

1. Un tur de orizont în limbajul C.

1.1. Structura unui program C foarte simplu

Un *limbaj de programare* reprezintă o interfață între *problema de rezolvat* și *programul de rezolvare*. Limbajul de programare, prin specificarea unor acțiuni care trebuie executate *eficient* este *apropiat de mașină*. Pe de altă parte, el trebuie să fie *apropiat de problema de rezolvat*, astfel încât soluția problemei să fie exprimată *direct* și *concis*.

Trecerea de la specificarea problemei la program nu este directă, ci presupune parcurgerea mai multor *etape*:

- **analiza și abstractizarea problemei.** În această etapă se identifică *obiectele* implicate în rezolvare și acțiunile de transformare corespunzătoare. Ca rezultat al acestei etape se crează un *univers abstract al problemei* (UP), care evidențiază o mulțime de tipuri de obiecte, relațiile dintre acestea și restricțiile de prelucrare necesare rezolvării problemei.
- **găsirea metodei de rezolvare** acceptabile, precizând operatorii de prelucrare ai obiectelor din UP.
- **elaborarea algoritmului de rezolvare**
- **codificarea algoritmului**

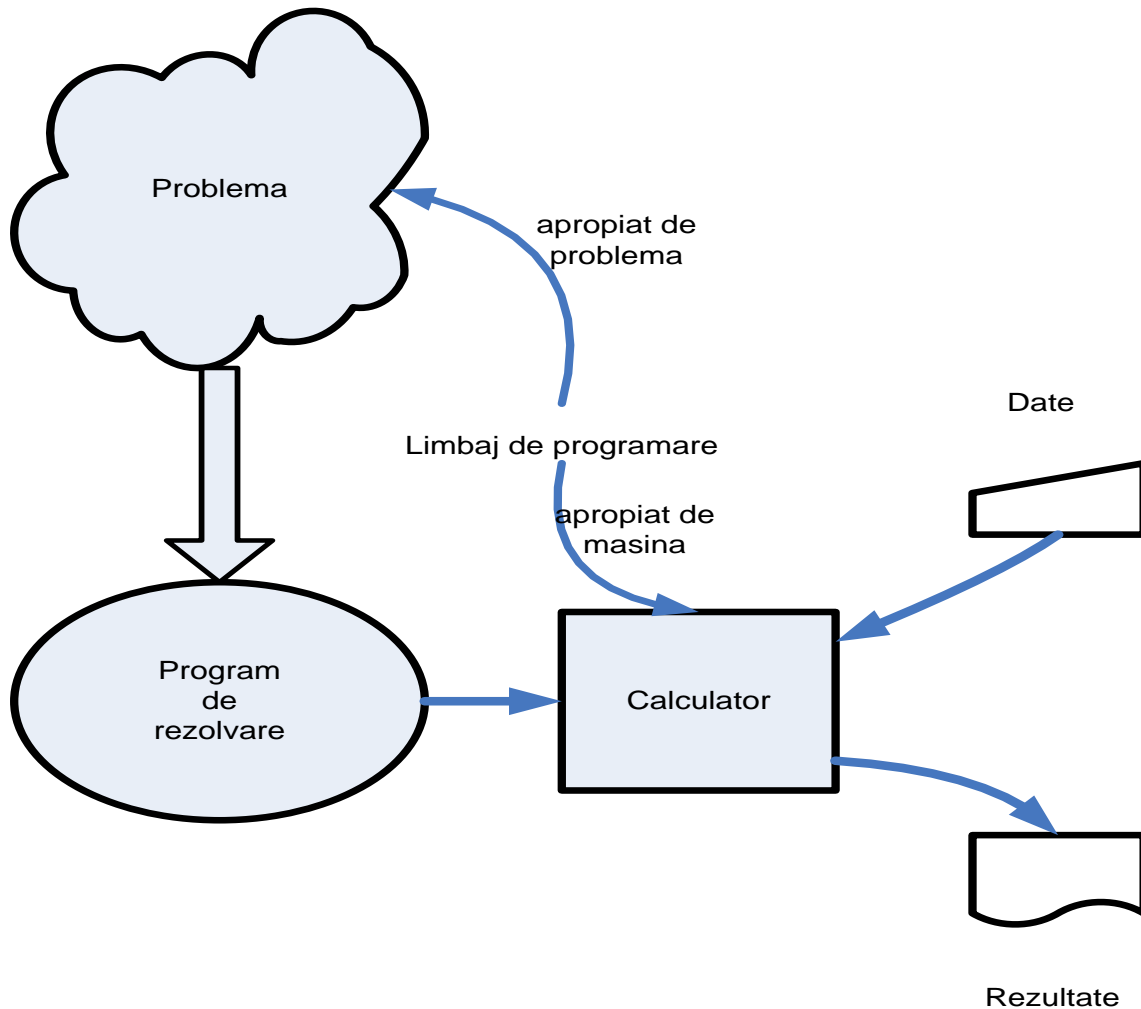


Fig.1.1. Rezolvarea unei probleme folosind calculatorul

Limbajul C s-a impus în elaborarea programelor datorită:

- ușurinței de reprezentare a obiectelor cu caracter nenumeric
- capacității de reprezentare a obiectelor dinamice
- capacității de exploatare a caracteristicilor mașinii de calcul pentru controlul strict al performanțelor programului
- asigurării unei interfețe transparente cu sistemul de operare al mașinii utilizate.

Limbajul C a fost creat de Dennis Ritchie și Brian Kernighan și implementat pe o mașina DEC PDP 11, cu intenția înlocuirii limbajului de asamblare.. Limbajul are precursori direcți limbajele BCPL (Richards) și B (Thompson). Limbajul este folosit ca mediu de programare pentru sistemul de operare UNIX. Limbajul a fost standardizat în 1983 și 1989.

Limbajul C++ a fost dezvoltat de Bjarne Stroustrup pornind de la limbajul C, începând din anul 1980.

