**LUCRAREA DE LABORATOR nr. 2**

*Tema****:* Programarea problemelor simple în baza algoritmilor cu structuri ramificate şi ciclice.**

***Scrierea programelor cu cod perfect necesită gândire, gust şi răbdare!***

*Obiectivele temei*

1. Modul de realizare a dialogului utilizatorului în SO şi mediului integrat fie Turbo C/C++, fie Borland C/C++ .

2. Acumularea informaţiilor suplimentare referitoare la operaţiile şi tehnicile de prelucrare a schemelor logice în Word.

3. Familiarizarea cu tehnicile de lucru în mediul integrat fie Turbo C/C++, fie Borland C/C++ ..

4. Aprofundarea cunoştinţelor în domeniul algoritmizării proceselor ramificate şi ciclice cu programarea în C, evidenţiind specificul implementării instrucţiunilor if şi case (vezi Anexa îndrumarului).

5. Perfecţionarea tehnicilor de derulare, verificare şi testare a programelor în C.

*Subiectele temei şi ordinea executării*

1. Însuşirea instrucţiunilor şi comenzilor SO pentru prelucrarea fişierelor de sistem şi directorii.

2. Operarea cu textele pentru diverse perfectări în mediul Word.

3. Însuşirea principiilor efective de lucru în mediul integrat C: submeniul FILE (încărcarea, înmagazinarea, extragerea); Edit**,** submeniul COMPILE (compilarea programelor); RUN (execuţia programelor); OPTIONS (definirea şi stabilirea parametrilor a regimului de lucru).

4. Elaborarea algoritmilor şi însuşirea modalităţilor de transpunere algoritmilor în limbajului C pentru realizarea următoarelor:

* însuşirea procesului de declarare a constantelor şi variabilelor în C;
* procesul de citire, de afişare şi de calcul al algoritmului cu structură ramificată;
* organizarea transferului condiţionat şi necondiţionat în C;
* elaborarea algoritmilor cu structură ciclică simplă şi încorporată cu un număr predifinit de repetare şi verificare, utilizînd numai instrucţiunile if şi switch.

5. Transpunerea algoritmilor şi programelor TP, obţinute în LUCRAREA DE LABORATOR nr.1,în limbajului C.

6. Derularea, testarea programului şi verificarea soluţiei problemelor trasate.

7. Manipularea cu comenzile submeniurilor mediului integrat C pentru obţinerea depănării eficiente.

*Conţinutul raportului (vezi lucr. de laborator nr.1)*

1. Suplimentar: Enumerarea şi caracterizarea erorilor admise pe parcursul efectuării lucrării.

# **Noţiuni generale**

**Implementarea programului: editare, compilare, editare de legături, execuţie.**

Un mediu de programare C conţine:

* Un editor de text – folosit pentru creearea şi modificarea codului sursă C
* Un compilator – pentru a converti programul C în cod înţeles de calculator.

Procesul de compilare cuprinde:

* o fază de precompilare (macroprocesare) în care are loc expandarea macrodefiniţiilor si compilarea condiţională si includerea fisierelor
* compilarea propriu-zisă în urma careia se generează cod obiect
* Fişiere incluse – care conţin definiţii şi prototipurile unor funcţii folosite de program
* Fişiere biblioteci de funcţii
* Editor de legături – care leagă la codul obiect anumite funcţii de bibliotecă (de exemplu intrări/ieşiri) extrase dintr-o bibliotecă de funcţii, creindu-se un program executabil.

**1.1. Descrierea generală ale compilatorului-translatorului(CT) Borland C/C++**

CT este compus din trei părţi principale :

* 1. Bara de meniuri
  2. Editorul text
  3. Bara de statut

Pentru a înţelege cum funcţionează softul dat , vom examina mai întâi fiecare parte ale CT aparte :

**A. Bara de meniuri** – include în sine aşa meniuri ca :

**File** – În acest meniu sunt incluse funcţiile de operare cu fişierul prelucrat . Adică fişierul în care se conţine sursa programului . Datorită acestui meniu avem posibilitate să salvăm , deschidem sau însuşi să creem un document nou , în care pe parcurs vom înscrrie programul principal.

**Edit** – meniul dat reprezintă în sine funcţiile de prelucrare curentă ale textului programei.

**Search** – datorită meniului dat efectuarea operaţiilor de căutare a cuvintelor cheie(FIND) , deplasarea directă pe linia necesară(GO TO LINE NUMBER ...) şi operaţiei REPLACE CT Borland prezintă un soft uşor utilizabil.

**Run** – în meniul dat sunt include comenzile de complilare translare ale sursei programului . Tot datorită unei comenzi (trace) ale acestui meniu avem posibilitate să efectuăm trasarea programulu. Trasarea programului include în sine operaţia de executarea a programului pe pasuri .

**Compile** – meniul dat prezntă un set de comande ce permit detectarea greşelilor sintactice şi semantice ale programului curent. Aici sunt ataşate aşa comenzi ca „link, build şi compile”.

**Debug** – datorită acestui meniu apar aşa posibilităţi ca ataşarea breack- pointerilor (breackpoint), examinarea valorilor variabilelor pe parcursul executării programului (inspect...) şi a altor operaţii de tipul dat .

**Project** – prin meniul dat putem efectua operaţiile de închidere , deschidere , adăugare sau ştergere unor obiecte ale însuşi proectului prelucrat.

**Options** – meniul dat include în sine opţiunele sistematice ale softului dat.

**Window** – meniul dat prezintă comenzile de prelucrare ale ferestrei de redactare.

**Help** – acest meniu include în sine , informaţia despre producătorul acestui soft şi funcţiile (cu exemple de interpretarea acestora în cadrul elaborării ale programului ) ale limbajului C/C++ .

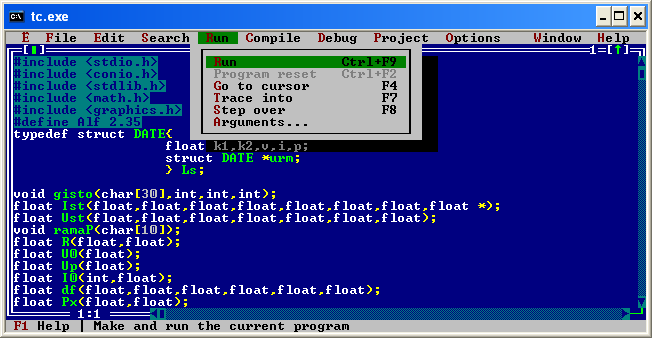
**B. Platforma de editare** include în sine panoul (fereastra) de redactare ale sursei programului .

**Panoul de ediatare** are posibilitate de a schimba dimensinele sale. Datorită acestei operaţii avem posibilitate să examinăm informaţia a panourilor ce se află în spatele panoului prelucrat fără a se adresa la acela însuşi.

Platforma dată mai are o prioritate de afişare ale **panoului de compilare**. Această fereastră ne dă posibilitate după efectuarea operaţiei de compilare şi link-uire ale codului programei , să afişeze greşelile sintactice sau şi semantice ale sursei programului prelucrat.

**C. Bara de Statut** În bara dată se afişează informaţia de despre butoanele fierbinţi ale comenzilor clasice ale compilatorlui dat.

Pentru a simplifica lucrul utilizatorului asupra softului dat , aici apare posibilitatea de a se folosi de combinaţiile de taste pentru efectuarea unor comenzi , fără a se adresa la meniuri corespondente. De exemplu : Pentru a efectua operaţia de compilare tastăm Alt+F9 sau pentru a se adresa la bara de meniuri tastăm butonul F10.

 Fig1. Fereastra principală ale colpilatorului- translatorului Borland C

**2. Limbajului C. Meta-limbajul şi setul de caractere**

Programul în C este format: din *directivele preprocesorulu*, din funcţii, descrieri, şi din blocul de execuţie a funcţiei ce cuprinde în acolade instrucţiuni de citire, afişare şi calcul. Una din funcţiile programului trebuie să aiba numele *main* – funcţia principală. Ca regula funcţile se foloseşte pentru a calcula şi a returna un rezultat, de aceea înaintea numelui funcţiei se indică tipul ei. Cele mai importante reguli ale funcţiei sunt:

* dacă funcţia nu returnează un rezultat, se indică tipul *void*
* corpul funcţiei este un bloc, şi de aceea ea se pune în acolade {...}
* fiecare operator se termină cu separatorul  **;**

Directivele preprocesorulu

*# include* – Include alt cScript fişier (fişiere din bibliotecile header sau alte surse.

Formatul: #include <file\_name> #include “file\_name”

fişiere din bibliotecile header:

* stdio.h- se foloseşte în program pentru a apela funcţiile de intrare ieşire (scanf, printf)
* math.h- se foloseşte în program pentru a apela funcţiile matematice

#include <stdio.h> /\* funcţii standard de I/O; \*/

#include <conio.h> /\* extensie a funcţiilor de I/O la consolă;\*/

#include <stdlib.h> /\* funcţii „standard” ale bibliotecilor „C”; \*/

#include <string.h> /\* funcţii „standard” şiruri de caractere \*/

#include <math.h> /\* funcţii matematice; pentru sin, cos, tan etc.\*/

#include <stdlib.h> /\*exit, etc.\*/

#include <conio.h>/\*getch, etc.\*/

#include <ctype.h>/\*isdigit, etc.\*/

#include <graphics.h > /\*– funcţii de lucru în regim grafic;\*/

#include<time.h>

Parantezele vide *void main****()***indică faptul că funcţia principală nu conţine argumente. Acoladele ***{…}*** joacă rolul lui begin şi end în limbajul Pascal, adică cuprinde corpul funcţiei (programului).

