CAPITOLUL 2

2. Limbajul C

Limbajele de programare de nivel mediu au fost serios dezvoltate pe la mijlocul anilor ’50. Ideile de bază folosite pentru proiectarea lor derivă din formulele funcționale λ-Church ce fuseseră propuse prin anii ’30. La ora actuală se estimează că există peste 2000 de limbaje de programare, diferențele între ele fiind legate în principal de stilul de programare.

Limbajul FORTRAN (FORmula TRANslator) a fost unul dintre primele limbaje de programare. El este folosit şi acum în industrie sau cercetare pentru a implementa aplicații specifice cu volume mari de calcule matematice complexe.

Pe lângă limbaje de programare pentru maşini secvențiale, s-au dezvoltat şi limbaje care suportă procesare paralelă. Din nefericire majoritatea nu sunt portabile, fiind orientate strict pe arhitectura maşinii pentru care a fost dezvoltat limbajul. De abia în ultimii zece ani dezvoltarea rapidă a rețelelor de calculatoare a condus şi la o încercare de standardizare a unor limbaje ce pot crea cod executabil simultan pe mai multe sisteme de calcul.

Avantajele programării la nivel inalt sunt:

- ascund diferențele de arhitectură internă
- sunt uşor de urmărit şi menținut (modificare şi depanare uşoară)
- sunt portabile . De exemplu un cod scris în C poate fi executat şi sub UNIX şi sub DOS
- nu au cel mai eficient cod, în comparație cu cel scris in ASM direct

De obicei sistemele de operare erau dezvoltate in limbaj de ansamblare pentru a fi mai rapide . Limbajul C, dezvoltat in 1972 este Dennis M.Retchie la Laboratoarele AT&T Bell, este primul limbaj pentru crearea sisteme de operare. Numele limbajului provine din faptul că este rezultatul îmbunătățirii limbajului B, folosit în scrierea UNIX pentru DEC PDP7. Prima documentație despre acest limbaj a fost “The C Programing Language”, scrisă de Retchie şi Brian Kernighan în 1977. Înaintea ei exista doar “The C Reference Manual”, scrisă de Retchie. O caracteristică importantă a acestui limbaj este faptul că poate fi considerat simultan şi un limbaj de nivel mediu şi un limbaj de nivel scăzut.

O dată cu răspândirea lui printre programatorii profesionişti s-a trecut şi la adaptarea compilatorului pentru sistemul de operare MS-DOS. De asemenea a apărut şi altă problemă şi anume faptul că standardul K&R nu mai era satisfăcător. De asemenea datorită variantelor de compilare pentru PC-uri C începuse să-şi piardă din portabilitate.

În acel moment Institutul Național pentru Standarde al Americii (ANSI), a creat un subcomitet pt. a defini o versiune oficială pt. limbajul C, rezultând astfel ANSI C.

Limbajul C şi versiunile sale OOP (Object Oriented Programming) C++ şi mai noul Visual C++ sunt printre cele mai folosite limbaje de programare la ora actuală. A fost proiectat pornind de la două limbaje dezvoltate spre sfârşitul anilor ’60: BCPL şi prima sa versiune B.

Un limbaj de nivel scăzut este orientat pe arhitectură şi poate fi definit ca având facilitatea de a scrie instrucțiuni ce se referă direct la elemente ale respectivei arhitecturi interne (ca în cazul limbajului de asamblare).

Un limbaj de nivel înalt sau mediu este orientat pe problemă permițând programatorului să-şi structureze programul cât mai apropiat posibil de problema ce trebuie implementată. Acest lucru scade şansele unei erori de proiectare, la nivel logic, al aplicației. Chiar dacă programul va fi mai ineficient din punct de vedere al vitezei şi al ocupării resurselor (prin comparație cu un program scris în limbaj de asamblare) el are avantajul că poate fi scris repede şi extins cu uşurință.

Limbajul C permite folosirea ambelor tehnici: programare structurată şi acces direct la maşină, fapt care-l face să fie foarte flexibil.

Ultimul şi poate cel mai important motiv pentru învățarea limbajului C este faptul că permite trecerea cu uşurință, la C++ sau Java.

Deşi limbajul **C** are o libertate foarte mare in scrierea codului, el acceptând multe forme de scriere care în alte limbaje nu sunt premise, acest lucru ascunde însă şi pericole, în special pentru programatorii fără experiență.