C++ împrumută din Simula 67 *conceptul de clasă* și din limbajul Algol 68 - *supraîncărcarea operatorilor*.

Dintre noutățile introduse de C++ menționăm: *moștenirea multiplă, funcțiile membre statice și funcțiile membre constante, șabloanele, tratarea excepțiilor, identificarea tipurilor la execuție, spațiile de nume, etc.*

Deși C++ este considerat o extensie a limbajului C, cele două limbaje se bazează pe *paradigme de programare* diferite. Limbajul C folosește *paradigma programării procedurale și structurate*. Conform acesteia, un program este privit ca o mulțime ierarhică de blocuri și proceduri (funcții). Limbajul C++ folosește *paradigma programării orientate pe obiecte*, potrivit căreia un program este constituit dintr-o mulțime de obiecte care interacționează.

Elementul constructiv al unui program C este *funcția*. Un program este constituit dintr-o mulțime de funcții, declarate pe un singur nivel (fără a se imbrica unele în altele), grupate în *module program*.

O *funcție* este o secțiune de program, identificată printr-un nume și parametrizată, construită folosind declarații, definiții și instrucțiuni de prelucrare. Atunci când este apelată, funcția calculează un anumit rezultat sau realizează un anumit efect.

Funcția **main()** este prezentă în orice program C. Execuția programului începe cu **main()**. Funcția **main()** poate întoarce un rezultat întreg (**int**) sau nici un rezultat (**void**). Numai în C este posibil să nu specificăm tipul rezultatului întors de funcție, acesta fiind considerat în mod implicit **int**.

```
/* program C pentru afisarea unui mesaj */
#include <stdio.h>
main() {
    printf("Acesta este primul program in C /n");
}
```

Programul folosește un *comentariu*, delimitat prin */** și **/* care, prin explicații în limbaj natural, crește claritatea programului. Comentariul este constituit dintr-o linie sau mai multe linii, sau poate apare în interiorul unei linii. Nu se pot include comentarii în interiorul altor comentarii.

