Operatorii relaționali, operatorii logici și instrucțiunile de selecție, if și switch

Ca și categorii mari de operatori am prezentat deja pe cei aritmetici (+, -, *, /, %) și pe cei de atribuire (=, +=, -=, *=, /=, %=, ++, --).

Pentru utilizarea lor toate datele au fost considerate de tip numeric.

Cele două categorii de operatori pe care îi vom prezenta vor lucra în principal tot cu date numerice, dar vom vedea că în anumite cazuri aceste date sunt interpretate de program din punct de vedere logic. Sunt instrucțiuni (cum vom vedea chiar în această lecție că este `if`) care cer scrierea unei expresii cu rezultat *adevărat* sau *fals*, rezultat în funcție de care se va lua decizia de continuare a programului într-un fel sau altul. Aceste expresii logice sunt de fapt tot expresii numerice, dar acolo unde limbajul le interpretează ca logice se procedează astfel: **dacă expresia are valoare nenulă se consideră adevărată iar dacă are valoarea 0 se consideră falsă**. Subliniem deci că în limbajul C/C++ valorile de adevăr se obțin tot cu expresii numerice. Vom face așadar, acolo unde este cazul, și abuz de limbaj, folosind termeni ca: *expresie logică*, rezultat *adevărat* (sau *true*), rezultat *fals* (sau *false*). Uneori chiar ne vom referi la adevărat prin valoarea `1` iar la fals prin valoarea `0`.

Operatorii relaționali (de comparare)

Sunt următorii: <(mai mic), >(mai mare), <=(mai mic sau egal), >=(mai mare sau egal), == (egal), != (diferit).

Din start subliniem diferența dintre operatorul de atribuire (care folosește un singur simbol *egal*) și cel de comparare (care folosește două simboluri *egal* consecutive).

Acești șase operatori sunt binari, se aplică la date numerice, iar rezultatul este 0 sau 1.

Modul de aplicare a lor este următorul: se evaluează cei doi operanzi și apoi se compară valorile obținute. Dacă acestea sunt în relația matematică indicată prin operator, rezultatul este `1`. Dacă nu, rezultatul este `0`. Rezultatul este mai degrabă util să îl interpretăm din punct de vedere logic, fiind adevărat sau fals.

Ca regulă generală, acest grup de operatori este mai puțin prioritar decât cei aritmetici, adică nu trebuie să ne preocupăm în a pune paranteze în cei doi membri care se compară dacă acolo apar operatori aritmetici. Ei se încadrează și în regula generală, aceea că operatorii de aceeași prioritate se aplică de la stânga la dreapta.

Exemple:

Expresia	Rezultat
`1 <= 1`	`1`
`1 < 1`	`0`
`3 < 1`	`0` (Atenție că de multe ori ne putem grăbi să spunem că nu este corectă o astfel de expresie. Ea este corectă, dar cu rezultat fals)
`2 == 2`	`1`
`0 == 0`	`1`
`3 <= 1 + 2`	`1`
`2 == 2 == 2`	`0` (Este vorba despre doi operatori de aceeași prioritate, se aplică mai întâi cel din stânga, rezultatul `2==2` va fi adevărat, adică `1`, apoi se aplică celălalt operator, dar în urma comparării `1==2` rezultatul este fals)

`2*3-4 <= 9*9`	`1`
`(2*3-4) <= (9*9)`	`1` (expresie echivalentă cu aceea de mai sus)
`2.3 <= 3`	`1` (operanzii pot fi atât reali cât și întregi)

Evident că pot interveni și variabile sau apeluri de funcții.

Operatorii logici

Despre grupele de operatori studiate anterior putem spune și următoarele: operatorii aritmetici se aplică la date numerice și dau rezultat numeric iar operatorii relaționali se aplică la date numerice și dau rezultat logic.

Operatorii logici se aplica la date logice și rezultatul pe care îl dau este tot logic. Mai sintetic:

Operand 1	*Operator*	*Operand 2*	*Rezultat*
Numeric	Aritmetic	Numeric	Numeric
Numeric	De comparare	Numeric	Logic
Logic	Logic	Logic	Logic

Operatorii logici ne permit scrierea de expresii logice mai complexe pe care să le folosim atât la instrucțiunile de decizie cât și la cele repetitive, de regulă acolo unde este nevoie de o condiție.

Sunt trei operatori logici:

`!` – operator unar, numit *not*

`&&` - operator binar, numit *și*, *and*

`||` - operator binar, numit *sau*, *or*

Toți au ca operanzi expresii care sunt interpretate drept logice.

Modul de aplicare a operatorului not este următorul:

`! adevărat`	`fals`
`! fals`	`adevărat`

Exemple:

`!1`	`0`
`!0`	`1`
`!2`	`0` (se interpreteaza ca “not adevărat”)
`!(1+2)`	`0`

Observăm că operandul la care se aplică not poate fi orice valoare dar rezultatul aplicării sale este `0` sau `1`.