Meta-limbajul care serveşte la descrierea formală a sintaxei limbajului C este simplu.

Categoriile sintactice sau noţiunile care trebuie definite sunt urmate de simbolul '**:**'.

Definiţiile alternative de categorii sunt listate pe linii separate. Dacă o linie nu este suficientă, se trece la linia următoare, aliniată la un tab faţă de linia precedentă.

Un simbol opţional, terminal sau neterminal este indicat prin adăugarea imediată după el a configuraţiei de caractere „opt”.

* 1. **Alfabetul limbajului.**

Conţine setul de caractere ASCII (setul extins 256 caractere: noncifra : a…z A…Z; cifra : 0…9; Delimitatori: simboluri care separa diverse entitati. In C, unele simboluri au dubla semnificatie, de operator si delimitator( in functie de context)).

**2.2. Atomi lexicali.**

Există următoarele entităţi lexicale: identificatori, cuvinte cheie, constante, şiruri de caractere, operatori şi separatori.

*Spaţiile albe* în C sunt reprezentate de: spaţiu liber (blanc), tabulare orizontală, tabulare verticală, linie nouă, pagină nouă, comentarii. Spaţiile albe separă atomii lexicali vecini.

**2.3 Unităţile lexicale ale limbajului C**

În limbajul C există şase tipuri de unităţi lexicale:

* **Identificatori**(succesiune de litere şi cifre dintre care primul caracter este în mod obligatoriu o literă. Se admit şi litere mari şi litere mici dar ele se consideră caractere distincte. Liniuţa de subliniere \_ este considerată ca fiind literă.)
* **Cuvinte-cheie**(identificatori rezervaţi limbajului, ce au o semnificaţie bine determinată şi nu pot fi utilizaţi decât aşa cum cere sintaxa limbajului. Ei au o semnificaţie bine determinată şi nu pot fi utilizaţi decât aşa cum cere sintaxa limbajului. Cuvintele-cheie se scriu obligatoriu cu litere mici.)

*int float double long char short register const auto*

*extern static void unsigned typedef if else*

*switch for while do case default main*

*struct union continue return break sizeof etc.*

* **Constante**-există următoarele tipuri de constante:
* **întreg** (constă dintr-o succesiune de cifre.)

– numere intregi pozitive cu valori intre 0 si 4.294.964.295 si care sunt de treii tipuri :

* zecimale(baza 10) (ex : 43 ; 152 ;7567)
* octale(baza 8) – sisunt precedate de un ″0″ nesemnificativ (ex : 0345­->354(8)
* hexazecimale(baza 16) – sunt precedate de 0x sau 0X (ex : 0X2F4->2F4(16)
* **întreg lung explicit** (O constantă întreagă zecimală, octală sau hexazecimală, urmată imediat de litera l sau L este o constantă lungă. Aceasta va fi generată în calculator pe 4 octeţi)..
* **flotant** (constă dintr-o parte întreagă, un punct zecimal, o parte fracţionară, litera e sau E şi opţional, un exponent care este un întreg cu semn. Partea întreagă şi partea fracţionară sunt constituite din câte o succesiune de cifre. Într-o constantă flotantă, atât partea întreagă cât şi partea fracţionară pot lipsi dar nu ambele; de asemenea poate lipsi punctul zecimal sau litera e şi exponentul, dar nu deodată). Orice valoare reala (ex : 32, 45, 0,5)
* **caracter**(constă dintr-un singur caracter scris între apostrofuri. Participă la operaţiile aritmetice ca şi oricare alte numere). Orice caracter scris intre apostrofuri(ex : ‘A’)
* **simbolic**(un identificator cu valoare de constantă. Valoarea constantei poate fi orice şir de caractere introdus prin construcţia #define).
* **Siruri**(o succesiune de caractere scrise între ghilimele. Ghilimelele nu fac parte din şir; ele servesc numai pentru delimitarea şirului. Caracterul " (ghilimele) poate apărea într-un şir dacă se utilizează secvenţa de evitare \". Cînd un şir apare într-un program C, compilatorul creează un masiv de caractere care conţine caracterele şirului şi plasează automat caracterul NULL ('\0') la sfîrşitul şirului, astfel ca programele care operează asupra şirurilor să poată detecta sfîrşitul acestora. Se admit şi şiruri de lungime zero).

Constantele identificatori se obţin folosind directiva #define a preprocesorului:

#define constantă-identificator literal sau constantă-identificator

#define constantă-identificator (expresie-constantă)

**Tipuri.** Un *tip de date* este precizat prin:

- *o mulţime finită de* ***valori*** corespunzătoare tipului (constantele tipului)

- *o mulţime de* ***operatori*** prin care se prelucrează valorile tipului

- *o multime de* ***restricţii*** de utilizare a operatorilor.

De exemplu tipul întreg (int) este definit prin:

- *mulţimea valorilor* reprezentând numere întregi (între -32768 şi 32767)

- *mulţimea operatorilor* : +, -, \*, /, %

- *mulţimea restricţiilor*: pentru operatorul / împărţitorul nu poate fi 0, etc.

Tipurile pot fi *tipuri fundamentale* şi *tipuri derivate*.

*Tipurile fundamentale* (*predefinite* sau *de bază* sunt:

* Tipul caracter **(char)**
* Tipuri întregi **(int, short, long)**
* Tipuri reale **(float, double, long double)**
* Tipul vid **(void)**
* Tipurile enumerate **(enum)**
* Tipul boolean **(bool)**

Tipurile boolean, caracter, întreg, real şi tipurile enumerate sunt *tipuri aritmetice*, deoarece valorile lor pot fi interpretate ca numere.

Tipurile boolean, caracter, întreg şi enumerările sunt *tipuri întregi*.

*Tipurile derivate* sunt construite pornind de la tipurile fundamentale. Tipurile derivate sunt:

* Tablourile, funcţiile, pointerii, referinţele, structurile (sau înregistrările), uniunile (înregistrările cu variante)

În cele ce urmează, vom înţelege prin *obiect*, o zonă de memorie.

*Declararea unui obiect* specifică numai *proprietăţile obiectului, fără a aloca memorie* pentru acesta.

*Definirea unui obiect* specifică *proprietăţile obiectului şi alocă memorie* pentru obiect.

Declararea obiectelor ne permite referirea la obiecte care vor fi definite ulterior în acelaşi fişier sau în alte fişiere care conţin părţi ale programului.

**Tipuri fundamentale.** Calculatoarele pot lucra în mod direct cu caractere, întregi şi reali. Acestea sunt *tipuri fundamentale* (*predefinite*).

         **intregi**: sunt un sir de cifre zecimale din care prima este diferita de zero. Felul constantei intregi (short, long, hexa) poate fi dat de literele L, l (long), U, u ( unsigned) sau x (hexazecimal):

2000u (16 biti)

2000U (16 biti)

49327 l (32 biti; unsigned)

2000 lu (32 biti; unsigned)

2000 u l (32 biti; unsigned)

0x123 (hexazecimal pe 16 biti)

0x12ab3f (hexazecimal pe 32 biti)

         **flotante**: se reprezinta in dubla precizie. Prezenta literelor f sau l indica simpla respectiv dubla precizie.

123. 123

123.7 123,7

.25 0,25

78e4 78\*104

1.1e-3 1.1\*10-3 a0,0011

         **caracter**: reprezinta codificarea in ASCII a caracterului de memorat, care este plasat intre semnele apostrof: 'a', 'A', '\*'

Exceptii: *i se codifica prin 'ii'*

*' se codifica prin 'i''*

*" se codifica prin 'i"'*

Semnul i este folosit pentru a defini caractere negrafice:

*'it' tabulator*

*'in' rand nou*

*'ir' retur de car*

*'if' salt la pagina noua*

*'ia' activare sunet*

*'ib' revenire cu un spatiu (backspace)*

         **sir de caractere:** o succesiune de mai multe caractere incluse intre *ghilimele*. Caracterele se pastreaza in memorie sub forma de coduri ASCII intr-o zona contigua, ultimul caracter fiind valoarea 0 (NULL), cu rol de marcaj de sfarsit de sir. Exista diferenta intre o constanta care reprezinta *un caracter* si *sirul de caractere* care e format din acelasi caracter.



**'A'** 65 1 octet (A e vazut ca un caracter)



**"A"** 65 00 2 octeti  (A e vazut ca un sir de caractere)

  Pentru caracterul **"** se foloseste in definirea sirului de caractere " i ".

**e) Comentariu** - Este un text introdus de programator pentru a usura intelegerea programului prin scurte explicatii. Textul se introduce intre simbolurile /\* si \*/, pentru a-l deosebi de functiile programului. Un comentariu începe prin **/\*** , se termină prin **\*/** şi se poate întinde pe mai multe linii. Comentariile nu pot fi incluse unele în altele (imbricate). În C++ au fost introduse comentariile pe o singură linie, care încep prin // şi au terminator sfârşitul liniei.

**Terminatorul de instrucţiune.**  Caracterul **;** este folosit ca terminator pentru instrucţiuni şi declaraţii, cu o singură excepţie – după o instrucţiune compusă, terminată prin acoladă, nu mai este necesar terminatorul **;** .

