] acest lucru în C++, dacă a şi b sunt întregi, îl vom transcrie ca a/b.

Iată principalele instrucţiuni în pseudocod şi transcrierile lor în C++:

| Pseudocod | C++ |
| --- | --- |
| 1. Citirea:  citeşte v1,v2,v3,...  Obs: toate entităţile citite trebuie să fie VARIABILE | cin>>v1>>v2>>v3>>....; |
| 2. Afişarea:  scrie e1,e2,e3,...  Obs: entităţile afişate pot fi constante, variabile, expresii | cout<<e1<<e2<<e3....; |
| 3. Atribuirea:  variabilă ← valoare  Obs: în membrul drept (valoare) putem avea o expresie, o constantă sau o altă variabilă | variabilă = expresie; |
| 4. Structura condiţională / de decizie / alternativă / de tip "dacă"  **4a.**  ┌dacă **condiţie** atunci  │ **secv\_adevăr**  └■  **4b.**  ┌dacă **condiţie** atunci  │ **secv\_adevăr**  │altfel  │ **secv\_fals**  └■ | **4a.**  if(**condiţie**)  {  **secv\_adevăr;**  }  **4b.**  if(**condiţie**)  {  **secv\_adevăr;**  }  else  {  **secv\_fals;**  }  Obs: Dacă vreuna dintre secvenţe (adevăr sau fals) este formată dintr-o singură instrucţiune, atunci PUTEM omite parantezele acolade. |
| 5. Repetitiva cu test anterior / iniţial / precondiţionată / de tip "cât timp"  ┌cât timp **condiţie** execută  │ **secv**  └■  Dacă din start condiţia este falsă, nu face nimic.  Dacă e adevărată se apucă de repetat şi tot repetă cât timp condiţia rămâne adevărată | while(**condiţie**)  {  **secv;**  } |
| 6. Repetitiva cu test posterior / final / postcondiţionată / de tip "repetă..până când"  ┌repetă  │ **secv**  └până când **condiţie**  Mai întâi se trece "ca-n brânză" şi se execută odată secvenţa.  După executare verifică dacă este falsă condiţia şi, dacă da, continuă repetarea până când condiţia devine adevărată  Obs: Datorită programatorilor de C/C++, pentru care repetitiva cu test final este  do {...} while(cond);  se acceptă şi oricare din scrierile următoare în pseudocod:  ┌repetă  └cât timp ...  ┌execută  └cât timp ... | do  {  **secv;**  }while(**neg** **condiţie**); |
| 7. Repetitiva cu contor / de tip "pentru" (for)  7a. for crescător  ┌pentru **contor**←**li**,**lf** execută  │ **secv**  └■  contor ia rând pe rând, crescător, din 1 în 1, toate valorile dintre li şi lf şi, pentru fiecare dintre ele, execută secvenţa.  Dacă din start li>lf – nu execută nimic.  7b. for **descrescător**  ┌pentru **contor**←**li**,**lf,-1** execută  │ **secv**  └■  La fel, doar că merge descresc. de la li la lf.  Dacă din start li<lf – nu execută nimic.  Obs: În pseudocod se consideră greşeală dacă în cadrul **secv** unui for se modifică **contorul**, **li** sau **lf**. (!ceea ce în C++ este permis!) | 7a.  for(**contor**=**li;contor**<=**lf**;**contor**++)  {  **secv;**  }  7b.  for(**contor**=**li;contor**>=**lf**;**contor**--)  {  **secv;**  } |