Operatorul “not”, fiind unar are prioritate mare, iar ceilalți doi operatori, ca regulă generală, au prioritate mai mică decât operatorii relaționali (deci mai mică și decât a operatorilor aritmetici).

Iată tabelul cu modul de aplicare a operatorului `&&`

Operand	Operator	Operand	Rezultat
`fals`	`&&`	`fals`	`fals`
`fals`	`&&`	`adevărat`	`fals`
`adevărat`	`&&`	`fals`	`fals`
`adevărat`	`&&`	`adevărat`	`adevărat`

Observăm așadar că rezultatul este adevărat doar dacă ambii operanzi au rezultat logic adevărat. Operatorul este utilizat când dorim ca o condiție să fie adevărată doar dacă mai multe condiții sunt simultan adevărate.

Și operatorii binari logici se aplică de la stânga la dreapta într-o expresie cu mai mulți operatori de aceeași prioritate.

Exemple

`1 && 2`	`1`
`1+2 && 0`	`0`
`1 <= 2 && 2 <= 3`	`1`
`4>3>2`	`0` Aici ar părea că se testează dacă trei valori sunt în ordine descrescătoare. Nu este chiar așa întrucât este vorba despre doi operatori de aceeași prioritate, care se aplică de la stânga a dreapta, se calculează mai întâi `4>3`, rezultatul este `1` iar apoi `1>2`, cu rezultat `0`.
`4>3 && 3>2`	`1` Se evaluează `4>3` (cu rezultat `1`) apoi `3>2` (tot cu rezultat `1`) apoi `1 && 1` cu rezultat adevărat.

Iată tabelul cu modul de aplicare a operatorului `||`

Operand	Operator	Operand	Rezultat
`fals`	`||`	`fals`	`fals`
`fals`	`||`	`adevărat`	`adevărat`
`adevărat`	`||`	`fals`	`adevărat`
`adevărat`	`||`	`adevărat`	`adevărat`

Observăm așadar că rezultatul este fals doar dacă ambii operanzi au rezultat logic fals. Operatorul este utilizat când dorim ca o condiție să fie adevărată dacă este adevărată cel puțin una dintre mai multe alte condiții.

Exemple

`1 || 2`	`1`
`1+2 || 0`	`1`
`1-1 || !2`	`0`
`1 <= 2 || 2 <= 3`	`1`

Iată câteva rezultate importante:

`a%2 == 0`	Aceasta este o expresie adevătată *dacă și numai dacă* variabila `a` memorează un număr par. **O expresie spunem că este adevărată dacă și numai dacă se îndeplinește o anumită condiție atunci când expresia este adevărată când condiția este îndeplinită și totodată expresia este falsă când condiția nu este îndeplinită.**
`a%2`	Expresie adevărată dacă și numai dacă variabila `a` memorează un număr impar.
`a%2 != 0`	Expresie echivalentă cu cea de mai sus.
`(a % 2 == 0 && b % 2 == 0) || (a % 2 != 0 && b % 2 != 0)`	Cele trei expresii sunt adevărate dacă și numai dacă ambele variabile memorează numere de aceeași paritate. Facem aici și observația că dacă a este negativ atunci `a%2` are ca rezultate `-1`. Deci în aceste cazuri trebuie grijă la expresia din mijloc.
`a % 2 == b % 2`	
`(a + b) % 2 == 0`	

`a > 0`	Expresie adevărată dacă și numai dacă variabila `a` memorează un număr strict pozitiv.
`a * b > 0`	Expresie adevărată dacă și numai dacă cele două variabile memorează numere nenule cu același semn.
`a == (int)sqrt(a) * (int)sqrt(a)`	Expresie adevărată dacă și numai dacă variabila `a` memorează un număr pozitiv pătrat perfect.
`sqrt(a) == (int)sqrt(a)`	Acesta este alt mod de a testa dacă un număr este pătrat perfect.
`a/10 == 0`	Expresie adevărată dacă și numai dacă variabila `a` memorează un număr întreg de o singură cifră
`(a == b) && (a == c)`	Expresie adevărată dacă și numai dacă cele trei variabile memorează valori egale. Așa cum am subliniat și la unele exemple anterioare, acesta este modul prin care testăm dacă sunt egale valorile din trei variabile, varianta `a==b==c` fiind greșită.
`(x >= a) && (x <= b)`	Expresie adevărată dacă și numai dacă variabila `x` aparține intervalului `[a, b]`. Așa cum spuneam mai sus, varianta `a <= x <= b` este greșită pentru acest test.
`!((x >= a) && (x <= b))`	Aceste două expresii sunt echivalente și adevărate dacă și numai dacă variabila `x` este în afara intervalului `[a, b]`.
`x < a || x > b`	

De avut în vedere următoarele rezultate logice:

`not (P and Q)`	Echivalent cu	`(not P) or (not Q)`
`not (P or Q)`	Echivalent cu	`(not P) and (not Q)`

Astfel, putem arăta echivalența ultimelor două formule:

```
!((x >= a) && (x <= b))
```

=>

```
!(x >= a) || !(x <= b)
```

=>

```
x < a || x > b
```

Este bine de cunoscut aceste rezultate întrucât sunt multe situații în care este mai simplu de pus condiția opusă celei dorite și apoi negarea ei.

