Introducere în programare, cu suport limbajul C/C++

Un **program** de calculator presupune interacțiunea dintre utilizator și mașină în următorul mod: utilizatorul transmite mașinii anumite date, mașina le prelucrează și apoi transmite înapoi utilizatorului alte date.

Datele transmise de utilizator se numesc **date de intrare**, iar acestrea pot fi introduse folosind diverse dispozitive de intrare (tastatura, mouse, microfon ...). Datele transmise înapoi de către program se numesc **date de ieșire** și sunt primite de către utilizator prin diverse dispozitive de ieșire (monitor, imprimantă, boxe ...).

Cel care creează un program va trebui să cunoască instrumentele necesare pentru a-l scrie (adică un alt program, special, numit **limbaj de programare**) și să le folosească în modul potrivit pentru a realiza corect interacțiunea despre care s-a scris mai sus.

Pentru simplitate și înțelegerea principiilor de bază, datele programelor ce le vom scrie (atât cele de intrare cât și cele de ieșire) vor fi numere și presupunem că dispozitivele prin intermediul cărora realizăm interacțiunea cu mașina de calcul sunt tastatura (pentru datele de intrare) și monitorul (pentru datele de ieșire).

Când proiectăm un program trebuie să descriem tot ceea ce se întâmplă în momentul executării acestuia.

Există două elemente fundamentale despre care vom discuta în amănunt: **datele** (am amintit mai sus) și **instrucțiunile**.

Pentru date se rezervă spațiu în **memoria intrenă** (RAM) în timpul executării programului.

Instrucțiunile ni le putem imagina ca instrumente pe care **microprocesorul** le are pentru a prelucra datele, instrumente (sau le mai putem spune comenzi) pe care noi le putem folosi în program. Așadar avem în vedere două lucruri principale, date și instrucțiuni.

Iată o secvență de program în limbajul C/C++:

```
int a, b;
cin>>a>>b;
cout<<(a+b)*2
;
```

Pe prima linie se anunță că vom folosi două date variabile care se numesc `a` și `b`. Acest lucru poate fi intrerpretat și în modul următor: în memoria RAM se vor rezerva două zone, în fiecare dintre ele noi vom putea ulterior să stocăm o valoare întreagă.

Linia a doua și a treia reprezintă instrucțiuni (vom vedea ulterior ca este vorba de ceva putin mai complex, dar deocamdată ne putem imagina că `cin` și `cout` sunt cuvinte care desemnează instrucțiuni). Astfel, instrucțiunea de pe linia a doua are semnificația de cerere de date de la tastatură (când programul va executa acea instrucțiune, utilizatorul va fi invitat să introducă două numere, primul va fi stocat în zona de memorie numită `a` iar al doilea în zona de memorie numită `b`). Aceasta este deci instrucțiunea prin care se preiau datele de intrare. Pe linia a treia este o instrucțiune prin care se trimit către utilizator (prin intermediul ecranului), datele de ieșire. În afară de datele `a` și `b` (care ulterior vom vedea că se cheamă **variabile**), în acest program

mai apare o dată **constantă** (2) dar și operații cu datele - plus, asterix și parantezele - pe care le vom numi mai departe **operatori**.

Când vom scrie un program va trebui să anunțăm în prealabil datele variabile pe care dorim să le folosim și va trebui să indicăm instrucțiunile, în ordinea în care dorim să se execute.

Un program compus din secvența de mai sus funcționează astfel: La pornire rezervă spațiu în memoria RAM pentru a se putea stoca acolo două numere întregi, apoi cere utilizatorului să introducă două numere și în final afișează pe ecran dublul sumei celor două numere introduse. Observăm că dacă utilizatorul rulează și a doua oară programul, el va putea introduce alte numere decât la prima rulare și, evident, se va obține alt rezultat. Se deduce de aici un principiu de bază în programare: **generalitatea**.

Date

Ne ocupăm în continuare de modul de lucru cu date în limbajul C/C++. Datele unui program se pot clasifica după mai multe criterii, deocamdată ne interesează ierarhizarea după natura informației pe care o pot stoca. Din acest punct de vedere, vom considera pentru început că avem date de două feluri (**tipuri**): date care pot memora numere întregi (spunem pe scurt **date întregi**) și date care pot memora numere reale (pe scurt, **date reale**). Așa cum am spus și la început, vom considera că toate datele sunt numere.

În cadrul celor două categorii (întregi și reale) vom vedea că datele se pot subclasifica (în funcție de memoria ocupată și, implicit, de intervalul din care pot lua valori), dar despre acest lucru vom discuta mai tarziu.

Din alt punct de vedere, datele pot fi clasificate în variabile și constante. În exemplul de program de mai sus, am indicat `a` și `b` ca fiind variabile. Acestora le putem schimba valoarea pe care o stochează pe parcursul programului. Constantele sunt numere pe care le indicăm direct în program.

Pentru prezentarea următoarelor noțiuni vom lucra cu date constante.