**f) Declararea variabilelor simple -** Orice variabila inainte de a fi utilizata trebuie declarata. Declararea se face printr-o asociere a tipului cu numele variabilei.

*tip nume;*

sau *tip lista\_de\_nume;*

  int i; /\* se declara i ca variabila intreaga \*/

int j,ka0; /\* j si k sunt intregi, k fiind initializat cu 0 \*/

unsigned p; /\* p este un intreg fara semn pe 16 biti \*/

char c; /\* se declara c ca variabila de tip caracter \*/

double a; /\* se declara a ca variabila reala reprezentata pe 64 biti\*/

**g) Declararea tablourilor** - Un tablou este o multime ordonata de variabile de acelasi tip, care pot fi referite prin intermediul indicilor. La declararea unui tablou, compilatorul aloca memorie pentru a pastra valorile elementelor sale. Numele tabloului are valoarea egala cu adresa de inceput a zonei de memorie care i-a fost alocata.

  tip nume\_tablousdimensiunet /\* declararea unui tablou \*/

  /\* declararea unui tablou (sir) cu numele tab, de 5 numere intregi \*/

**int tab[5];**



  tab[5]

/\* declararea unui tablou de 100 de caractere; numele sir\_caract are adresa primului element din sir: sir\_caracts0t \*/

**char sir\_caract[100];**

**float matrice[5][3];**  /\* declararea unei matrici de 5 linii si 3 coloane \*/

**Definiri de tip cu typedef.** Un tip de date poate avea şi un alt nume, prin folosirea declaraţiei typedef. De exemplu: **#typedef int intreg**

Acelaşi efect se obţine cu directiva define: **#define intreg int**

**Tipuri enumerate.** Folosirea tipului enumerare poate creşte claritatea programului, întrucât numele sunt mai semnificative decât valorile care se ascund în spatele lor.

Astfel este mai naturală folosirea valorilor ROSU, ALB, NEGRU, VERDE pentru a desemna nişte culori, decât a valorilor 0, 1, 2, 3.

Constantele simbolice pot fi introduse cu macrodefinitii #define.

#define FALSE 0

#define TRUE 1

Un *tip enumerat* foloseşte în locul valorilor tipului 0,1,2,… nume simbolice:

enum CULORI {ROSU, VERDE, GALBEN, ALBASTRU, NEGRU};

enum boolean {FALSE, TRUE};

Este posibil sa forţăm pentru numele simbolice alte valori întregi decât 0,1,2,…

enum ZILE {LUNI=1, MARTI, MIERC, JOI, VIN, SAMB, DUM};

enum escapes {BELL=’\a’,BACKSPACE=’\b’,TAB=’\t’,NEWLINE=’\n’,

VTAB=’\v’,RETURN=’r’};

Folosind typedef putem defini tipuri enumerative:

typedef enum {ROSU, GALBEN, ALBASTRU} culori;

typedef enum { lu, ma, mi, jo, vi, si, du } zile;

**Tipul vid (void).** Tipul void precizează o mulţime vidă de valori. Acesta se foloseşte pentru a specifica tipul unei funcţii care nu întoarce nici un rezultat, sau a unui pointer generic.

**Definiţii şi declaraţii de variabile,**

O valoare *constantă*  se reprezintă textual (este un *literal*) sau printr-un nume - *constantă simbolică.*

O *variabilă* este un nume (identificator) care desemnează o locaţie de memorie în care se păstrează o valoare.

O variabilă se caracterizează aşadar prin: *nume (adresă)*, *tip* şi *valoare*, atributul valoare putând fi modificat. De exemplu:

**int n, p;**

**char c;**

**float eps;**

O variabilă poate fi *iniţializată* la declararea ei. De exemplu: **float eps=1.0e-6;**

Iniţializarea se face numai o dată, înaintea execuţiei programului.

Variabilele externe şi statice sunt iniţializate implicit la 0.

Pentru o variabilă automatică (declarată în interiorul unui bloc), pentru care există iniţializare explicită, aceasta este realizată la fiecare intrare în blocul care o conţine.

În C++ calificatorul const aplicat unui nume simbolic arată că acesta nu mai poate fi modificat pe parcursul programului şi reprezintă o constantă. Definirea unei constante presupune şi iniţializarea acesteia.

const float pi=3.1415926;

O *definiţie* este o construcţie textuală care asociază unui nume o zonă de memorie (un obiect) şi eventual iniţializează conţinutul zonei cu o valoare corespunzătoare tipului asociat numelui.

**Echivalenţa tipurilor.**  Două tipuri se consideră echivalente în următoarele situaţii:

* echivalenţă structurală a tipurilor
* echivalenţa numelui tipului

Două obiecte sunt de tipuri structural echivalente, dacă au acelaşi tip de componente.

Două obiecte sunt de tipuri echivalente după nume, dacă au fost definite folosind acelaşi nume de tip.

Exemple:

**typedef int integer;**

**int x, y;** /\* x şi y sunt echivalente dupa numele tipului\*/

**integer z**;/\* x şi z sunt de tipuri structural echivalente \*/

* **Variabile** *Scop:* permit includerea datelor de programe

*Definitie:* reprezinta ansamblul adresa fixa, continut variavil din memoria caruia ii este pus in corespondenta numere din program.

Memoria poate fi asimilata unor casute postale. O variabila reprezinta o zona de memorie cu adresa de inceput fixa, dar continut variabil.

Continutul variabilei - se suprascrie o noua valoare peste valoarea veche.

O celula de variabila este egal cu sizeof(tip).

Sizeof este o functie care returneaza lungimea unei variabile sau a unui tip de data.

Pentru a fi folosita in program, o variabila se defineste o singura data.

Obs: Numele variabilei este pus in corespodenta cu adresa de-nceput a variabilei in memorie. Orice regasire, inscriere, citire, se face de la adresa respectiva pe o zona de memorie.

*Operatii tipice:* Variabilele de memorie suporta un set de operatii tipice. Cunoasterea lor este necesara pentru utilizarea corecta a variabilelor. Operatii tipice sunt: atribuirea, citirea de la tastatura, afisarea si utilizarea in axpresii.

Un *operator* este un simbol care arată ce operaţii se execută asupra unor operanzi (termeni).

Un *operand* este o constantă, o variabilă, un nume de funcţie sau o subexpresie a cărei valoare este prelucrată direct de operator sau suportă în prealabil o conversie de tip.

* **Operatori** (pot fi clasificaţi după diverse criterii. Există operatori unari, binari şi ternari, operatori aritmetici, logici, operatori pe biţi etc.). Operatorii sunt împărţiţi în *clase de precedenţă* (sau de *prioritate*). În fiecare clasă de precedenţă este stabilită o *regulă de asociativitate*, care indică ordinea de aplicare a operatorilor din clasa respectivă: de la stânga la dreapta sau de la dreapta la stânga.
* **Separatori**(un caracter sau un şir de caractere care separă unităţile lexicale într-un program scris în C. Separatorul cel mai frecvent este aşa numitul spaţiu alb (blanc) care conţine unul sau mai multe spaţii, tab-uri, new-line-uri sau comentarii.

*Lista separatorilor admişi în limbajul C :*

* **( )** Parantezele mici rotunde  încadrează lista de argumente ale unei funcţii sau delimitează anumite părţi în cadrul expresiilor aritmetice etc.
* **{ }** Acoladele  încadrează instrucţiunile compuse, care constituie corpul unor instrucţiuni sau corpul funcţiilor
* **[ ]** Parantezele mari drepte încadrează dimensiunile de masiv sau indicii elementelor de tablou
* **" "** Ghilimelele  încadrează un şir de caractere
* **' '** Apostrofurile  încadrează un singur caracter sau o secvenţă de evitare
* **;** Punct şi virgula  termină o instrucţiune
* **/\*** Slash asterisc  început de comentariu
* **\*/** Asterisc slash  sfârşit de comentariu.

###### Liniile de control ale compilatorului Caracterul **#** la începutul unei linii indică faptul că aceasta va fi tratată de către preprocesor.

În tabelul de mai jos, operatorii aflaţi pe aceeaşi linie au pondere egală; liniile tabelului sunt în ordinea descrescătoare a ponderilor:

|  |  |
| --- | --- |
| **Operator** | **Modul de evaluare** |
| () [] -> . | de la stânga la dreapta |
| ! ~ ++ -- - (tip) \* & sizeof (operatori unari) | de la dreapta la stânga |
| \* / % | de la stânga la dreapta |
| + - | de la stânga la dreapta |
| << >> | de la stânga la dreapta |
| < <= > >= | de la stânga la dreapta |
| == != | de la stânga la dreapta |
| & | de la stânga la dreapta |
| ^ | de la stânga la dreapta |
| | | de la stânga la dreapta |
| && | de la stânga la dreapta |
| || | de la stânga la dreapta |
| ?: | de la dreapta la stânga |
| = += -= etc. | de la dreapta la stânga |

***O expresie*** este o combinaţie de operanzi, separaţi între ei prin operatori; prin *evaluarea* unei expresii se obţine o *valoare rezultat*.Tipul valorii rezultat depinde de tipul operanzilor şi a operatorilor folosiţi. Evaluarea unei expresii poate avea *efecte laterale*, manifestate prin modificarea valorii unor variabile.