Linia **#include <stdio.h>** anunță compilatorul că trebuie să insereze *fișierul antet stdio.h*. Acest fișier conține prototipurile unei serii de funcții de intrare și ieșire folosite de majoritatea programelor C. Fișierele antet au prin convenție extensia **.h**. Fișierul de inclus este căutat într-o zonă standard de includere, în care sunt memorate fișierele antet ale compilatorului C, dacă numele este încadrat între paranteze unghiulare (< și >), sau căutarea se face în zona curentă de lucru, dacă

fișierul este încadrat între ghilimele(""). Fișierele antet sunt foarte utile în cazul funcțiilor standard de bibliotecă; fiecare categorie de funcție standard are propriul fișier antet.

Valoarea întoarsă de funcția **main()** este în mod obișnuit 0, având semnificația că nu au fost întâlnite erori, și se asigură prin instrucțiunea **return 0**.

Instrucțiunea **printf()** servește pentru afișarea la terminal (pe ecran) a unor valori *formatate*.

Față de limbajul C, care este considerat un subset, limbajul C++ permite: *abstractizarea datelor, programarea orientată pe obiecte și programarea generică*.

1.2. Câteva elemente necesare scrierii unor programe C foarte simple.

1.2.1. Directiva define.

Directiva **#define nume text** este o *macrodefiniție*. Prelucrarea acesteia, numită *macroexpandare*, înlocuiește fiecare apariție a numelui prin textul asociat.

O aplicație o reprezintă creerea de *constante simbolice*. De exemplu:

```
#define PI      3.14159
#define mesaj  "Bonjour madame"
#define MAX    100
```

O constantă simbolică astfel definită nu poate fi redefinită prin atribuire.

1.2.2. Tipuri.

Fiecărui nume i se asociază un tip, care determină ce operații se pot aplica aceluși nume și cum sunt interpretate acestea. De exemplu:

```
char c='a';    /*c este variabilă caracter initializata cu 'a' */
int f(double); /*f functie de argument real cu rezultat intreg */
```

1.2.3. Definiții și declarații 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 locație de memorie în care se păstrează o valoare.

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

```
int    n, p;
char   c;
float  eps;
```

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

Inițializarea se face numai o dată, înaintea execuției programului.

Variabilele *externe* și *statice* sunt inițializate implicit la 0.

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

O *definiție* este o construcție textuală care asociază unui nume o zonă de memorie (un obiect) și eventual inițializează conținutul zonei cu o valoare corespunzătoare tipului asociat numelui.

1.2.4. Atribuirea.

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

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

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

Operatorii de incrementare folosiți 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;**

1.2.5. Decizia.

Permite alegerea între două alternative, în funcție de valoarea (diferită de 0 sau 0) a unei expresii:

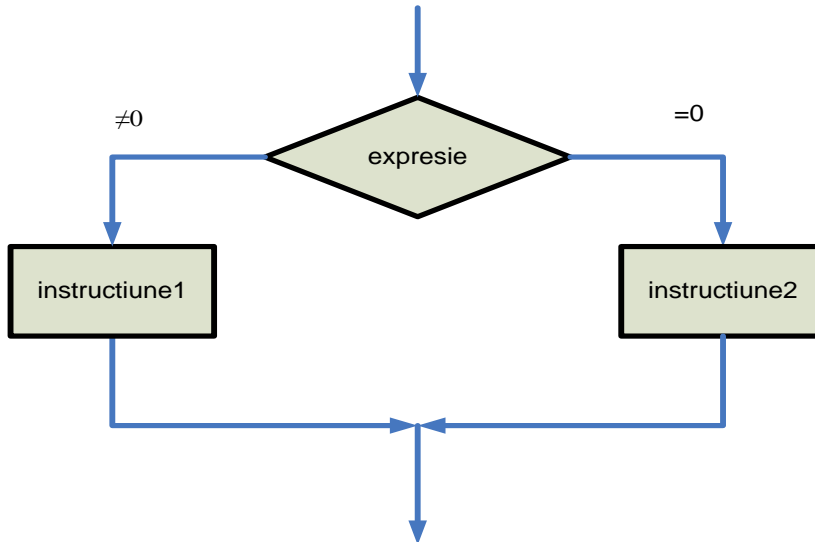


Fig.1.2. Structura de control decizie

```

if (expresie)
    instructiune1;
else
    instructiune2;
  