În limbajul C/C++ constantele întregi se pot specifica în mai multe feluri. Deocamdată vom folosi scrierea lor în modul în care se folosesc și la matematică, adică succesiune de cifre zecimale precedată eventual de + sau -

Exemple: `10, +77, 0, -199`

Și pentru constantele reale sunt mai multe moduri de scriere în limbaj. Noi îl vom folosi pe următorul: succesiune de cifre zecimale, precedată eventual de + sau − și în care apare semnul `.` (punct) pentru a separa partea întreagă de cea fracționară (atenție, nu virgula, ca la matematică).

Exemple: `12.34  -2.08  5.0`

Mai sus virgula este doar separator folosit de noi.

În afară de prelucrarea datelor prin intermediul a diverse instrucțiuni, mai dispunem de un instrument pentru a obține din date alte date: **operatorii**.

Aceștia reprezintă semne care au o semnificație bine definită în limbaj. Pentru fiecare operator vom indica la ce fel de date se poate aplica (vom vedea că nu putem folosi neapărat orice operator cu orice fel de date).

Datele care se pun lângă operatori se numesc **operanzi**.

Exemplu:

`17+22`	Este vorba aici de operatorul + care se aplică la două date, ambele constante. În cadrul construcției de mai sus spunem că `17` și `22` sunt operanzi.

La fiecare operator este clar reglementat în limbaj modul în care se pot aplica. Unii operatori se pot aplica la două date, și se numesc **binari** (operatorul din exemplul de mai sus), alții se pot aplica la o singură dată (se numesc **unari**) și vom vedea că există și operatorul **ternar**.

Iată, în continuare, un prim set de operatori în limbajul C/C++.

Operatorii aritmetici

Ei sunt:

+ (plus)	pentru operația de adunare
− (minus)	pentru operația de scădere
* (înmulțit)	pentru operația de înmulțire
/ (slash)	pentru operația de împărțire reală sau, după caz, de cât al împărțirii întregi
% (procent)	pentru operația de rest al împărțirii întregi

Toți operatorii prezentați mai sus sunt binari. **Operatorii +, −, *, aplicați la două date întregi oferă rezultat întreg, iar dacă este real cel puțin un operand, atunci și rezultatul va fi real.** Vom vedea mai departe că este extrem de important să cunoaștem care este tipul de date al rezultatului.

Exemple:

Expresie	***Explicație***
`12+23`	Rezultat `35`, de tip întreg
`12.3*2`	Rezultat `24.6`, de tip real
`0.0+3`	Rezultat `3.0`, de tip real
`12-24`	Rezultat `-12`, de tip întreg
`2.5*4`	Rezultat `10.0`, de tip real

Am folosit anterior noțiunea de *expresie*. O vom mai face, iar acum o vom defini. **Numim expresie orice construcție corectă în limbaj care are, după realizarea calculelor, o valoare.** Deocamdată putem spune că:
- o constantă este o expresie;
- orice aplicare corectă de operatori asupra altor expresii este o expresie.

Vom extinde ulterior definiția noțiunii cu alte entități.

Operatorul / aplicat la două date întregi are ca rezultat câtul împărțirii (adică tot un întreg). Dacă este real cel puțin un operand, atunci rezultatul va fi real (deci rezultatul va putea conține și zecimale).

Exemple

Expresie	***Explicație***
`12/4`	Rezultat `3`, de tip întreg
`12/4.0`	Rezultat `3.0`, de tip real
`-8.2/2`	Rezultat `-4.1`, de tip real
`1/4`	Rezultat `0`, de tip întreg
`1.0/4`	Rezultat `0.25`, de tip real

Observăm că rezultatul respectă regula semnelor cunoscută la matematică la operația de împărțire(ca și la înmulțire, de altfel).

Operatorul % se aplică doar la date întregi. Nu este deci permisă plasarea sa lângă un operand real. Acest lucru înseamnă că, o încercare totuși de punere a unei date reale lângă % produce o **eroare de compilare** a programului.

Ce este o eroare de compilare?

După scrierea programului într-un limbaj de programare, se va cere întâi o analiză a sa pentru a se stabili dacă s-au respectat regulile limbajului. Când se întâlnesc elemente neconsiderate corecte în limbaj, apare o astfel de eroare. Este ca si cum am folosi într-o limbă cuvinte care nu îi aparțin.

Rezultatul aplicării lui procent între cei doi operanzi întregi este restul împărțirii celor două numere (deci o valoare de tip întreg).

Exemple

Expresie	***Explicație***
`12%4`	Rezultat `0`, de tip întreg
`12%4.0`	Eroare de compilare
`1%4`	Rezultat `1`, de tip întreg
`17%5`	Rezultat `2`, de tip întreg

Într-o expresie pot apărea mai mulți operatori. Ordinea în care se aplică asupra datelor respectă principiile:
- fiecare operator are o anumită prioritate, se aplică mai întâi operatorii cu prioritatea cea mai mare;
- operatorii de aceeași prioritate se aplică de la stânga la dreapta (vom vedea ulterior că sunt și unele excepții);
- ordinea aplicării operatorilor poate fi modificată prin folosirea parantezelor rotunde; pot fi folosite mai multe seturi de paranteze (toate rotunde), corect asociate;

Dintre operatorii prezentați mai sus, `*`, `/` și `%` sunt de prioritate mai mare (dar toați trei de aceeași prioritate), iar `+` și `−` de prioritate mai mică (și ei, ambii, pe același nivel de prioritate).