**Instrucţiunile** simple se formează din expresii de asignare, aritmetice, etc., prin adăugarea caracterului **;**, care este terminator de instrucţiune. Prin definiţie, un **bloc de instrucţiuni** (sau o **instrucţiune compusă)** se obţine prin includerea unui grup de instrucţiuni simple între acolade. Are loc următoarea convenţie importantă: **oriunde este permisă apariţia unei instrucţiuni simple, este permisă şi apariţia unui bloc de instrucţiuni.** Astfel, sintaxa instrucţiunilor de control precizează prezenta unei instrucţiuni simple, dar aceasta se poate substitui oricând cu un bloc de instrucţiuni.

Declararea variabilelor se face în felul următor: [tipul variabilei] <nume\_variabilă>;

* Tipurile de bază sunt ***int*, *float*, *double* şi *char****.*

***Tipurile fundamentale*** ale limbajului C/C++ sunt

|  |  |  |
| --- | --- | --- |
| *char* | Reprezentînd tipul *caracter* | pe 1 octet, |
| *int* | întreg | pe 2 octeţi |
| *long* | Întreg | pe 4 octeţi |
| *float*, | însemnînd un număr real | pe 4 octeţi |
| *double*, | ataşat unui număr real | pe 8 octeţi |

Aceste tipuri admit diferite variante, numite tipuri de bază de date.

* Modificatorii de tipuri sunt ***long*, *short*, *unsigned***.

Dacă la declararea variabilelor denumirea tipului lipseşte atunci se consideră implicit tipul *int*.

Cu ajutorul modificatorilor de tipuri şi tipurilor de bază se pot obţine combinaţii:

*Lista diverselor tipuri de date în Turbo-C*

------------------------------------------------------------------------------

* Tipul Dimensiunea(în Biţi) Intervalul de valori
* unsigned char 8 0-255
* char 8 -128-127
* enum 16 -32768-32767
* unsigned short 16 0-65535
* short 16 -32768-32767
* unsigned int 16 0-65535
* int 16 -32768-32767
* unsigned long 32 0-4294967295
* long 32 -2147483648-2147483647
* float 32 3.4E-38-3.4E+38
* double 64 1.7E-308-1.7E+308
* long double 64 1.7E-308-1.7E+308
* pointer \* 16 (near, \_cs, \_ds, \_ss)
* pointer \* 32 (far, huge)

Exemple de declarare a variabilelor:

Turbo Pascal (versiunea 3.x) Turbo C

**var**

**I : Integer; int i;**

**X : Real; float x;**

**Ch : Char; char ch;**

**Line : string[80]; char line[80];**

**2.4. Comenzi şi expresii în limbajul C**

# **Expresie.** Numim o combinaţie validă de operatori, variabile, constante, apeluri de funcţii.

*Operatorii* sunt simboluri care specifică operaţii efectuate asupra unor variabile sau constante. Limbajul C oferă o gamă largă de operatori. Operatorii ce admit doi operanzi se numesc binari, iar cei ce admit un singur operand se numeşte unar.

Sintaxa expresiilor:

<operand> ++ <operand> -- ++ <operand> -- <operand> <operand> = <operand>

<operand> <compound-assignment-ops> <operand>

**Operatorul de atribuire** Operaţia ce apare cel mai des în programare este memorarea unei valori într-o variabilă. Pentru a executa această operaţie, limbajul C foloseşte operatorul de atribuire “**=**”, de forma **id.var=expresie**. Operatorul de atribuire admite operanzi de orice tip scalar şi tipuri definite de utilizatori.

Într-o expresie de atribuire cu operanzi de tipuri aritmetice diferite, se face conversia valorii atribuite la tipul variabilei. Probleme apar cînd tipul valorii atribuite are un domeniu de valori mai mare decât tipul variabilei. Exemplu: a = 5; b=7; a = a + b; a = a \* b;

## Operatori aritmetici. Limbajul C dispune de operatori pentru cele patru operaţii aritmetice de bază şi operatori unari , la care se adaugă operatorul modulo şi doi operatori de incrementare, respectiv decrementare.

## Operatori binari Operatori unari

**+** adunarea **-** (semnul) minus **+** (semnul)plus

**\*** înmulţirea **-** scăderea **--** decrementarea

**/** împărţirea **++** incrementarea **%** restul împărţirii întregi(modulo)

Cu excepţia operatorului modulo **%**, care admite numai operanzi întregi, ceilalţi pot fi aplicaţi tuturor tipurilor aritmetice.

*Atribuirea simplă* este de forma:variabilă = expresie şi are ca efect modificarea valorii unei variabile.

*Atribuirea compusă* a op= b reprezintă într-o formă compactă operaţiaa = a op b

*Atribuirea multiplă* este de forma variabilă1 = variabilă2 = … = expresie şi iniţializează variabilele, pornind de la dreapta spre stânga cu valoarea expresiei.

Operatorii de incrementare folosiţi în atribuiri au efecte diferite. Astfel:

a = ++b este echivalentă cu b=b+1; a=b; în timp ce:

a = b++ are ca efect a=b; b=b+1;

Exemple de expresii care pot fi reduse la o formă prescurtată:

a = a + b; se reduce la a += b;

a = a - b; se reduce la a -= b;

a = a \* b; se reduce la a \*= b;

a = a / b; se reduce la a /= b;

a = a % b; se reduce la a %= b;

a = a << b; se reduce la a <<= b;

a = a >> b; se reduce la a >>= b;

a = a & b; se reduce la a &= b;

a = a | b; se reduce la a |= b;

a = a ^ b; se reduce la a ^= b;

**2.5. Funcţiile de citire şi afişare a datelor**

Pentru citirea unor valori introduse de la terminal, sub controlul unui format se foloseşte funcţia:

scanf(lista\_descriptori, lista\_adrese);

Funcţia **scanf**(c, p1, p2, p3, …) citeşte de la intrarea standardă date sub controlul unor formate. Are mai mulţi parametri. De obicei, parametrul **c** este un şir de caractere scris în ghilimele, care conţine specificatorii de format pentru datele de la intrare. El poate conţine şi caractere albe care sunt neglijate. Parametrii **p1, p2, p3**, … sunt adresele zonelor de memorie unde se vor păstra datele citite. De obicei, adresa unei zone de memorie se exprimă prin operatorul **&** în faţa numelui unei variabile simple sau cu indice. La fiecare dintre aceşti parametri, în şirul de caractere c, îi corespunde un specificator de format, care indică cum se va citi data respectivă de la intrare. Un specificator de format începe cu % şi se termină cu una sau două *litere anumite*. El determină conversia datei de citit din formatul extern în formatul intern.

Funcţia citeşte toate datele ce corespund specificatorilor de format din parametrul c. Dacă o dată de la intrare nu corespunde specificatorului de format, atunci citirea se întrerupe. Funcţia returnează numărul datelor citite corect. Funcţia are prototipul în fişierul <stdio.h>.

Citirea datelor se realizează cu ajutorul funcţiei ***scanf***

Sintaxa: **scanf**(“zona de control”,&arg1,&arg2,…,arg*n*);

Unde zona de control indică formatul citirii datelor de la tastatură.

De exemplu: scanf(“%d%d/n”, &a, &b);

Funcţia ***printf*** este funcţia elementară care se foloseşte la afişarea datelor.

Funcţia **printf**(c, p1, p2, p3, …) scrie la ieşirea standard date sub controlul unor formate. Are unul sau mai mulţi parametri. De obicei, parametrul c este un şir de caractere scris în ghilimele, care conţine textele eventuale de extras şi specificatorii de format eventuali pentru datele de extras. Textele din parametrul c sunt extrase fără schimbare. Secvenţele escape, cum ar fi ‘\n’, ‘\t’, ‘\0’ ş.a., nu sunt extrase, ci sunt “executate”. Parametrii p1, p2, p3, … sunt expresii, valorile cărora vor fi extrase. La fiecare din aceşti parametri, în şirul de caractere c, îi corespunde un specificator de format, care indică cum se va extrage valoarea respectivă. Un specificator de format începe cu % şi se termină cu una sau două litere anumite. El determină conversia valorii de extras din formatul intern în formatul extern. Parametrii p1, p2, p3, … pot lipsi. Atunci vor fi extrase numai textele din parametrul c.

Funcţia returnează numărul de octeţi (caractere) extraşi la ieşirea standard sau –1, în caz de eroare.

Funcţia are prototipul în fişierul <stdio.h>.

**printf**(“zona de control”,&arg1,&arg2,…,arg*n*)

Argumentele pot fi variabile, constante care pot fi afişate. Zona de control specifică formatul afişării şi mesajul ce va însoţi afişarea. Fiecare format este indicat de caracterul *%* urmat de un alt caracter care va specifica conversia necesară.