```

De exemplu:

```

if (a > b)
    max=a;
else
    max=b;
  
```

1.2.6. Ciclul.

Execută în mod repetat instrucțiunea, cât timp condiția este îndeplinită.

```

while (expresie)
    instructiune;
  
```

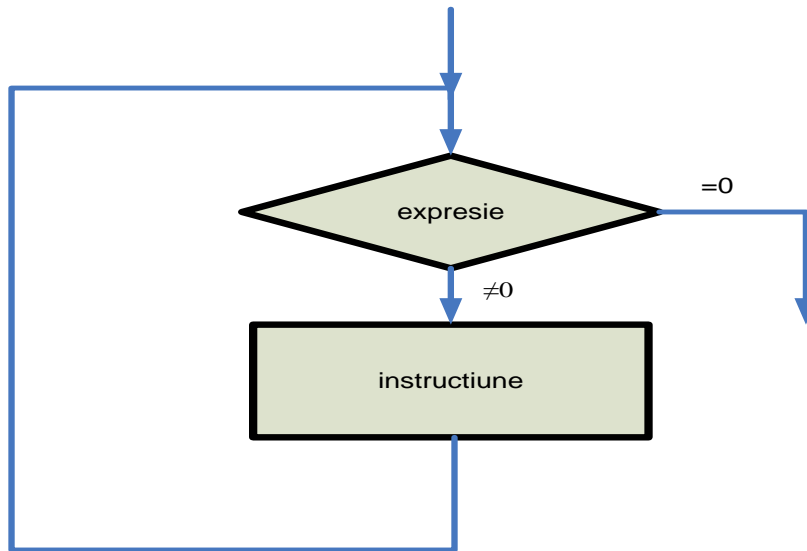


Fig.1.3. Structura de control iterație (ciclu)

De exemplu calculul celui mai mare divizor comun se realizează cu:

```

while (a!=b)
  if (a > b)
    a -=b;
  else
    b -=a;
  
```

1.2.7. Afișarea valorii unei expresii (descriptori),

Pentru afișarea unei valori la terminal, sub controlul unui format se folosește funcția:

```
printf(lista_descriptori, lista_expresii);
```

De exemplu: `printf("pi=%.5f\n", M_PI);`

1.2.8. Citirea valorilor de la terminal.

Pentru citirea unor valori introduse de la terminal, sub controlul unui format se folosește funcția:

```
scanf(lista_descriptori, lista_adrese);
```

De exemplu: `scanf("%d%d/n", &a, &b);`

1.3. Structura unui program.

Un program în C este un ansamblu de funcții și variabile, între care există în mod obligatoriu o funcție cu numele `main()` care se execută întotdeauna prima.

Cea mai simplă structură de program conține o singură funcție `main()`:

```

int main() {
    <declaratii_locale>
    <instructiuni>
}
  
```

2. Elementele fundamentale ale limbajului C .

2.1. Alfabetul limbajului.

Conține setul de caractere ASCII (setul extins 256 caractere), în care un caracter este reprezentat printr-un octet, în care reprezentările caracterelor litere sunt contigue.

2.2. Atomi lexicali.

Există următoarele entități lexicale: identificatori, cuvinte cheie, constante, șiruri de caractere, operatori și separatori.

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

2.2.1. Identificatori.

Identificatorii servesc pentru numirea constantelor simbolice, variabilelor, tipurilor și funcțiilor.

Sunt formați dintr-o literă, urmată eventual de litere sau cifre. Caracterul de subliniere `_` este considerat literă.

Intr-un identificator literele mari și cele mici sunt distincte. Astfel **Abc** și **abc** sunt identificatori diferiți.