Instrucțiunea de decizie if (numită și instrucțiunea de selecție)

Instrucțiunile studiate până acum sunt: citirea, scrierea, atribuirea. Am văzut că scrierea lor se face în funcția `main` și că ele se execută în ordinea în care apar, acest lucru fiind foarte important ținând cont că pot apărea modificări ale valorilor variabilelor după executarea fiecărei instrucțiuni.

Instrucțiunea `if` ne permite să alegem la un moment dat o singură cale de continuare din două posibile. Putem deci, ca la un moment dat să decidem dacă executăm un set de instrucțiuni sau pe altul.

Sintaxa

```
if (expresie_logică)
    Instrucțiune1
else
    Instrucțiune2
```

`expresie_logică` trebuie să fie orice expresie din limbaj care poate să fie evaluată la adevărat sau fals, pentru că instrucțiunea `if` o va trata din acest punct de vedere. Astfel, din ce am discutat mai sus, deducem că expresia poate fi de orice tip numeric (dacă se evaluează la `0` se consideră falsă iar dacă se evaluează la o valoare nenulă se consideră adevărată). De multe ori folosim termenul *condiție* atunci când ne referim la expresia logică a unei instrucțiuni `if`.

Expresia logică trebuie scrisă toată între paranteze rotunde; `if` respectiv `else` reprezintă cuvinte cheie ale limbajului recunoscute automat de compilator.

Cele două instrucțiuni care apar pot fi oricare ale limbajului (deci și dintre cele învățate deja, alt `if`, sau structuri repetitive, care vor fi ulterior studiate).

Dar ce este cel mai important: fiecare dintre cele două instrucțiuni (ramuri) poate fi instrucțiunea bloc (mai multe alte instrucțiuni plasate între acolade).

Ceea ce tocmai am descris mai sus poate fi perceput și altfel: Dacă pe o ramură (dintre cele două) avem nevoie de o singură instrucțiune o putem indica direct, dacă însă programatorul decide că are nevoie de cel puțin două, atunci va trebui să le plaseze între acolade.

Modul de executare

Primul lucru care se face la întâlnirea unei astfel de instrucțiuni este evaluarea expresiei logice. Dacă aceasta are valoarea adevărat se va executa numai instrucțiunea 1, iar dacă expresia are valoarea fals se va executa numai instrucțiunea 2. Apoi, se trece la ce urmează în program după instrucțiunea if. Așadar, acesta este mecanismul de care dispunem în limbaj pentru a hotărî ca la un moment dat să urmăm o cale sau alta. Mai facem des referire la *ramurile* instrucțiunii if, asta însemnând, evident, ramura cu instrucțiunea 1, corespunzătoare cazului când condiția este adevărată respectiv ramura corespunzătoare instrucțiunii 2, pentru cazul în care instrucțiunea este falsă.

Un exemplu des întâlnit în practică de folosire a deciziei este următorul: dorim să părăsim un program și la alegerea opțiunii de închidere suntem întrebați dacă dorim cu adevărat să ieșim sau nu. În funcție de alegerea noastră, programul se va comporta într-un fel sau altul: dacă alegem că dorim chiar să ieșim se va executa codul care face închiderea programului, iar dacă nu dorim se va executa în continuare cod cu programul pornit.

Alt lucru foarte important este cel legat de indentarea (alinierea codului). Aici este vorba de instrucțiuni care se sebordonează altora. Astfel, cele două ramuri (instrucțiune 1 și instrucțiune 2) se subordonează lui `if`. Ele trebuie scrise un tab la dreapta decât linia care conține `if` și cea care conține `else`. Iată trei exemple, toate corecte din punct de vedere sintactic, însă doar ultima respectă regula de indentare.

```
if (a>b) cout<<a;
else cout<<b;
```

```
if (a>b)
cout<<a;
else
cout<<b;
```

```
if (a>b)
    cout<<a;
    else
        cout<<b;
```

```
if (a>b)
    cout<<a;
else
    cout<<b;
```

Avem în continuare un prim exemplu de folosire a instrucțiunii `if` prin două programe echivalente, ambele afișând maximul a două valori întregi introduse de la tastatură.

```
#include <iostream>
using namespace std;
```

```
#include <iostream>
using namespace std;
```

```
int a, b;
int main () {
    cin>>a>>b;
    if (a>b)
        cout<<a;
    else
        cout<<b;
}
```

```
int a, b, c;
int main () {
    cin>>a>>b;
    if (a>b)
        c = a;
    else
        c = b;
    cout<<c;
}
```

Soluția din partea stângă folosește o variabilă mai puțin întrucât se folosește pe ramurile lui `if` direct instrucțiunea de afișare. În partea dreaptă doar se notează (prin atribuire) în variabila `c` valoarea maximă iar tipărirea ei se realizează după `if`. Remarcați indentarea!