Echivalenţele Turbo Pascal în Turbo C:

Turbo Pascal (versiunea 3.x) Turbo C

----------------------------------------------------------------

Readln(A,B); scanf("%d%d",&a,&b);

Readln(Name); scanf("%s",name); /\* sau gets(name); \*/

Readln(X,A); scanf("%f%d",&x,&a);

Readln(Ch); scanf("%c",ch);

Read(Kbd,Ch); ch = getch();

Writeln('Hello, world.'); printf("Hello, world.\n");

Write('What''s your name?'); printf("What's your name?");

**Caractere de conversie:**

|  |  |
| --- | --- |
| Caracter | Conversie |
| d | zecimal(decimal) |
| o | octal |
| x | hexazecimală |
| f | real, fixă |
| e | real, mobilă(exponenţial) |
| c | caracter |
| s | şir de caractere |
| u | fară semn |

%\*c- ignoră un caracter introdus de la tastatură

#### Secvenţe Escape:

**\n**- trecerea în următoarea linie

**\t**- afişează o tabulaţie

**\a**- alarmă sonoră

**\b**- deplasează cursorul cu un spaţiu în urmă

**\\**- afişează o bară \

**\”**- afişează ghilimele ”

**Structura unui program C**

La începutul programului C se scrie directivele preprocesorului. Includerea de fisiere din bibliotecile standarde- se face cu directiva **#include**

  #include<nume.fisier>

#include "nume.fisier"

*Exemple:* #include<stdio.h> /\* include biblioteca de intrare iesire \*/

#include<graphics.h> /\* include biblioteca grafica \*/

#include "fisier1.c" /\* include fisier1.c \*/

Un program C contine una sau mai multe functii care specifica operatiile efective de calculat care trebuiesc facute.

Fiecare program contine o functie principala,numita "main()", care va invoca in mod obisnuit alte functii pentru a-si realiza scopul; unele dintre aceste functii invocate vin din acelasi program iar altele din biblioteci ce contin functii scrise anterior.

O functie este apelata prin nume, urmat de o lista de argumente intre paranteze. O metoda de a comunica date intre functii este prin intermediul argumentelor functiei. O functie fara argumente se indica prin "( )".

Acoladele "{ }" includ instructiunile care alcatuiesc functia. Un program C, oricare

i-ar fi marimea, consta din una sau mai multe "functii" care specifica operatiile

efective de calculat care trebuiesc facute.

Functiile din C sint similare cu functiile si subrutinele dintr-un program Fortran sau cu procedurile din Pascal.

In exemplul de mai jos, "main" este o astfel de functie.

In mod normal aveti libertatea de a da functiilor ce nume doriti, dar "main" este un nume special - orice program scris in C se va executa de la inceputul lui "main". Aceasta inseamna ca fiecare program trebuie sa aibe un "main" undeva."main" va invoca in mod obisnuit alte functii pentru a-si realiza scopul, unele venind din acelasi program iar altele din biblioteci ce contin functii scrise anterior.

O metoda de a comunica date intre functii este prin argumentele functiilor. Parantezele care urmeaza dupa numele functiei includ lista de argumente. In cazul nostru, "main" este o functie fara argumente ceea ce se indica prin "()".

Acoladele "{ }" includ instructiunile care alcatuiesc functia. Ele sint analoage lui "begin-end" din ALGOL, PASCAL, etc. O functie este apelata prin nume, urmate de o lista de argumente in paranteze. Parantezele trebuie sa fie prezente chiar daca nu exista argumente.

Un program care afiseaza textul "hello, world!!!!!" este urmatorul:

#include <stdio.h>

int main()

{

printf("hello, world!!!!!\n");

return 0;

}

Linia: ***#include <stdio.h>***  reprezinta o directiva C, care specifica un nume de fisier ce contine prototipuri de functii.

Pentru a fi compilate programele in C toate apelurile de functii, fie ca sint definite in bibliotecile standard C, fie ca sint definite de programator, trebui sa respecte o signatura. Aceasta signatura este compusa din tipul de date al rezultatului intors, daca functia intoarce un rezultat, cit si numarul si tipul de argumente pe care le primeste functia atunci cind este apelata. Pentru functia “printf” cat si pentru alte functii de lucru cu dispozitivele standard de I/O prototipul(signatura) acesteia este descris in fisierul 'stdio.h', care se afla in intr-unul din directoarele pachetului de dezvoltare C.

Linia: ***printf("hello, world!!!!!\n");***  este un apel de functie, care invoca functia numita "printf" cu argumentul ("hello, world!!!!!\n"). "printf" este o functie din biblioteca care tipareste pe monitor (daca nu este specificata o alta destinatie). In acest caz, ea tipareste sirul de caractere care alcatuiesc argumentul.

Exemple programelor în C

#include < stdio.h >

**main()**

{ int a,b; float ratio;

printf("Introduceţi valorile ale două numere întregi: ");

scanf("%d %d,&a,&b); ratio = a / b; printf(" Rezultatul = %f \n", ratio); }

#include < stdio.h >

**main()**

{ int a,b,sum; char \*format;

format = "a = %d b = %d sum = %d \n"; a = b = 5;

sum = a + b; printf(format,a,b,sum); sum = a++ + b; printf(format,a,b,sum);

sum = ++a + b; printf(format,a,b,sum); sum = --a + b; printf(format,a,b,sum);

sum = a-- + b; printf(format,a,b,sum); sum = a + b; printf(format,a,b,sum);

}

#include<stdio.h>

#include<conio.h>

void main()

**{** /\*Declararea variabilor\*/

int a=15,x;

char b='A';

long h=89350,z;

float d=923.2222;

double e=923.2222;

clrscr();

/\*citirea de la tastatură şi afisarea la ecran a valorilor variabilelor\*/

* printf("dati valoare lui a ="); scanf("%d",&a); fflush(stdin);
* printf("dati valoare lui b="); scanf("%c",&b);
* printf("dati valoare lui h="); scanf("%ld",&h);
* printf("dati valoare lui d="); scanf("%f",&d);
* printf("dati valoare lui e="); scanf("%lf",&e);

printf("Variabila de tip intreg(int)::->%d\n",a);

printf("nVariabila de tip caracter(char)::->%c\n",b);

printf("Variabila de tip intreg long(long)::->%ld\n",h);

printf("Variabila de tip reala(float)::>%f\n",d);

printf("Variabila de tip reala(double)::>%e",e); **}**

## 2.6. Operatori logici şi operaţionali

În limbajul C nu există tipul boolean (adevărat sau fals). Operatorii logici admit operanzi de orice tip scalar pe care îl interpretează conform convenţiei -adevărat=valoare -fals=0

Se pot efectua operaţiile logice de bază (booleene) folosind următoarele simboluri.

**&&** --(and); şi **||** --sau(or) **!** --negare(not)

Rezultatul unei opearţii este de tipul integer şi se bazează pe convenţia:

1-adevărat 0-fals

**Operatorii relaţionali** permit testarea relaţiei dintre două valori de tipuri aritmetice sau doi pointeri

<,<=,>,>=,= =,!=.

Valoarea expresiei relaţionale este de tip întreg.

Operatorii de relaţie **sunt: < -** mai mic, **<= -** mai mic sau egal, > - mai mare, >= - mai mare sau egal.

Operatorii de egalitate **sunt:** == - egal, != - diferit.

Operatori la nivel de bit. Această familie de operatori constituie una din extinderile limbajului C. Operatorii logici pe biţi **sunt: ~ -** complement faţă de 1, << - deplasare la stînga, >> - deplasare la dreapta, & - şi logic pe biţi, ^ - sau exclusiv logic pe biţi, | - sau logic pe biţi**. Aceşti operatori se aplică la operanzi de tip** int**.** Operatorul de nergare este unar, iar ceilalţi sunt binari.

**Operatorii de atribuire combinaţi**. Operatorul de atribuire poate fi combinat cu o serie de operatori aritmetici şi operatori la nivel de bit. Există zece combbinaţi posibile. += -= **%= >>=**

**\*= /= ^= &= |= <<=**

**Expresii.** Expresiile se evaluează pe baza unui set de reguli care precizează precedenţa şi asociativitatea operatorilor precum şi conversiile aplicate operanzilor. Precedenţa determină ordinea de efectuare a operaţiilor într-o expresie cu diverşi operatori. Asociativitatea indică ordinea de efectuare a operaţiilor într-o secvenţă de operaţii care au aceiaşi precedenţă. Expresia finală este adevărată dacă nici o valoare nu este

**Comenzi** Comenzile nu sunt altceva decât nişte instrucţiuni. Deosebim mai multe tipuri de instrucţiuni.

INSTRUCTIUNI In sens general o instructiune este o portiune a programului care poate fi executata. Deci o instructiune specifica o actiune. Standardul ANSI C impart instructiunile in urmatoarele grupe:

Instructiunile de selectie (conditionala) cuprind *if* si *switch.*

Instructiunile iterare (de buclare) sunt *while*, *for* si *do-while.*

Instructiunile de salt sunt *break*, *continue*, *goto* si *return*.

Instructiunile eticheta sunt *case* si *default* care sunt prezentate impreuna cu instructiunea *switch.*

Instructiunile expresie sunt instructiuni compuse dintr-o expresie valida.

Instructiunile bloc sunt simple blocuri de cod. Standardul ANSI C++ mai denumeste instructiunile bloc si *instructiuni compuse*.

Instrucţiunea if  **este o instrucţiune de ramificare. Ea are două formate: incomplet şi complet. Formatul incomplet este:** if(expresie) instrucţiune1;  **Ea realizează următoarea construcţie de ramificare:**

true false

**expresie**

### instrucţiune

La întîlnirea acestei instrucţiuni se evaluează expresia din paranteze. Dacă expresia are valoarea true, adică o valoare diferită de 0, atunci se execută instrucţiune1 şi apoi se trece la instrucţiunea imediat următoare după instrucţiunea if. În caz contrar, se trece imediat la instrucţiunea următoare.