Identificatorii pot avea orice lungime, dar numai primele 32 caractere sunt semnificative.

2.2.2. Cuvinte cheie.

Cuvintele cheie sau cuvintele rezervate nu pot fi folosite ca identificatori. Acestea sunt:

Tabel 2.1. Cuvinte cheie

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	if	int	long	register
return	short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void	volatile
while					

2.2.3. Literalii.

Un literal este o reprezentare explicită a unei valori. Literalii pot fi de mai multe tipuri: întregi, caractere, reali, enumerări sau șiruri de caractere..

2.2.4 Șiruri de caractere.

Conțin caractere încadrate între ghilimele. În interiorul unui șir ghilimelele pot fi reprezentate prin `\"`. Un șir de caractere se poate întinde pe mai multe linii, cu condiția ca fiecare sfârșit de linie să fie precedat de `\`.

Șirurile de caractere în C se termină cu *caracterul nul* `'\0'`. În cazul literalelor șiruri de caractere, compilatorul adaugă în mod automat la sfârșitul șirului caracterul nul.

2.2.5. Comentarii.

Un comentariu începe prin `/*`, se termină prin `*/` și se poate întinde pe mai multe linii. Comentariile nu pot fi incluse unele în altele (îmbricate).

2.2.6. Terminatorul de instrucțiune.

Caracterul `;` este folosit ca terminator pentru instrucțiuni și declarații, cu o singură excepție – după o instrucțiune compusă, terminată prin acoladă, nu se pune terminatorul `;`.

2.2.7. Constante.

Constantele identificatori se obțin folosind directiva #define a preprocesorului:

```
#define constantă-identificator    literal    sau    constantă-
identificator
#define constantă-identificator    (expresie-constantă)
```

2.3. Ciclul de dezvoltare al unui program

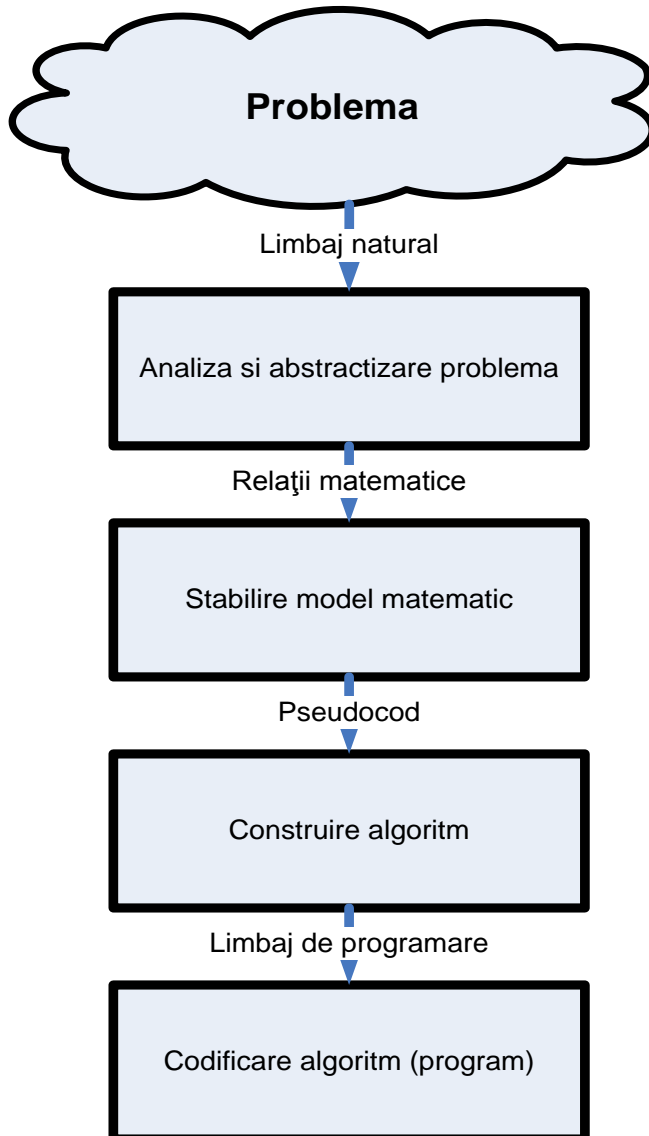


Fig.2.1. Trecerea de la problemă la programul de rezolvare

Conține următoarele etape:

1. Definirea problemei de rezolvat.

Analiza problemei cuprinde în afara formulării problemei în limbaj natural, o precizare riguroasă a intrărilor (datelor problemei) și a ieșirilor (rezultatelor).