Forma precurtată a instrucțiunii if (fără ramura else).

Sunt deseori cazuri când la îndeplinirea unei condiții este nevoie să executăm un anume cod, dar dacă nu se îndeplinește condiția trebuie doar să trecem mai departe fără să executăm ceva. Limbajul de programare pune la dispoziție o instrucțiune pentru a realiza acest lucru: varianta fără ramura `else` a instrucțiunii `if`.

Sintaxa:

```
if (expresie_logică)
    Instrucțiune
```

Modul de executare

Se evaluează expresia logică. Dacă aceasta are valoarea true se va executa instrucțiunea iar dacă are valoarea false se va trece la ce urmează după `if` fără să se execute ceva.

Iată încă un program care afișează maximul a două numere introduse de la tastatură, dar care folosește doar această formă a instrucțiunii `if`.

```
#include <iostream>
using namespace std;
int a, b, maxim;
int main () {
    cin>>a>>b;
    maxim = a;
    if (maxim < b)
        maxim = b;
    cout<<maxim;
}
```

Să analizăm ce se întâmplă dacă introducem în ordine, ca date de intrare, valorile `3` și `12`. Mai întâi, în variabila maxim se copiază valoarea `3`. Urmează apoi o instrucțiune `if` care nu conține ramura `else`. Se evaluează expresia sa logică și are valoarea true (`maxim(3) < b(12)`), așa că se va executa instrucțiunea, deci variabila `maxim` se va modifica, prin atribuire, ajungând să memoreze valoarea `12`. Instrucțiunea `cout<<maxim;` este după `if`, așa că se va tipări `12`.
Dacă numerele ar fi fost introduse invers, adică întâi `12` și apoi `3`, în urma instrucțiunii `maxim = a;` se copiază în variabila `maxim` valoarea `12`, apoi, condiția de la `if` fiind falsă și neexistând ramura `else`, se trece direct după `if`, adică la afișare. Deci și în acest caz se tipărește `12`.
Să analizăm câteva secvențe de cod (variabilele care apar sunt de tip `int`).

`if (a<=b<=c)` `  cout<<"Crescator";`	Este corect din punct de vedere sintactic dar, cum am mai spus în alte exemple mai sus, modul de a scrie condiția nu este cel corect pentru a testa dacă avem trei valori în ordine crescătoare.
`if (a<=b && b<=c)` `  cout<<"Crescator";`	Se afișează mesajul dacă valorile celor trei variabile sunt în ordine crescătoare. În caz contrar nu se afișează nimic.

`if (1)`
`  cout<<"da";`
`else`
`  cout<<"nu";`
	Este o secvență corectă dar care are ca efect afișarea întotdeauna a mesajului "da". Asta pentru că expresia logică fiind constantă și egală cu `1` va avea mereu valoarea adevărat, deci se va executa instrucțiunea de pe prima ramură.

`if (nota == 10)`
`  cout<<"premiant";`
`else`
`  cout<<"nu";`
	O secvență care afișează un mesaj dacă variabila nota are valoarea `10` și alt mesaj în caz contrar.

`if (nota = 10)`
`  cout<<"premiant";`
`else`
`  cout<<"nu";`
	Singura modificare este scrierea în condiție a unui singur caracter `=`, ceea ce schimbă fundamental lucrurile. Acum avem o instrucțiune de atribuire care pe lângă faptul că modifică valoarea variabilei nota, va avea ca rezultat și valoarea atribuită, adică `10`. Așadar condiția este mereu adevărată și întotdeauna se afișează primul mesaj.

`cin>>a>>b;`
`if (a > b) {`
`  maxim = a;`
`  x = a-b;`
`} else {`
`  maxim = b;`
`  x = b-a;`
`}`
`cout<<maxim<<" "<<x;`
	Secvența afișează maximul celor două valori introduse de la tastatură și valoarea absolută a diferenței lor. Observați că pe ambele ramuri ale instrucțiunii `if` avem câte două instrucțiuni. În ambele cazuri am pus acoladele, grupându-le ca instrucțiuni bloc.
	Remarcați modul de indentare, instrucțiunile subordonate sunt scrise cu un tab mai la dreapta iar acoladele sunt plasate astfel: cea deschisă pe aceeași linie cu cea căreia i se subordonează blocul iar cea închisă în dreptul coloanei de început a instrucțiunii principale.

`if (a > b) {`
`  cout<<a;`
`else`
`  cout<<b;`
`}`
	Aici avem eroare de sintaxă. Acoladele trebuie să grupeze instrucțiunile aceleiași ramuri, nu ca în acest caz unde acolada deschisă începe pe o ramură iar cea de închidere apare pe cealaltă. Compilatorul consideră că avem o instrucțiune `if` cu o singură ramură, pe care este instrucțiunea bloc, iar în interior, `else` nu are cu cine să se asocieze (`if`).