2.1 Structura generică a unui program scris în C.

De la primele programe scrise este bine să se țină cont de o serie de reguli privind organizarea codului sursă. Structura generală a unui cod va fi:

Apeluri ale directivelor speciale de compilare;
Declarații de biblioteci folosite;
Declarații funcții in-line;
Declarații ale prototipurilor funcțiilor personale;
Declarații variabile globale;
Corpul programului principal
{
declarații variabile;
apeluri de funcții;
}
Corpurile funcțiilor dezvoltate de programator;

Există câteva observații suplimentare privind tehnoredactarea codului sursă de care ar fi bine să se țină seama.

1. Să nu se scrie instrucțiunile una după alta ci una sub alta.
2. Toate instrucțiunile care formează un bloc pentru o altă instrucțiune să fie deplasate față de începutul respectivei instrucțiuni.

2.2. Etapele generării unui program executabil pornind de la o problemă reală

Una din greşelile începătorilor în programare constă în faptul că pun prea mult accent pe introducerea unor programe gata scrise, fără ca în prealabil să studieze problema de pornire precum şi algoritmul generat în urma analizei acestei probleme.
În momentul gândirii unui program, pe lângă alte considerente trebuie ținut cont şi de faptul că este o mare diferență între programele **bune** si cele **care merg.** Un program bun nu numai că merge, dar este şi uşor de înțeles şi menținut. Mulți dintre începătorii în

programarea C-ului au tendința de a se limita la programe care să meargă, şi acest fapt conduce la deprinderi nesănătoase.

De aceea mai jos vom prezenta paşii necesari a fi urmați în elaborarea unei aplicații pornind de la o problemă reală.

1. Înțelegerea problemei.
2. Realizarea modelului matematic, dacă este cazul.
3. Verificarea modelului matematic cu un set de date real sau calculate manual. Dacă se observă neconcordanțe se va relua verificarea punctelor anterioare, dacă nu se continuă.
4. Identificarea datelor de intrare, a datelor de ieşire precum şi a variabilelor intermediare.
5. Alegerea tipurilor de date folosite pentru reținerea în cursul procesării a respectivelor date.
6. Specificarea modului de interfațare cu utilizatorul la nivel de intrare sau ieşire.
7. Generarea schemei logice sau a pseudocodului ce descrie algoritmul propus pentru rezolvarea problemei.
8. Verificarea modelului creat la punctul 7 cu un set date reale sau calculate manual. În caz de neconcordanțe se reia verificarea punctelor anterioare, dacă nu se continuă.
9. Scrierea codului sursă (transpunerea într-un limbaj de programare oarecare).
10. Eliminarea erorilor de sintaxă.

Pentru a exemplifica paşii menționați anterior vom lua un exemplu extrem de simplu.
"Să se realizeze un program care rezovă o ecuație oarecare de gradul II"

Pasul 1
În acest caz este simplu, e clar că ne referim la o ecuație de foma $ax^2+bx+c=0$;

Pasul 2
Modelul matematic este cel studiat în liceu:
Fie $\Delta = b^2 - 4 \cdot a \cdot c$ atunci :
Dacă $\Delta>0$ atunci: $x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}$.
Dacă $\Delta<0$ atunci: $x_{1,2} = \frac{-b \pm i \cdot \sqrt{-b^2 + 4 \cdot a \cdot c}}{2 \cdot a}$.
Dacă $\Delta=0$ atunci: $x_{1,2} = -\frac{b}{2 \cdot a}$.

Pasul 3
În acest caz poate fi evitat modelul matematic este prea simplu.

Pasul 4
Să vedem deci care ar fi datele ce trebuiesc furnizate de utilizator. Având în vedere că se doreşte rezolvarea unei ecuații generice de ordinul II este clar că :
- a,b,c date de intrare

- $x_{1,2}$ date de ieşire
- Δ dată intermediară.

Analizând modelul matematic observăm că a trebuit să folosim un simbol suplimentar funcție de care se face întreaga discuție. Acest simbol nu este nici dat de utilizator şi nici nu interesează pentru rezultatul final, deci Δ este o dată intermediară de calcul.

Pasul 5

Acum trebuie să specificăm tipul de dată folosit pentru reprezentarea internă (în memoria calculatorului) a variabilelor. Deci avem câteva date suplimentare la dispoziție pentru a vedea tipul de reprezentare care trebuie ales:
- specificațiile utilizatorului
- modelul matematic
- experiența în programare

În acest caz modelul matematic ne spune că a,b,c sunt numere reale. Fără o specificare clară a utilizatorului care să îndice că ele aparțin altui interval este obligatoriu ca să luăm în considerare modelul cel mai extins, deci cel mai adaptabil. Desigur că nu vom alege cel mai cuprinzător tip de date pus la dispoziție de către compilator în lipsa unor specificații externe. Deci a,b,c vor fi de tipul **float**.