De exemplu ne propunem să rezolvăm ecuația de gradul 2: $ax^2+bx+c=0$

Datele de intrare sunt cei 3 coeficienți **a**, **b**, **c** ai ecuației, care precizează o anumită ecuație de grad 2.

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

2. Identificarea pașilor necesari pentru rezolvarea problemei începe cu formularea *modelului matematic*.

Ca model matematic vom folosi formula:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Formula nu este întotdeauna aplicabilă. Vom distinge următoarele situații:

1. **a=0**, caz în care nu avem de a face cu o ecuație de gradul 2, ci
 - 1.1. este posibil să avem ecuația de gradul 1: **bx+c=0**, cu soluția **x= - c/b**, dacă **b ≠ 0**.
 - 1.2. dacă și **b=0**, atunci
 - 1.2.1. dacă **c≠0**, atunci nu avem nici o soluție, în timp ce
 - 1.2.2. dacă și **c=0**, atunci avem o infinitate de soluții.
2. **a≠0** corespunde ecuației de gradul 2. In acest caz avem alte două situații:
 - 2.1. Formula este aplicabilă pentru rădăcini reale (discriminant pozitiv)
 - 2.2. Pentru discriminant negativ, întrucât nu dispunem de aritmetică complexă, va trebui să efectuăm separat calculele pentru partea reală și cea imaginară.

3. Proiectarea algoritmului folosind ca instrument **pseudocodul**.

Pentru simplificare, în pseudocodul utilizat s-a evitat folosirea vreunei sintaxe. Ierarhizarea acțiunilor în structurile de control introduse a fost făcută numai prin indentare (aliniere). Acțiunile componente (subordonate) dintr-o structură de control au fost scrise decalat spre dreapta, în timp ce acțiunile independente au aceeași aliniere. Pseudocodul utilizat cuprinde:

1. operații de intrare / ieșire:


```

citește var1, var2, ...
scrie expresie1, expresie2, ...
      
```
2. structuri de control:
 - 2.1. decizia:


```

dacă expresie atunci
    instrucțiune1
altfel
    instrucțiune2
          
```
 - 2.2. ciclul:


```

cât timp expresie repetă
    instrucțiune
          
```
 - 2.3. secvența:


```

instrucțiune1
...
instrucțiunen
          
```

Algoritmul dezvoltat pe baza acestui pseudocod este:

```

reali a,b,c,delta,x1,x2,xr,xi
citește a,b,c
dacă a=0 atunci

```



```

dacă b≠0 atunci
    scrie -c/b
altfel
    dacă c≠0 atunci
        scrie "nu avem nici o solutie"
    altfel
        scrie "o infinitate de solutii"
altfel
    delta=b*b-4*a*c
    dacă delta >= 0 atunci
        x1=(-b-sqrt(delta))/(2*a)
        x2=(-b+sqrt(delta))/(2*a)
        scrie x1,x2
    altfel
        xr=-b/(2*a)
        xi=sqrt(-delta)/(2*a)
        scrie xr,xi;

```

4. Scrierea programului folosind un limbaj de programare.

Vom codifica algoritmul descris mai sus folosind limbajul C:

```

#include <stdio.h>
#include <math.h>
int main()
{ 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);
        }
    }
}