`if (a > b)`
`  x = 1;`
`  y = 2;`
`else`
`  z = 3;`
	Un alt caz tipic de eroare de sintaxă. Imediat după linia cu `if`, neavând acolade se consideră că `x=1` este pe post de instrucțiune1. Apoi, neurmând imediat `else`, se consideră că `if` s-a terminat (ar fi vorba de un `if`, forma prescurtată). Instrucțiunea `y=2;` ar fi în continuare, independentă de `if` iar `else` nu are un `if` anterior cu care să se asocieze. Am fi putut pune între acolade primele două atribuiri și atunci totul ar fi în regulă din punct de vedere sintactic. În acest caz am avea un `if` în care pe o ramură avem acolade și pe alta nu și evident că nimic nu interzice acest lucru.

`if (a <= x) && (x <= b)`
`  cout<<"inauntru";`
`else`
`  cout<<"afara";`
	Codul alăturat, care dorește să testeze dacă `x` aparține intervalului `[a, b]` conține o eroare de sintaxă: expresia logică nu este toată plasată între peranteze rotunde. Corect ar fi:
	`if ((a <= x) && (x <= b)) …`
	Pornind de la acest exemplu, ne-am putea gândi că alt mod de a pune condiția ca `x` să fie în afara intervalului ar fi:
	`if !((a <= x) && (x <= b)) …`
	adică să scriem negarea condiției de mai sus. Am scris aproape bine, fiind necesară și adăugarea întregii expresii între paranteze rotunde. Corect:
	`if (!((a <= x) && (x <= b))) …`

Probleme rezolvate

1. Instrucțiunea `if` rezolvă problema deciziei când avem două variante de a continua.

În exemplul următor arătăm cum folosim instrucțiunea `if` pentru a rezolva cazul cu trei variante de continuare.

```
#include <iostream>
using namespace std;
int punctaj;
int main () {
    cin>>punctaj;
    if (punctaj >= 90)
        cout<<"bine";
    else
        if (punctaj >= 40)
            cout<<"mediu";
        else
            cout<<"slab";
}
```

```
#include <iostream>
using namespace std;
int punctaj;
int main () {
    cin>>punctaj;
    if (punctaj >= 40)
        if (punctaj >= 90)
            cout<<"bine";
        else
            cout<<"mediu";
    else
        cout<<"slab";
}
```

Cele două programe sunt echivalente și rezolvă corect următoarea problemă: Se citește de la tastatură un număr natural (considerăm că ne bazăm că este cuprins între `0` și `100`) care reprezintă punctajul obținut de un concurent și afișează mesajul *bine* dacă s-au obținut cel puțin `90` de puncte, mesajul *mediu* în caz contrar dar dacă sunt totuși cel puțin `40` de puncte și mesajul *slab* în cazul că nu s-au obținut nici `40` de puncte.
Varianta din stânga are un `if` ce tratează pe prima ramură unul dintre cazuri iar pe a doua ramură tratează celelalte două cazuri. Așadar, pe această ramură fiind iarăși două variante posibile, este necesar un alt `if`.
Varianta din dreapta are de asemenea un `if` direct în funcția `main`, dar pe post de instrucțiune1 a acestuia nu este direct o afișare ca la prima variantă ci un alt `if`. Astfel, această ramură tratează două dintre cele trei cazuri, iar al treilea caz rămâne în sarcina `else` a instrucțiunii `if` principale.
Concluzia este că atunci când sunt trei variante de continuare se folosesc două `if`-uri unul în altul (în stânga al doilea `if` a fost utilizat pe ramura `else` a primului iar în dreapta al doilea `if` a fost utilizat pe prima ramură a primului).

2. Se citește un număr. Să se verifice dacă este format dintr-o singură cifră, afișându-se un mesaj corespunzător.

Următoarea este o soluție corectă a acestei probleme:

```
#include <iostream>
using namespace std;
int x;
int main () {
    cin>>x;
    if (x < 10)
        cout<<"Da";
    if (x >= 10)
        cout<<"Nu";
}
```

Totuși, varianta care se folosește este aceasta:

```
#include <iostream>
using namespace std;
int x;
int main () {
    cin>>x;
```

```
    if (x < 10)
        cout<<"Da";
    else
        cout<<"Nu";
}
```

Am punctat acest exemplu întrucât sunt multe cazuri când la început există tendința de a scrie câte un `if` simplu pentru fiecare condiție chiar și în cazul în care (ca acum) cele două cazuri sunt complementare și pot fi tratate cu un `if` ce are ambele ramuri.

3. Se citesc `3` numere întregi `a`, `b`, `x`. Știindu-se că `a` și `b` sunt în relația `a <= b`, să se verifice dacă `x` aparține intervalului `[a,b]`, afișând un mesaj corespunzător (*da*/*nu*).

Putem studia problema astfel: Avem de analizat trei cazuri, matematic: `x < a` (se afișează nu), `a<=x<=b` (se afișează *da*) respective `x > b` (se afișează *nu*). Am văzut mai sus că putem rezolva asta cu două instrucțiuni `if` subordonate una alteia.

```
#include <iostream>
using namespace std;
int a, b, x;
int main () {
    cin>>a>>b>>x;
    if (x < a)
        cout<<"nu";
    else
        if (x <= b)
            cout<<"da";
        else
            cout<<"nu";
}
```

Celălalt mod în care putem face analiza este următorul: observăm că sunt două variante de dată de ieșire: *da* și *nu*. Ne gândim deci dacă nu cumva putem scrie un singur `if`, cu o condiție corespunzătoare. Acest lucru este, evident, posibil și găsiți rezolvarea la un paragraf anterior.