Formatul complet al instrucţiunii if este: if(expresie) instrucţiune1; else instrucţiune2; Ea realizează următoarea construcţie de ramificare:

expresie

### instrucţiune1

### instrucţiune2

Aici se evaluează expresia din paranteze şi în caz de adevăr se execută instrucţiune1, altfel se execută instrucţiune2. După aceasta se trece la instrucţiunea următoare după if. Instrucţiunile instrucţiune1 şi instrucţiune2 pot fi simple şi compuse. **Instrucţiunile** if **pot fi incluse una în alta.**

Instrucţiunea **goto** este o instrucţiune de salt necondiţionat. Ea are formatul goto nume; unde nume este numele unei etichete. Eticheta este un identificator (un nume) care se scrie în faţa unei instrucţiuni cu simbolul ‘:’ după el. de exemplu lab1: i++; Aici numele lab1 este o etichetă. La întîlnirea instrucţiunii goto lab1; se trece imediat la execuţia instrucţiunii cu eticheta lab1 în faţă, adică la instrucţiunea i++;

## Instrucţiuni compuse

**{**<lista\_declarativa>; lista \_instructiuni**}**

lista\_declarativa;

declaratie;

lista\_declarativa declaratie;

lista\_instructiuni;

instructiune;

lista\_instructiuni instructiune.

#include<stdio.h>

#include<math.h>

void main(void)

{float a,b,c;

puts("Introduceti variabilele\n");

printf("a= ");

printf("\n");

scanf("%f",&a);

printf("b= ");

printf("\n");

scanf("%f",&b);

c=a+b;

printf("Suma este %f",c);

}

Figura 4.5. Codul de program în C

###### **3. Insntrucţiunea condiţională if în C**

**3.1 DECIZIA SIMPLA: if – else**

Această instrucţiune permite programarea unei structuri de decizie în care o condiţie determină

-executarea sau nu a unei secvenţe de instrucţiuni.

-executarea unei secvenţe din două alternative.

Sintaxa: **if(***espresie***) instr\_a; if(***espresie***) instr\_a;** < **else instr\_1;** >

Ramura **else,** inclusă între < >, este opţională. Execuţia începe cu evaluarea lui **expresie:** dacă este adevărată (diferită de 0), atunci se execută **instr\_1.** Dacă **expresie** este falsă (0) şi există ramură **else,** se execută **instr\_2,** iar dacă nu există ramură **else,** controlul trece la instrucţiunea următoare. Construcţia de mai sus, cu sau fără ramură **else,** este văzută ca **o singură instrucţiune C:** instrucţiunea de tip **if.** Instrucţiunile **instr\_1** şi **instr\_2** sunt considerate instrucţiuni simple. Dacă este cazul, se pot folosi blocuri de instrucţiuni pe oricare din ramuri.

Deoarece o construcţie **if** cu ramură **else** este interpretată ca o singură instrucţiune, ea poate apare pe post de **instr\_1** într-o altă instrucţiune **if,** care nu are ramură **else.** Există acum o ambiguitate: cărui **if** îi aparţine ramura **else** ? Iată un exemplu: **if (y == 1) if(a == 2) x = 0; else x = 2;**

În C, această ambiguitate este rezolvată conform regulii:o ramură **else** aparţine celei mai apropiate instrucţiuni if anterioare**,** din cadrul aceluiaşi bloc, care nu are ramură else**.** În exemplul precedent, ramura **else** aparţine lui if(a == 2). Dacă se doreşte o altă apartenenţă, se foloseşte o instrucţiune compusă (un bloc):

**if (y == 1) { if(a == 2) x = 0; } else x = 2;**

Ramura **else** aparţine acum lui **if** (y == 1). Trebuie observat că, dacă if-ul exterior are ramură **else,** sau dacă if-ul interior este pe ramura **else** (deci pe post de **instr\_2),** atunci nu apare nici o ambiguitate şi regula de mai sus este superfluă. Iată câteva exemple:

if(e1) if(e1)

if(e2) x=1;

else x=2; if(e2) else x=3;

O altfel de scriere se poate în felul următor:

if(x != 0); se poate reduce la if(x);

if(x == 0); se poate reduce la if(!x);

#include <stdio.h>

void main(void)

{

int r;

printf ("Introduceti rezultatul la testului:");

scanf("%d",&r);

if (r >= 90)

printf( "Felicitari ai luat 10!");

else printf(„Poti mai mult !”);

}

**3.2. DECIZIA MULTIPLA: if - else- if**

Construcţiile de tipul celui de-al doilea exemplu anterior se numesc **if-else-if** şi, deşi nu reprezintă un tip distinct de instrucţiune, merită tratate separat. Forma generală a unei asemenea construcţii este:

**if (expr1) instz1 else if (expr1) instr2**

**............................... else if (exprn) instrn  < else instrn >**

Trebuie întâi observat că această construcţie este interpretată ca o singură instrucţiune **if:** cea de la nivel exterior. Se va executa una şi numai una din **instr1,..., instrn+1.** Dacă **expr1** este adevărată, se execută **instr1;** dacă **expr2** e adevărată, se execută **instr2** etc. Dacă nici una din expresii nu este adevărată şi este prezentă ultima ramură else, se va executa **instrn+1.** Dacă ultima ramură **else** lipseşte, atunci se trece la instrucţiunea următoare.

Ordinea condiţiilor este importantă: dacă toate expresiile sunt adevărate, se va executa numai **instr1.**

Sintaxa este:

if (conditie)

instructiune 1;

else

if (conditie)

else instructiune 2;

Conditiile sunt evaluate de sus pana jos. Cand este intalnita o conditie adevarata, va fi executata instructiunea care ii corespunde. Daca nici o conditie nu este adevarata este executata instructiune care corespunde ultimului *else.*

Un exemplu de astfel de program este urmatorul:

#include <iostream.h>

void main(void)

{

int Rezultat;

cout << "Introduceti rezultatul testului si tastati Enter: ";

cin >> Rezultat;

if (Rezultat >= 90)

cout << "Vedeti ca se poate?" << endl;

else if (Rezultat >= 80)

cout << "Aproape bine!" << endl;

else if (Rezultat >= 70)

cout << "Se poate si mai bine" << endl;

else if (Rezultat >= 60)

cout << "Nu va puteti multumi cu atat" << endl;

else

cout << "Ati picat cu brio" << endl;

}

**3.3 SELECŢIA: switch**

Este o instrucţiune de selecţie ce permite executarea unei singure secvenţe din mai multe alternative, în funcţie de valoarea unei expresii. În instrucţiunea **switch** se testează dacă o expresie coincide cu o valoare constantă întreagă, dintr-o listă de valori date, executându-se un salt la o etichetă asociată cu fiecare constantă. Forma generală este:

**switch(expr) {**

**case ex\_ct\_1: instrucţiuni;**

**case ex\_ct\_2: instrucţinni;…..**...........................

**case ex\_ct\_n: instrucţiuni;**

**[ default: instrucţiuni ]**

**}**

Se evaluează **expr;** dacă este egală cu una din expresiile constante **ex\_ct\_1,...,ex\_ct\_n** (care trebuie să fie distincte), se transferă controlul la eticheta (cazul) respectiv. Spre deosebire de alte limbaje, se vor executa instrucţiunile corespunzătoare cazului respectiv **precum şi toate cele care urmează până la sfârşitul lui switch.**

Un caz poate conţine una, mai multe sau nici o instrucţiune, adică pot exista mai multe cazuri asociate la acelaşi grup de instrucţiuni. Dacă **expr** nu este egală cu nici una din expresiile constante, controlul trece la eticheta **default. Default** este opţional: dacă lipseşte, se trece la execuţia următoarei instrucţiuni după **switch.** Ordinea cazurilor, inclusiv default poate fi oarecare.

Sintaxa este:

***switch*** (expresie) {

case const1:

instructiune 1;

break;

case const2:

instructiune 2;

break;

|

|

case constN:

instructiuneN;

break;

default:

instructiune4;

}

Dacă se doreşte ca la un anumit caz să se execute numai grupul de instrucţiuni asociat, nu şi cele care urmează, atunci se va pune, ca ultimă instrucţiune din grupul respectiv, instrucţiunea **break,** prin aceasta forţându-se ieşirea din **switch.** Instrucţiunea break **se foloseşte numai în corpul unui ciclu sau în instrucţiunea** switch**. La întîlnirea instrucţiunii** break **în corpul unui ciclu se termină execuţia ciclului şi se trece la instrucţiunea după ciclu. La întîlnirea instrucţiunii** break **în instrucţiunea** switch **se iese din instrucţiunea** switch **şi se trece la instrucţiunea următoare după** switch**.** Iată un exemplu:

**switch(c = getch( )) {**

**case 'd':**

**case 'D': del\_file( ); break;**

**case ' l':**

**case 'L': list\_file( ); break;**

**detault : error ( ); break;**

}

Se citeşte un caracter de la consolă (cu getch( )) şi se execută una din funcţiile del\_file( ), list\_file( ) sau error( ), după cum caracterul este 'd' sau 'D', respectiv T sau 'L', respectiv diferit de cele patru caractere. break asigură ieşirea din switch după execuţia funcţiilor. break-ul de la default nu este necesar, dar este o bună măsură de protecţie, în cazul în care se adaugă ulterior alte cazuri, după default.

În cazul în care **switch** este într-o funcţie, se poate folosi, în loc de **break,** instrucţiunea **return,** provocând ieşirea din **switch** şi, totodată, revenirea în programul apelant.