```

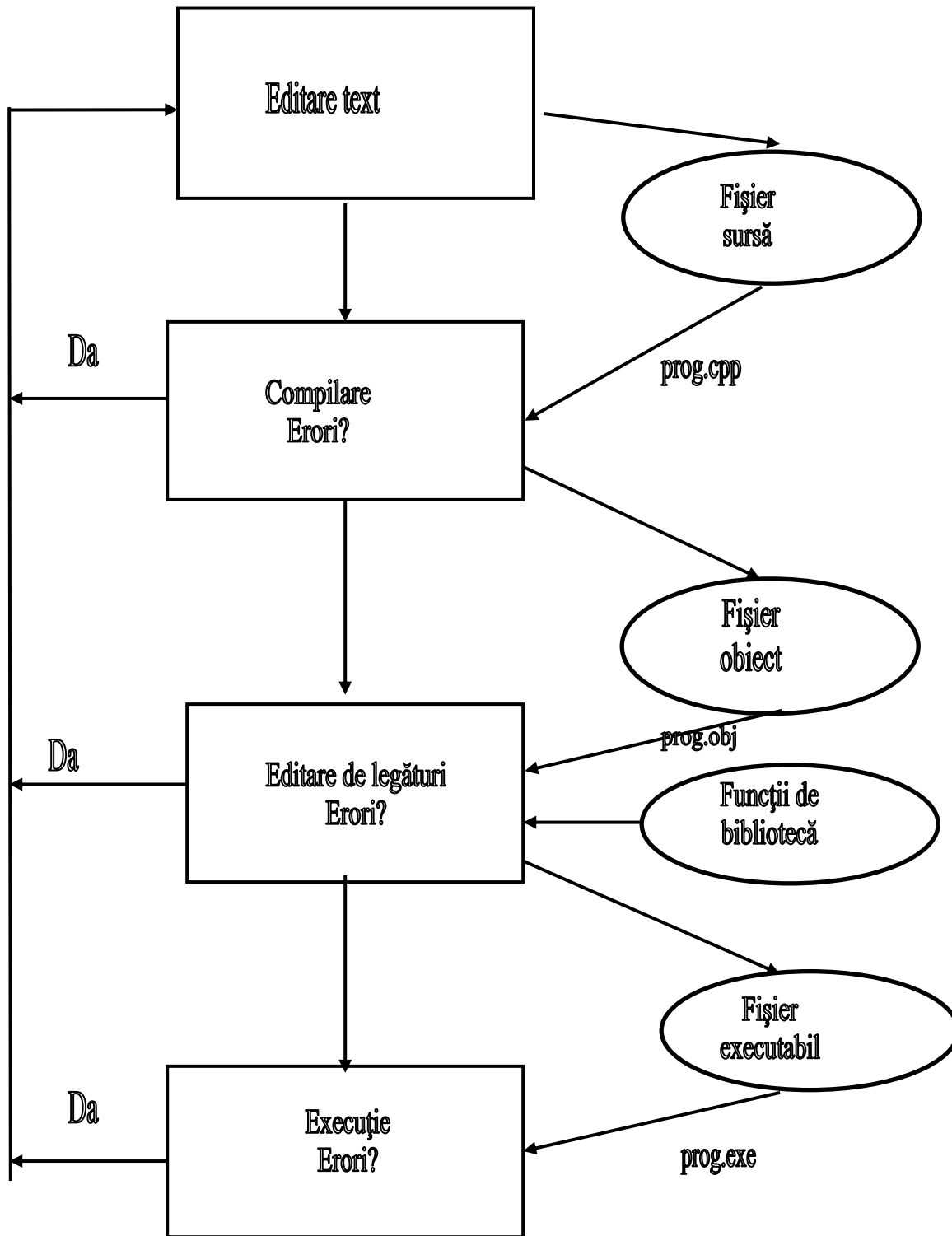



Fig.2.2. Etapele rezolvării unei probleme folosind calculatorul

5. Implementarea programului: editare, compilare, editare de legături, execuție.

Un mediu de programare C conține:

