**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)

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

Ex: int aparține de la -2147483648 la 2147483647

orice depășire în vreun sens, NU dă eroare, însă reține alte valori

short int aparține 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 octeți (bytes)

short int – 2 octeți

char – 1 octet

Domeniul de valori pentru fiecare, datorită memorării în binar de către calculator se calculează astfel:

- fie x = numărul de biți din tipul care ne interesează (8 \* numărul de bytes)

Domeniul de valori pentru întregii cu semn este: -2x-1 .. 2x-1-1

pentru întregii fără semn este: 0..2x-1

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

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

**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 și 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 științifică, adică **ME**+**P**

**M** se numește mantisă

**P** se numește 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 neafișabile, care au coduri. Aceste coduri încep cu \.

Ex: '\n' – trecere la rând nou (endline) - este MULT mai eficient decât **endl**.

'\t' – tab

'\\' – backslash

'**\'**' – apostrof

'**\"**' – ghilimele

**șir 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 rețină (să memoreze) o valoare. În orice moment putem folosi valoarea memorată (operație care se numește "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;

int suma\_elemente,\_x,y0;

...

Orice variabilă are un nume (identificator) și 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 și inițializarea lor se face prin

const tip identif\_constantă=valoare;

Ex:

const int a=12;

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

**Expresii în C++**

O expresie implică efectuarea unor operații.

În C++ avem operații:

• aritmetice: +, -, \*, /, %

• relaționale: <, <=, >, >=, ==, !=

• logice: ||, &&, !

**Operațiile aritmetice** se efectuează ca la matematică. Avem câteva particularități:

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

- operatorii \*, /, % au prioritate față de + și -, ei se fac primii, în ordinea apariției

**operatorul**

**/**  în C++ funcționează diferit în funcție de membrii săi.

Şi anume:

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

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

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

(tip)valoare

Ex:

cout<<10/3; → afișează 3

cout<<10/4; → afișează 2

cout<<10/4.0; → afișează 2.5

cout<<(double)10/4; → afișează 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 % (modulo)**

calculează restul împărțirii întregi dintre membrii săi.

E obligatoriu ca ambii membri să fie întregi. Dacă nu e așa, 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 împărțire cu rest arată așa:

Deîmpărțit | Împărțitor Deîmpărțit:Împărțitor=Cât

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

========== | Câtul ======

Restul Restul

Quotient = Cât (eng.)

Remainder = Rest (eng.)

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

iar restul poate fi cuprins între 0 și I-1

)

(Funcția f(x)=x%y, pentru x numere naturale este o funcție periodică de perioadă y)

Expresii aritmetice uzuale:

Dacă x este număr natural, atunci:

x%10 reprezintă ultima cifră (a unităților)

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

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

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

x/10%10 reprezintă cifra zecilor numărului dat

x/100%10 reprezintă cifra sutelor numărului dat

Există o serie de funcții, aflate în biblioteca <cmath> care ajută la efectuarea unor operații 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** (de fapt, în noile implementări, dacă parametrul x al lui abs este număr întreg atunci abs are rezultat întreg, dacă este real, are rezultat real)

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

**Obs:** La bac, dacă folosiți bibliotea <cmath>, se folosește abs atât pentru numere întregi cât și pentru numere reale.

Mai precis, dacă facem abs(număr\_întreg) avem rezultat întreg, iar abs(număr\_real) ne dă rezultat real.

**Expresiile relaționale**

Au ca scop verificarea valorii de adevăr a unei comparații (relație) 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 relaționale uzuale:

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

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

(dat fiind că %2 pt. numere negative poate fi -1, e foarte recomandat să

nu scrieți if(a%2==1) ci să folosiți ACEASTĂ formă)

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

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

un pătrat perfect

**Expresiile logice**

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

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

&& (și-ul logic, conjuncția) – se face înainte de ||

! (negația) – 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 |

Negația unei expresii relaționale schimbă semnul, ținând cont și 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

Negația expresiilor compuse:

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

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

Exemplu: fie variabila a de tip int.

Expresia logică prin care verificăm dacă a este o cifră e următoarea:

a>=0 && a<=9

Negația: !(a>=0 && a<=9) ⇔ !(a>=0) || !(a<=9) ⇔ a<0 || a>9

**Limbajul pseudocod**

Folosește la descrierea pur teoretică a unor algoritmi, de regulă nu foarte laborioși.

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

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

În pseudocod, operația de împărțire 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 și b sunt întregi, îl vom transcrie ca a/b.

Iată principalele instrucțiuni în pseudocod și transcrierile lor în C++:

| Pseudocod | C++ |
| --- | --- |
| 1. Citirea:  citește v1,v2,v3,...  Obs: toate entitățile citite trebuie să fie VARIABILE | cin>>v1>>v2>>v3>>....; |
| 2. Afișarea:  scrie e1,e2,e3,...  Obs: entitățile afișate 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 condițională / de decizie / alternativă / de tip "dacă"  **4a.**  ┌dacă **condiție** atunci  │ **secv\_adevăr**  └■  **4b.**  ┌dacă **condiție** atunci  │ **secv\_adevăr**  │altfel  │ **secv\_fals**  └■ | **4a.**  if(**condiție**)  {  **secv\_adevăr;**  }  **4b.**  if(**condiție**)  **{**  **secv\_adevăr;**  **}(!!atenție la acolada asta)**  else  {  **secv\_fals;**  }  Obs: Dacă vreuna dintre secvențe (adevăr sau fals) este formată dintr-o singură instrucțiune, atunci PUTEM omite parantezele acolade. |
| 5. Repetitiva cu test anterior / inițial / precondiționată / de tip "cât timp"  ┌cât timp **condiție** execută  │ **secv**  └■  Dacă din start condiția este falsă, nu face nimic.  Dacă e adevărată se apucă de repetat și tot repetă cât timp condiția rămâne adevărată | while(**condiție**)  {  **secv;**  } |
| 6. Repetitiva cu test posterior / final / postcondiționată / de tip "repetă..până când"  ┌repetă  │ **secv**  └până când **condiție**  Mai întâi se trece "ca-n brânză" și se execută odată secvența.  După executare verifică dacă este falsă condiția și, dacă da, continuă repetarea până când condiția devine adevărată  Obs: Datorită programatorilor de C/C++, pentru care repetitiva cu test final este  do {...} while(cond);  se acceptă și oricare din scrierile următoare în pseudocod:  ┌execută  └cât timp ...  ┌repetă  └cât timp ... | do  {  **secv;**  }while(**neg** **condiție**); |
| 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 și lf și, pentru fiecare dintre ele, execută secvența.  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 greșeală 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;**  } |