4. Se citește de la tastatură un număr natural. Să se verifice dacă el reprezintă o valoare cu exact două cifre, în caz afirmativ afișându-se mesajul *Da*, în caz contrat mesajul *Nu*.

```
#include <iostream>
using namespace std;
int x;
int main () {
   cin>>x;
   if (x >= 10 && x < 100)
       cout<<"Da";
   else
       cout<<"Nu";
}
```

```
#include <iostream>
using namespace std;
int x;
int main () {
    cin>>x;
    if (x / 10 != 0 && x / 100 ==
0)
        cout<<"Da";
    else
        cout<<"Nu";
}
```

Analizați, în cele două moduri echivalente de a rezolva problema, variantele propuse pentru a testa dacă numărul are exact două cifre.

5. Se citesc de la tastatură trei numere naturale. Afișați valorile lor în ordine crescătoare.

```
#include <iostream>
using namespace std;
```

```
#include <iostream>
using namespace std;
```

```
int a, b, c;
int main () {
    cin>>a>>b>>c;
    if (a <= b && b <= c)
        cout<<a<<" "<<b<<" "<<c;
    if (a <= c && c <= b)
        cout<<a<<" "<<c<<" "<<b;
    if (b <= a && a <= c)
        cout<<b<<" "<<a<<" "<<c;
    if (b <= c && c <= a)
        cout<<b<<" "<<c<<" "<<a;
    if (c <= a && a <= b)
        cout<<c<<" "<<a<<" "<<b;
    if (c <= b && b <= a)
        cout<<c<<" "<<b<<" "<<a;
}
```

```
int a, b, c, aux;
int main () {
    cin>>a>>b>>c;
    if (a > b) {
        aux = a;
        a = b;
        b = aux;
    }
    if (b > c) {
        aux = b;
        b = c;
        c = aux;
    }
    if (a > b) {
        aux = a;
        a = b;
        b = aux;
    }
    cout<<a<<" "<<b<<" "<<c;
}
```

Varianta din stânga este una brută care analizează toate cele `6` moduri de a așeza cele trei numere. Pe lângă faptul că este nepractică pentru mai multe numere, cere și multă atenție întrucât if-urile nu sunt separate între ele și o mică greșeală la o condiție face fie să nu se afișeze nimic sau chiar să se afișeze mai multe rezultate (dacă ar fi adevărate mai multe condiții). În plus, varianta funcționează bine doar dacă sunt la intrare valori distincte.

Varianta din dreapta este mai elegantă, interschimbând valori ale variabilelor după un algoritm care garantează că la final acestea ajung în ordine crescătoare. Principiul folosit este: se compară `a` cu `b` și dacă este cazul le interschimbăm, asigurându-ne că `a` și `b` ajung în ordine crescătoare. Apoi se compară `b` (acum maximul primelor două) cu `c` și după această etapă în `c` ajunge valoarea cea mai mare. Revenim cu ultimul `if` să ne asigurăm că și primele două valori ajung în ordine crescătoare.

Observați la toate problemele modul de indentare în exemplele de mai sus în care avem mai multe nivele de subordonare a instrucțiunilor.

Operatorul ternar

În limbajul C/C++ majoritatea operatorilor sunt binari dar sunt și câțiva operatori unari. Există și un singur operator care solicită trei operanzi, deci este operator ternar. În sintaxa sa apar ca separatori caracterele `?` și `:`

Sintaxa:

```
(expresie1) ? (expresie2) : (expresie3)
```

Prezența parantezelor rotunde nu face parte din sintaxă dar este recomandată pentru a evita neplăceri ce pot apărea din cauza priorității operatorilor.

Modul de aplicare a sa este următorul: se evaluează mai întâi `expresie1` din punct de vedere logic, iar dacă rezutatul său este true se va evalua și `expresie2` și acesta va fi rezultatul întregii expresii, iar dacă `expresie1` se evaluase la false se evaluează apoi doar `expresie3` și rezultatul său va fi și rezultatul întregii expresii.

Forțând puțin exprimarea, putem afirma că acest operator este un `if` care are ca efect returnarea unei valori.

Este de regulă folosit pentru a scrie cod mai compact.

```
cin>>a>>b;
cout<<((a > b) ? a : b);
```

Aveți anterior un cod care are ca efect afișarea maximului dintre două valori citite de la tastatură.

Instrucțiunea de selecție cu mai multe ramuri (switch)

Am observat că dacă avem mai mult de două variante de ieșire posibile putem rezolva asta folosind mai multe `if`-uri plasate unul în altul. Există și o altă instrucțiune care poate fi utilă în aceste cazuri și este la alegerea noastră dacă o folosim pe ea sau dacă alegem să scriem mai multe instrucțiuni `if`. Este vorba despre instrucțiunea `switch`.