- I. *Editor de text* – folosit pentru crearea și modificarea codului sursă C
- II. *Compiler* – pentru a converti programul C în cod înțeles de calculator. Procesul de compilare cuprinde:
 - o fază de *precompilare (macroprocesare)* în care are loc *expandarea macrodefinițiilor și compilarea condițională și includerea fișierelor*
 - *compilarea propriu-zisă* în urma careia se generează cod obiect

Compilarea din linia de comandă în GCC se face cu:

```
gcc -c prog.c          prog.c → prog.o
```

- III. *Editor de legături* –leagă la codul obiect anumite *funcții de bibliotecă* (de exemplu intrări/ieșiri) extrase dintr-o bibliotecă de funcții, creindu-se un *program executabil*.

```
gcc -o prog prog.o    prog.o → prog.exe
```

Compilarea și editarea de legături se poate face printr-o singură comandă:

```
gcc -o prog prog.c    prog.c → prog.exe
```

Opțiunea `-o` permite specificarea fișierului de ieșire. Dacă acesta lipsește, se consideră în mod implicit ca nume al fișierului executabil `a.out`, Așadar:

```
gcc prog.c           prog.c → a.out
```

- IV. *Fișiere biblioteci de funcții* –sunt fișiere de funcții gata compilate care se adaugă (se leagă) la program.

Există biblioteci pentru: funcții matematice, șiruri de caractere, intrări/ieșiri.

Biblioteca standard C la execuție (run-time) este adăugată în mod automat la fiecare program; celelalte biblioteci trebuiesc legate în mod explicit.

De exemplu, în GNU C++ pentru a include biblioteca matematică `libm.so` se folosește opțiunea `-lm`:

```
gcc -o prog prog.c -lm
```

Bibliotecile pot fi:

- *statice* (`.lib` în BorlandC, `.a` în GCC) –codul întregii biblioteci este atașat programului
- *dinamice* (`.dll` în BorlandC, `.so` în GCC) –programului i se atașează numai funcțiile pe care acesta le solicită din bibliotecă.

În GCC se leagă în mod implicit versiunea dinamică a bibliotecii; pentru a lega versiunea statică se folosește opțiunea `-static`.

- V. *Fișiere antet (header files)*–datele și funcțiile conținute în biblioteci sunt declarate în fișiere antet asociate bibliotecilor. Prin includerea unui fișier antet compilatorul poate verifica corectitudinea apelurilor de funcții din biblioteca de funcții asociată (fără ca aceste funcții să fie disponibile în momentul compilării).

Tabel 2.1. Fișiere antet

Bibliotecă	Fișier antet
matematică	math.h
șiruri de caractere	string.h
intrări/ieșiri	stdio.h

Un fișier antet este inclus printr-o directivă cu sintaxa:

```
#include <nume.h>
```

care caută fișierul antet într-un director special de fișiere incluse (include), în timp ce directive:

```
#include "nume.h"
```

caută fișierul antet în directorul current.

Încărcarea și execuția programului (fișierului executabil .exe) se va face în GCC prin:

```
./ prog
```

6. Depanarea programului (debugging).

Depanarea unui program reprezintă localizarea și înlăturarea erorilor acestuia.

Cele mai frecvente *erori de sintaxă* se referă la: lipsa terminatorului de instrucțiune, neechilibrarea parantezelor, neînchiderea șirurilor de caractere, etc.

Erorile detectate de către *editorul de legături* sunt *referințele nerezolvate* (apelarea unor funcții care nu au fost definite, sau care se află în biblioteci care nu au fost incluse).

Cele mai des întâlnite erori la execuție provin din:

1. confuzia între = și ==
2. depășirea limitelor memoriei alocate tablourilor
3. variabile neinițializate
4. erori matematice: împărțire prin 0, depășire, radical dintr-un număr negativ, etc.

Dacă s-a depășit faza erorilor la execuție, vom testa programul, furnizându-i date de intrare pentru care cunoaștem ieșirile. Apariția unor neconcordanțe indică prezența unor *erori logice*. Dacă însă, rezultatele sunt corecte, nu avem certitudinea că programul funcționează corect în toate situațiile. Prin testare putem constata prezența erorilor, nu însă și absența lor.

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