Exemple

Expresie	***Rezultatul evaluării expresiei***	***Explicație***
`2+3*5`	`17`	Se aplică mai întâi operatorul `*` (are prioritate mai mare), apoi se adună `2` cu `15`
`(2+3)*5`	`25`	Se realizează întâi operația `+` (întrucât se află între paranteze) apoi rezultatul se înmulțește cu `5`
`125/10%10`	`2`	Avem doi operatori de aceeași prioritate, așadar se realizează întâi calculul `125/10` (operatorul din stânga), apoi se calculează restul împărțirii lui `12` la `10`
`((2+3)*4-6)*2`	`28`	
`2.0*2%2`	Eroare de compilare	Se încearcă aplicarea operatorilor de la stânga la dreapta (ambii fiind de aceeași prioritate), dar în urma calculului `2.0*2` s-ar obține un rezultat real, deci acesta nu poate reprezenta un operand lângă operatorul `%`.
`(2+3))-3*(4`	Eroare de compilare	Parantezele rotunde nu sunt asociate corect
`2$4`	Eroare de compilare	Apare un simbol necunoscut

`27/(12%4)`	Eroare la executare	Observați că alături este precizată o altfel de eroare, nu de compilare. Acest lucru se întâmplă întrucât nici calculatoarele nu pot realiza calcule nedefinite la matematică (așa cum este aici împărțirea la `0`). Nu apare eroare de compilare întrucât sunt respectate regulile de scriere, așa că programul va porni în executare. Problema este că undeva pe parcurs se realizează o operație nepermisă. Putem extinde comparația cu o limbă vorbită, făcută la definirea erorii de compilare: în acest caz este ca și cum am vorbi corect într-o limbă străină (adică folosim cuvinte existente) însă nu spunem ce trebuie :D. Observație: Exemplul de aici este didactic. Unele compilatoare mai noi evaluează deja expresiile și e posibil să detecteze devreme erori ca aceasta, semnalându-le de la compilare. Dar dacă în loc de constante am avea variabile cu valorile care apar în acest exemplu, compilatorul nu ar mai putea face nimic. Despre variabile vom învăța în lecția viitoare.

Este foarte important să interpretăm tot ce folosim în limbaj în funcție de semnificația elementelor care apar. Nu înseamnă că dacă se cunoaște de ala altă disciplină de studiu o anume notație, atunci automat limbajul C/C++ o recunoaște și el. Pe de altă parte, poate a părut banală prezentarea, mai sus, a operatorului de adunare. Trebuie considerat că nu am făcut altceva decât să îl prezentăm așa cum limbajul C/C++ îl recunoaște.

Operatorii `+` și `-` pot fi utilizați și ca operatori unari (se aplică la un singur operand). Limbajul are mecanism prin care, în funcție de context, îi opate identifica. Operatorii unari au cea mai mare prioritate în limbaj.

Exemple

Expresie	***Rezultatul evaluării expresiei***	***Explicație***
`3 * -(2 + 7)`	`-27`	Aici operatorul `+` este binar, iar `-` este unar. Operandul la care el se aplică este rezultatul parantezei. Așadar se realizează în final calculul `3 * -9`
`3 - -(2 + 7)`	`12`	Ca și mai sus, al doilea operator `-` este unar, însă primul este binar. Ultima dată se aplică operatorul binar `-` la operanzii `3` și `-9`. Observați că între cele două semne `minus` este un spațiu. Este foarte important acest lucru pentru că vom vedea ulterior că două semne `-` lipite reprezintă în C/C++ alt operator.

Tema

Întrebări și exerciții

1. Competați coloana din dreapta cu rezultatul evaluarii expresiei aflată în celula corespunzătoare din stânga (valoare și tip). Am folosit noțiunea de *evaluare a unei expresii*, semnificația fiind aceea de a aplica toți operatorii, după regulile limbajului, până la obținerea unui rezultat.

`1+2.2`	`3.2`, real
`12*12`	
`-10 * 12`	
`(12 + 3*5)/4`	
`43569 % 10`	
`43569 /10 % 10`	
`43569 %100 / 10`	
`43569 /100 % 10`	
`43569 %1000 / 100`	
`43569 /1000 % 10`	
`43569 %10000 / 1000`	

2. Răspundeți la următoarele întrebări:

- Ce este un operator?
- Ce este un operand?
- Ce este o expresie?
- Ce înseamnă "evaluarea unei expresii"?
- Clasificați operatorii învățați după prioritate.
- Ce tipuri de erori am prezentat mai sus? Descrieți în ce context apare fiecare dintre ele.
- Cum se clasifică operatorii în funcție de numărul de operanzi la care se aplică ?