Variabila intermediară Δ este rezultat al aplicării unei funcții lineare asupra unor variabile de tip real deci şi delta va trebui să fie **float**.

Datele de ieşire $x_{1,2}$, după cum se observă sunt rezultatul unei înpărțiri şi deci obligatoriu sunt reale indiferent de tipul de dată ales pentru datele de intrare.

Pasul 6

În lipsa unor specificații de la utilizator vom alege pentru citirea datelor cea mai simplă metodă citirea directă după cum am făcut şi până acum. Pentru afişare însă avem două metode afişarea de tipul Re(x) şi Im(x) sau direct rezultatul ca în calculul manual. Vom alege cea de a doua metodă.

Pasul 7

Pseudocodul va arăta ca mai jos.

citeşte a
citeşte b
citeşte c
*calculează delta după formulă delta=b*b-4*a*c;*
dacă delta >=0 atunci
{
*x_1=(-b+sqrt(delta))/(2*a);*
*x_2=(-b-sqrt(delta))/(2*a);*
afiafişează x1,x2
}
altfel
{
delta=-delta;
*delta=sqrt(delta)/(2*a)*

*afişează x1 de forma re+i*im*
*afişează x2 de forma re-i*im*
}

Câteva observații relative la construcția pseudocodului

- se observă că nu am tratat separat situația delta=0 pentru că zero la adunare nu modifică rezultatul
- în a doua parte am folosit un artificiu de calcul în sensul că am păstrat tot în variabila delta rezultate parțiale de calcul. Altfel ar fi trebuit să introducem variabile suplimentare pentru păstrarea părții întregi şi a părții reale. Codul ar fi fost mai clar dar în acelaşi timp am fi folosit mai multă memorie. Dacă în cazul acestui program câteva variabile suplimentare nu ne deranjează, în cazul unor programe foarte mari acest lucru poate ridica greutăți ce nu pot fi eliminate decât prin trecerea la alocarea dinamică a memoriei. Această metodă va fi prezentată ulterior.
- se observă că am scris mult cod suplimentar care din punct de vedere al calculului brut nu ne trebuie, dar din punct de vedere al modului în care îi prezint rezultatul uni utilizator este esențial pentru că este exact în forma matematică cunoscută.

Pasul 8

Va fi evitat pentru acest exemplu deoarece este prea simplu.

Pasul 9

Codul în cazul limbajului C va araăta ca mai jos:

```
        #include <stdio.h>
*       #inlude <conio.h> // aici eroare de sintaxă detectabilă de interpretor

        void main(void)
        {
         float a,b,c,delta,x1,x2;

         clrscr()
         printf("Pentru calculul ecuatiei a*x*x+b*x*c=0 dati coeficientii");
        printf("\na=");
**      scanf("%f",a); //aici eroare de sintaxă nedetectabilă de interpretor
        printf("\nb=");
        scanf("%f",&b);
        printf("\nb=");
        scanf("%f",&b);
***     delta=b*b+4*a*c; // aici eroare de calcul
         if( delta >=0)
         {
         x1=(-b+sqrt(delta))/(2*a);
         x2=(-b-sqrt(delta))/(2*a);
         printf("\nx1=%f",x1)
         printf("\nx2=%f",x2)
                }
```

```
else
{
        delta=-delta;
delta=sqrt(delta)/(2*a)
printf("x1=%f+i*%f",-b/(2*a),delta);
printf("x2=%f-i*%f",-b/(2*a),delta);
}
}
```

Pasul 10

După cum se observă am introdus intenționat trei tipuri de greşeli. Prima, marcată cu o stea este greşeală de sintaxă ce va fi detectată de compilator şi deci o putem corecta în această fază.

Pasul 11

În cadrul rulării programului rezultat se observă că acesta nu funcționează corect. La execuția pas cu pas se va observa că în variabila a nu se depune ceea ce se citeşte de la tastatură. Este o greşeală de logică de programare, pentru că **scanf** aşteaptă un pointer pentru a returna corect rezultatul, iar a nu este declarat pointer trebuie transmisă adresa lui a. Această eroare nu poate fi detectată de interpretor.

De asemenea cu trei stele am marcat o greşeală de transpunere a algoritmului, o greşeală simplă de transcriere a unei formule de calcul. Deabia după eliminarea acestor erori se poate continua testarea programului.

Este bine ca să existe un comentariu la începutul oricărei sursei C având următorul conținut

```
/*----autor:
------data creării/modificării
------copyright
*/
```