Exemplu: Găsiţi eroarea:

switch(c){

case 1: x++; break;

case 2: y++; break;

defalt: z++; break;

}

Dacă c=3, atunci z++ nu se efectuează. De ce? (Pentrucă, defalt: - este etichetă, şi nu cuvântul cheie default).

Efect: daca valoarea expresiei este egala cu una din constantele corespunzatoare fiecarei etichete case, se executa instructiunea aferenta constantei respective.

Fiecare grup case trebuie terminat cu o instructiune break.

Daca lipseste ***break,*** se executa toate instructiunile pana la primul ***break***. Instuctiunea corespunzatoare cazului implicit (default) este executata atunci cand expresia nu ia nici una din valorile constantelor. Default este optional si, daca nu este prezent, nu are loc nici o actiune daca nu se gaseste o constanta potrivita.

Standardul ANSI C stipuleaza ca ***switch*** poate sa aiba cel mult 257 de instructiuni de tip ***case***.

Desi ***case*** este o instructiune eticheta ea nu poate exista de una singura, in afara unei instructiuni ***switch.***

Instructiunea break este o instructiune de salt in C. Cand este intalnita o instructiune de acest gen in constructia ***switch***, programul executa un salt la linia de cod care urmeaza instructiunii ***switch.***

Trebuie mentionate urmatoarele:

* ***switch*** difera de ***if*** prin aceea ca testeaza doar egalitatea, in timp ce ***if*** poate sa evalueze orice tip de expresie relationala sau logica;
* in acelasi ***switch*** nu pot exista doua constante ***case*** cu valori identice. Desigur, doua instructiuni ***switch,*** una inclusa in cealalta, pot sa aiba aceeasi constanta ***case***.
* Daca in instructiunea ***switch***  sunt utilizate constante de tip caracter, ele sunt automat convertite in intregi.

#include<stdio.h>

void main()

{ int c, i, spatii, altele, cifre[10];

spatii=altele=0;

for (i=0; i<10;i++) cifre[i]=0;

while((c=getchar())!=EOF)

switch (c ) {

case '0':

case '1':

case '2':

case '3':

case '4':

case '5':

case '6':

case '7':

case '8':

case '9':

cifre[c-'0']++;

break;

case ' ':

case '\n':

case\t:

spatii++;

break;

default:

altele++;

break;

}

printf("cifre=");

for (i=0; i<10;i++)

printf("%d ",cifre[i]);

printf("\n spatii=%d, alte caractere=%d\n", spatii, altele);

}

Citirea caracterilor. Se citeşte un caracter de la consolă (cu getch( )) şi se execută una din funcţiile del\_file( ), list\_file( ) sau error( ), după cum caracterul este 'd' sau 'D', respectiv T sau 'L', respectiv diferit de cele patru caractere. break asigură ieşirea din switch după execuţia funcţiilor. break-ul de la default nu este necesar, dar este o bună măsură de protecţie, în cazul în care se adaugă ulterior alte cazuri, după default.

**Testarea şi depanarea programului.**

Dacă s-a depăşit faza erorilor sintactice la execuţie, vom testa programul, furnizându-i date de intrare pentru care cunoaştem ieşirile. Apariţia unor neconcordanţe indică prezenţa unor *erori logice*. Dacă însă, rezultatele sunt corecte, nu avem certitudinea că programul funcţionează corect în toate situaţiile. Prin testare putem constata prezenţa erorilor, nu însă şi absenţa lor.

Desfăşurarea calculelor poate fi controlată prin execuţie pas cu pas,sau prin asigurarea unor puncte de întrerupere în care să inspectăm starea programului, folosind în acest scop un *depanator* (debugger).

In urmatoarea figura voi prezenta modelul conceptual al elaborarii unui program in C sau C++.

* 1. E X E M P L E: Modele de program C în baza algoritmilor

**Definirea problemei de rezolvat.**  Analiza problemei cuprinde în afara formulării problemei în limbaj natural, o precizare riguroasă a intrărilor (datelor problemei) şi a ieşirilor (rezultatelor).

De exemplu ne propunem să rezolvăm ecuaţia de gradul 2: ax2+bx+c=0

Datele de intrare sunt cei 3 coeficienţi a, b, c ai ecuaţiei, care precizează o anumită ecuaţie de grad 2.

Rezultatele sunt cele două rădăcini (reale sau complexe) sau celelalte situaţii particulare care pot apare.

**2.Identificarea paşilor necesari pentru rezolvarea problemei** începe cuformularea ***modelului matematic*.**

Vom distinge următoarele situaţii:

1. a=0, caz în care nu avem de a face cu o ecuaţie de gradul 2, ci

1.1. este posibil să avem ecuaţia de gradul 1: bx+c=0, cu soluţia

x= - c/b , dacă b  0.

1.2. dacă şi b=0, atunci

* + 1. dacă c0, atunci nu avem nici o soluţie, în timp ce
    2. dacă şi c=0, atunci avem o infinitate de soluţii.

2. a0 corespunde ecuaţiei de gradul 2. In acest caz avem alte două situaţii:

* 1. Formula este aplicabilă pentru rădăcini reale (discriminant pozitiv)
  2. Pentru discriminant negativ, întrucât nu dispunem de aritmetică complexă, va trebui să efectuăm separat calculele pentru partea reală şi cea imaginară.

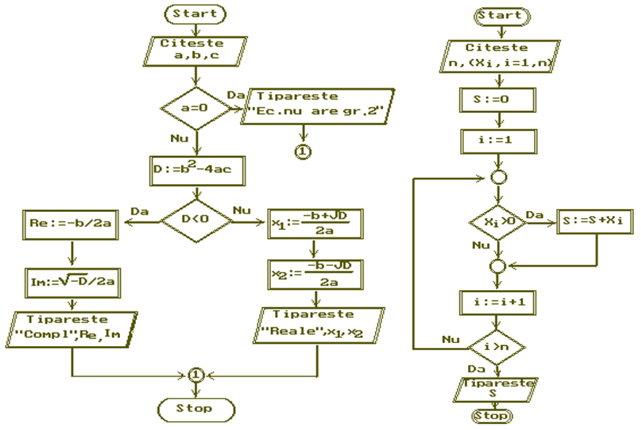


Fig.a. Algoritm pentru rezolvarea ecuaţiei de gradul doi

Vom codifica algoritmul descris mai sus folosind limbajul C:

#include <stdio.h>

#include <math.h>

void main(void)

{ double a,b,c,delta,x1,x2,xr,xi;

scanf(“%f %f %f”,&a,&b,&c);

if (a==0)

if (b!=0) printf(“o singura radacina x=%6.2f\n”,-c/b);

else

if (c!=0) printf(“nici o solutie\n”);

else printf(“o infinitate de solutii\n”);

else

{ delta=b\*b-4\*a\*c;

if(delta >= 0)

{ x1=(-b-sqrt(delta))/2/a;

x2=(-b+sqrt(delta))/2/a;

printf(“x1=%5.2f\tx2=%5.2f\n”,x1,x2);

}

else

{ xr=-b/2/a; xi=sqrt(-delta)/2/a;

printf(“x1=%5.2f+i\*%5.2f\nx2=%5.2f-i\*%5.2f\n”,

xr,xi,xr,xi);

}

}

}

# **Întrebările de autocontrol**

1. **Alcătuiţi schemele logice pentru instrucţiunile: if completă şi trunchiată; switch cu selecţia valorilor întregi şi caracteriale. Analizaţi-le şi apreciaţi avantajele şi dezavantajele utilizării fiecăreia.**
2. **Transcrieţi exemplele programelor de mai sus, utilizînd instrucţiunea switch.**
3. **De alcătuit schemele logice şi de verificat ce se v-a obţine pentru următoarele secvenţe de program:**

**a) {/\* pentru x=1, a=-1 şi a=1 \*/**

**if a>0 f=2\*a\*x+abs(a-1);**

**else f=exp(sqr(x))/sqrt (sqr(a)); write(‘f=’, f);**

**b) {i:=1;**

**/\* pentru x[1]=3, x[2]=4, x[3]=5…x[n]=i+2 i=1…n, n=5 \*/**

**1: y[i]:=1+abs (x[i]);**

**y[i]:=(1+sqr(x[i])); printf(‘i=’, i:2,’ y=’,y[i]); i:=i+1; if i<5 goto 1 };**

**c) { /\* pentru b=3, c=2, a=-5 şi a=1 \*/**

**if a<>0**

**{ D:=sqr (b)-4\*a\*c;**

**if d<0 { printf('n-are radăcini'); exit();}**

**if D>0 { x1:=(-b+sqrt( D))/(2\*a);**

**x2:=(-b-sqrt(D))/2\*a;**

**printf(‘\n x1= ’, x1:6:3,‘ x2= ’, x2:6:3); }**

**else { x1:=(-b+sqrt(D)/(2\*a); printf(‘x1=x2= ’,x1); }}}**

**d) Transcrieţi din TP în C:**

**begin {pentru b=4, c=3, a=-4 şi a=1 \*/**

**if a<>0 then**

**begin D:=sqr (b)-4\*a\*c;**

**if D>0 then begin x1:=(-b+sqrt( D))/(2\*a);**

**x2:=(-b-sqrt(D))/2\*a;**

**writeln(‘x1= ’, x1:6:3, ‘x2=’, x2:6:3) end**

**else begin if D=0 then x1:=(-b+sqrt(D)/(2\*a);**

**write(‘x1=x2= ’, x1); end;**

**if d<0 then Writelen ('n-are radacini');**

**end; End.**

**e) verificaţi ce se efectuează:**

***Schema-bloc***

**START**

**j=1;**

**k=2;**

**h=0.94;**

**a= exp(-h\*k\*j)\*cos(h\*k-j)+sqrt(fabs(j-k));**

**b=a\*log(a\*pow(j,4)-sin(2\*h))-k;**

**Afiş.: a,b**

**STOP**

**Rezultatul este:**

***Listingul programului***

#include<stdio.h>

#include<conio.h>

#include<math.h>

void main() {

float j=1,k=2,h=0.94,a,b;

clrscr();

a=exp(-h\*k\*j)\*cos(h\*k-j)+sqrt(fabs(j-k));

b=a\*log(a\*pow(j,4)-sin(2\*h))-k;

printf("Rezultatul este:\na=%.2f\nb=%.2f",a,b);

getch();

}

**TESTE: DESFASURAREA LUCRARII**

1. Rulati programul **TEST 1** si verificati actiunea altor secvente, notandu-va efectul acestora.

2. Rulati programele **TEST 2** si **TEST 3** si alcatuiti un program care sa citeasca de la tastatura numele si varsta dvs. si sa le afiseze.