Sintaxa

```
switch (expresie) {
    case expresie_constanta1:
        instructiuni1
        break; /// optional
    case expresie_constanta2:
        instructiuni2
        break; /// optional

/// putem plasa aici oricate ramuri case ca si cele de mai sus

    default:
        instructiuniD
        break; /// optional
}
```

Expresiile prezente trebuie să fie de tipuri ***întregi***. În caz contrar apare eroare de compilare.
Instrucțiunile `break;` sunt opționale.
Expresia din antetul instrucțiunii (numită și expresie selector) se scrie între paranteze rotunde. Celelalte expresii, așa cum le este scris și numele trebuie să poată fi evaluate de la compilare, așadar trebuie să conțină doar date constante.
Ramurile se introduc prin cuvintele cheie `case`.
După fiecare expresie constată ce introduce o ramură se scrie caracterul `:` (două puncte).
Instrucțiunile de pe fiecare ramură în parte nu trebuie grupate neapărat între acolade.

Mod de executare

Se evaluează mai întâi expresia din antet (selectorul). Dacă se găsește vreo expresie constantă egală cu valoarea ei, încep să se execute instrucțiunile de după acea expresie constantă. **Este foarte important de remarcat că dacă lipsește `break`, se trece apoi la instrucțiunile de pe ramurile ce urmează.** Când se ajunge la acolada de final a instrucțiunii `switch` sau la întâlnirea lui `break; switch` se termină. Dacă valoarea expresiei evaluate nu este egală cu a niciuneia dintre expresiile constante se execută instrucțiunile de după eticheta `default`, dacă aceasta există. Putem așadar privi un `swhitch` ca o decizie cu mai multe ramuri, în care ramura `default` este un fel de `else`.

Iată câteva exemple pe care le analizăm în primul rând din punct de vedere sintactic.

`double x;`
`cin>>x;`
`switch(x) {`
`  case 1:`
`  cout<<1;`
`  break;`
`  case 2:`
`  cout<<2;`
`  break;`
`  default:`
`  cout<<"Altceva";`
`}`
	Eroare de sintaxă. Variabila `x` este de tip real și este folosită pe post de expresie selector. Acolo trebuie să fie o valoare de tip întreg.

`int x;`
`cin>>x;`
`switch(x) {`
`  case 1:`
`  cout<<1;`
`  break;`
`  case 2`
`  cout<<2;`
`  break;`
`  default:`
`  cout<<"Altceva";`
`}`
	Eroare de compilare, nu am scris caracterul *două puncte* după a doua expresie constantă.

`int x;`
`cin>>x;`
`switch x {`
`  case 1:`
`  cout<<1;`
`  break;`
`  case 2:`
`  cout<<2;`
`  break;`
`  default:`
`  cout<<"Altceva";`
`}`
	Eroare de compilare. Expresia selector trebuie plasată între paranteze rotunde.

`int x, y;`
`cin>>x;`
`switch x {`
`  case 1+y:`
`  cout<<1;`
`  break;`
`  case 2:`
`  cout<<2;`
`  break;`
`  default:`
`  cout<<"Altceva";`
`}`
	Eroare de compilare. `1+y` nu este o expresie constantă întrucât intervine acolo o variabilă. Dacă ar fi fost, spre exemplu `1+7`, totul era corect din punct de vedere sintactic.

Probleme rezolvate.

1. Fie următoarea secvență de program:

```
int x;
cin>>x;
switch (x) {
   case 1:
       cout<<"Luni";
       break;
   case 2:
       cout<<"Marti";
       break;
   case 3:
       cout<<"Miercuri";
       break;
   case 4:
       cout<<"Joi";
       break;
   case 5:
       cout<<"Vineri";
       break;
   default:
       cout<<"Nu este zi lucratoare";
}
```

Efectul este următorul: se cere un număr și dacă se introduce una dintre valorile `1`, `2`, `3`, `4`, `5` se va tipări numele zilei corespunzătoare din săptămână. În caz contrar se va tipări un mesaj.

2. Mai jos este același program cu următoarea modificare: de la ziua miercuri s-a eliminat `break;`

```
int x;
cin>>x;
switch (x) {
   case 1:
       cout<<"Luni";
       break;
   case 2:
       cout<<"Marti";
       break;
   case 3:
       cout<<"Miercuri";
   case 4:
       cout<<"Joi";
       break;
   case 5:
       cout<<"Vineri";
       break;
   default:
       cout<<"Nu este zi lucratoare";
}
```

Efectul este aproape același cu al programului de la problema `1`. Diferența este că dacă se introduce de la tastatură valoarea `3`, se va tipări `MiercuriJoi` (se evaluează la valoarea `3` selectorul, se sare la ramura cu expresia constantă `3`, se execută instrucțiunea de acolo – adică se afișează `Miercuri`, dar în lipsa lui `break`

se trece la instrucțiunile de la următoarea ramură, în acest caz mai afișându-se și `Joi`, apoi `switch` se termină pentru că se întâlnește `break`).

3. Nu este obligatoriu ca ramura `default` să fie scrisă ultima. Să analizăm codul:

```
int x;
cin>>x;
switch (x) {
   case 1:
       cout<<"Luni";
       break;
   case 2:
       cout<<"Marti";
       break;
   case 3:
       cout<<"Miercuri";
   default:
       cout<<"Nu este zi lucratoare";

   case 4:
       cout<<"Joi";
       break;
   case 5:
       cout<<"Vineri";
       break;
}
```

Dacă se dă de la intrare `9`, se va sări la eticheta `default`, se va afișa mesajul `Nu este zi lucratoare` apoi, în lipsă de `break` se va mai scrie și `Joi`.

Așa cum ne-am mai exprimat, `break` este o instrucțiune, una mai specială care în afară de contextul de mai sus se folosește în structurile repetitive. Vom analiza asta la momentul potrivit.

Tema

Întrebări și exerciții

1. Indicați operatorii relaționali. Cum se plasează, ca prioritate, față de cei aritmetici?
2. Indicați operatorii logici. Cum se plasează, ca prioritate, față de cei aritmetici și cei relaționali?
3. Care este sintaxa instrucțiunii `if` ? Descrieți modul de executare a acestei instrucțiuni.
4. Care este sintaxa cu care utilizăm operatorul ternar? Identificați asemănări și deosebiri între el și instrucțiunea `if`.
5. Care este sintaxa folosită la instrucțiunea `switch` ?
6. Descrieți modul de funcționale a acestei instrucțiuni.
7. Fie codul:

```
int x;
cin>>x;
switch (x) {
case 1:
    cout<<"Ianuarie\n";
    break;
case 2:
    cout<<"Februarie\n";
    break;
case 3:
```

```
    cout<<"Martie\n";
    break;
case 4:
    cout<<"Aprilie\n";
    break;
case 5:
    cout<<"Mai\n";
    break;
case 6:
    cout<<"Iunie\n";
    break;
default:
    cout<<"Al doilea semestru sau luna
incorecta";
}
```

a) Ce se afișează dacă se citește `3`
b) Ce se afișează dacă se citește `8`
c) Ce se afișează dacă se citește `-1`
d) Scrieți o secvență echivalentă în care înlocuiți instrucțiunea `switch` cu instrucțiuni `if`.

8. Considerăm următoarea secvență de cod. Indicați ce se afișează pentru fiecare set de date de intrare (luați în calcul că nu s-a urmărit indentarea corectă).

```
int x;
cin>>x;
if (x > 10)
   if (x < 100)
       cout<<1;
else
   cout<<2;
```

a) 1
b) 10
c) 20
d) 99
e) 100
f) 200

9. Considerăm următoarea secvență de cod. Indicați ce se afișează pentru fiecare set de date de intrare (luați în calcul că nu s-a urmărit indentarea corectă).

```
int x;
cin>>x;
if (x <= 100)
    cout<<1;
else
    if (x > 10)
        cout<<2;
```

a) 1
b) 10
c) 20

d) 99
e) 100
f) 200

10. Considerăm următoarea secvență de cod. Indicați ce se afișează pentru fiecare set de date de intrare (luați în calcul că nu s-a urmărit indentarea corectă).

```
int x;
cin>>x;
if (x > 10)
   if (x < 100)
       cout<<1;
   else
       cout<<2;
else
    cout<<3;
```

a) 1
b) 10
c) 20
d) 99
e) 100
f) 200

Probleme

1. Se citește de la tastatură un număr natural de maxim 2 cifre. Să se afișeze pe ecran valori astfel: dacă numărul este mai mic sau egal cu 15 se va afișa pătratul valorii sale; dacă numărul este cuprins între 16 și 30 (inclusiv) se va afișa suma cifrelor sale; în caz contrar se va afișa produsul cifrelor sale. (pbinfo, #451)
2. Se citește de la tastatură un număr natural de maxim 3 cifre. Să se determine câte cifre are (pbinfo, #449).
3. Se citește de la tastatură un număr natural de 3 cifre. Să se determine câte cifre impare conține (pbinfo, #452).
4. Se citește de la tastatură un număr natural de 3 cifre, distincte. Să se afișeze pe ecran cifra din mijloc, ca valoare (pbinfo, #447).
5. Se dau 5 numere distincte. Să se determine suma celor mai mari 3 dintre ele (pbinfo, #559).
6. Se citește de la tastatură un număr natural ce reprezintă o lună a anului (dacă citim 3 ne referim la a treia lună, deci Martie). Programul va afișa (câte una pe rând) lunile rămase în acel an după cea citită. Se garantează că se introduce de la intrare o valoare cuprinsă între 1 și 11. Se recomandă utilizarea instrucțiunii switch. (pbinfo, #2606)

Exemplu:

Date de intrare	Date de ieșire
9	Octombrie Noiembrie Decembrie