3. Rulati programul **TEST 4** si **TEST 5** si alcatuiti un program care sa calculeze mediile aritmetica si armonica a cinci numere.

1. Rulati programele **TEST 6** si **TEST 7** si pe baza acestora alcatuiti un program care sa afiseze toti divizorii unui numar introdus de la tastatura.

// **TEST 1**

#include <stdio.h>

int main()

{

printf("hello, world!!!!!\n");

return 0;

}

Linia:

***#include <stdio.h>***

reprezinta o directiva C, care specifica un nume de fisier ce contine prototipuri de functii.

//**TEST 2**

# include <stdio.h>

# define PI 3.14159

# define name "Cristina"

int main()

{

int int1,int2;

long long1,long2;

float float1, float2;

char char1, char2, char3;

int1=99;

int2=0xFF01;

long1=-68741L;

long2=0xFFF1;

float1=3.14159;

float2=1.5E04;

char1='C';

char2=15;

char3=0xF1;

printf(" int1== %d \n" , int1);

printf(" int2== %#x \n" , int2);

printf(" long1== %ld \n" , long1);

printf(" long2== %ld \n" , long2);

printf(" float1== %f \n" , float1);

printf(" PI== %f \n" , PI);

printf(" float2== %E \n" , float2);

printf(" char1== %c \n" , char1);

printf(" char2== %d \n" , char2);

printf(" char3== %c \n" , char3);

printf(" numele este== %s \n" , name);

}

//**TEST 3**

# include <stdio.h>

# include <conio.h>

void main()

{

clrscr();

int i;

char ch;

float fl;

printf("introduceti valori pentru i, c, fl \n");

printf("i=");

scanf("%d",&i);

printf("ch=");

scanf("%c",&ch);

printf("fl=");

scanf("%f",&fl);

printf("i=%d fl=%4.2f ch=%c", i, fl, ch);

}

Instructiunea *scanf()* - citeste caractere de la intrare, le interpreteaza conform

formatului specificat in sirul de control si le memoreaza in argumentele urmatoare.

Sirul de control contine specificatii de conversie care sunt utilizate pentru interpretarea secventelor de intrare:

Caracterele de conversie si semnificatiile lor:

d, i - un intreg zecimal este asteptat la intrare - argumentul va fi un pointer la un intreg;

o - un intreg octal este asteptat la intrare - argumentul va fi un pointer la un intreg;

x - un intreg hexazecimal este asteptat la intrare - argumentul va fi un pointer la un intreg;

h - un intreg scurt este asteptat la intrare - argumentul va fi un pointer la un intreg scurt;

c - un singur caracter este asteptat la intrare - argumentul va fi un pointer la un caracter;

s - un sir de caractere este asteptat la intrare - argumentul va fi un pointer la un tabel de caractere;

f - un numar in virgula flotanta este asteptat la intrare - argumentul va fi un pointer la un numar in virgula flotanta;

//**TEST 4**

# include <stdio.h>

# include <conio.h>

void main()

{

int m,n, i=33, j=55;

float f2=12.23f;

m=i+j;

n=1;

n+=j;

f2/=3;

int z=i%5;

printf("m=%d \t n=%d\n",m,n);

printf("f2=%f \t z=%d\n",f2,z);

printf("++i este %d\n" , ++i);

printf("i este %d\n" ,i);

printf("i++ este %d\n" ,i++);

printf("i este %d\n" ,i);

}

//**TEST 5**

# include <stdio.h>

# include <conio.h>

void main()

{

clrscr();

int m,n, i=33, j=55;

float f1,f2=12.23f;

m=i+j;

n+=j;

f2/=3;

printf("m=%d \t n=%d\n",m,n);

printf("f2=%f\n",f2);

printf("i+1 este %d iar noua valoare a lui i este %d" , ++i, i);

printf("j este %d iar noua valoare a lui j este %d" , j++, j);

}

//**TEST 6**

Sa se verifice daca numarul n introdus de la tastatura este sau nu prim.

n=prim <=> n se imparte fara rest la a cu 2 <= a <= [radical n]

#include <math.h>

#include <conio.h>

#include <stdio.h>

//#include <iostream.h> il utilizati cand folositi cout si cin

void main()

{

unsigned int n,i,cont=0;

clrscr();

printf ("Introduceti un numar : ");

scanf ("%d",&n);

/\*cout << "Introduceti un numar : ";

cin >> n;\*/

for (i=2;i<floor(sqrt(n));i++) // impartim pe n la i care creste cu 1 de fiecare data pana la parte intreaga (floor) din radical (sqrt) de n

{

if (n%i==0) //daca n se imparte fix (%) la i atunci nu este prim

{

printf("Numarul %d nu este prim!",n);

/\*cout << "Numarul " <<n<< "nu este prim!";\*/

cont=1; //in cazul in care nu este prim cu ajutorul lui cont avem grija sa nu mai afiseze ca este prim

break; //cand am aflat ca nu este prim iesim din for astfel scurtam timpul de ciclare

}

}

if (cont!=1) //daca este prim cont ramane 0 cum l-am initializat si se va afisa ca numarul n este prim

{

printf("Numarul %d este prim!",n);

/\*cout << "Numarul " <<n<< "este prim!";\*/

}

getch(); //asteapta introducerea unui caracter

}

**OBS:** **<<** este operatorul de insertie

**>>** este operatorul de extractie

**cout** este "console out"

**cin** este "console in"

//**TEST 7**

Sa se determine cel mai mic divizor propriu al lui n introdus de la tastatura.

#include <conio.h>

#include <stdio.h>

//#include <iostream.h> il utilizati cand folositi cout si cin

void main()

{

unsigned int n,i,u,d[40],cont=0;

clrscr();

printf ("Introduceti un numar : ");

scanf ("%d",&n);

/\*cout << "Introduceti un numar : ";

cin >> n;\*/

for (i=2;i<=(n/2);i++) // mergem pana la jumatatea numarului

if (n%i==0) //daca n se imparte fix (%) la i atunci i este divizor

{

printf("Cel mai mic divizor propriu al lui %d este %d ",n,i);

//cout << "Cel mai mic divizor al lui "<<n<<" este "<<i;

cont=1;

break;

}

if(cont==0)

printf("Numarul este prim!");

//cout <<"Numarul este prim!";

getch(); //asteapta introducerea unui caracter

}

Bibliografie:

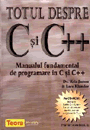
1. Brian Kernighan, Dennis Ritchie *” The C Programming Language”* traducere în română
2. Аммерал Л. Машинная графика на языке Си.(4 кн.) М.: СолСистем. 1993
3. Джехани Н. Программирование на языке Си. М.: Радио и св.. 1988. /681.3; Д409/
4. Уэйтм Н. И др. Язык Си. М.: Радио и св.. 1988./681.3; У97/
5. Берри Р., Микинз Б. Язык Си. Вв. для программистов. М.: Финансы и ст. 1988./681.3; Б518/
6. Thomas H. Cormen, Charles E. Leiserson, Ronald R.Rivest, *Introducere în Algoritmi*, editura Computer Libris Agora
7. Informatica. Indrumar de laborator./A.Popescu, S. Marin s.a./- Chişinău.: UTM, 2003.

[www.ace.tuiasi.ro/ro/academice/curricula/programe/ingineri/pa\_109\_%20pc\_serban.htm](http://www.ace.tuiasi.ro/ro/academice/curricula/programe/ingineri/pa_109_%20pc_serban.htm)

Programarea in **limbajul C++** pentru liceu. Volumul al II-lea: Metode si tehnici de **...**www.pri**c**e.ro/preturi\_**limbaj**e\_de\_**program**are\_si\_**algoritm**i

Bazele programarii in **C**. Aplicatii. Autor (i): Doina Logofatu. Editura [Polirom](http://www.kaboo.ro/Results---1--X--X--Polirom--X--X.html)

Programarea in limbajul **C**/**C++** pentru liceu Autor (i): Marinel Serban , Emanuela Cerchez Editura [Polirom](http://www.kaboo.ro/Results---1--X--X--Polirom--X--X.html)

1.  Dr. Kris Jamsa si Lars Klander ’’Totul despre C si C++’’, traducere de Eugen Dumitrescu