4. Intrările în program. Scrierea aplicațiilor

Un program are nevoie de date pentru a opera. Până acum am scris programe în care valorile datelor se găsesc chiar în program, în constante simbolice sau literale. Dacă dorim să modificăm o dată, trebuie să facem o mică modificare în program, să îl recompilăm şi să îl executăm din nou. În acest capitol vom vedea cum putem introduce date în program chiar în timpul rulării lui.

Odată ce am aflat cum să introducem date în program, să procesăm datele şi să afişăm rezultatele, putem să ne gândim la scrierea unor programe mai complicate. Pentru aceasta avem nevoie de o abordare ceva mai organizată. În finalul capitolului vom discuta despre descompunerea în module şi despre programarea orientată pe obiecte.

4.1 Transmiterea datelor către programe

Unul dintre marile avantaje ale calculatorului este că un program poate fi folosit cu diverse seturi de date. Pentru aceasta trebuie să separăm datele de program până în momentul execuției. Anumite instrucțiuni din program copiază valori în variabilele din program. După stocarea acestor valori în variabile, programul poate să le folosească în calcule.

Procesul de plasare a unor valori dintr-o mulțime de date în variabile din program se numeşte *intrare* (*input*). Într-o terminologie mai largă, calculatorul se spune că *citeşte* date din exterior în variabile. Datele pot proveni dintr-un fişier, de la tastatură etc. Dispozitivul standard de intrare este tastatura.

Stream-uri de intrare şi operatorul de extracție >>

În C++, conceptul de stream este esențial. Putem gândi un *stream de ieşire* ca pe o secvență infinită de date care circulă dinspre program înspre un dispozitiv de ieşire. Un *stream de intrare* este o secvență infinită de caractere care porneşte de la un dispozitiv de intrare şi este dirijată către program.

Fişierul header `iostream` conține, printre altele, definițiile a două tipuri de date: `istream` şi `ostream`. Aceste tipuri de date reprezintă stream-uri de intrare şi stream-uri de ieşire. Acest fişier header mai conține două declarații care arată aproximativ astfel:

```
istream cin;
ostream cout;
```

Cele două declarații arată că `cin` este un obiect de tip `istream` şi `cout` este un obiect de tip `ostream`. În mod explicit, cin este asociat cu tastatura, iar cout cu display-ul.

Am văzut că trimiterea unor valori către `cout` se face folosind *operatorul de inserție* <<:

```
cout << 3 * pret;
```

Similar, putem citi date din `cin` folosind *operatorul de extracție* >>:

```
cin >> cost;
```

Operatorul de extracție >> are doi operanzi. În stânga se găseşte un stream, în cel mai simplu caz `cin`, iar în dreapta se găseşte o variabilă având un tip predefinit (`char`, `int`, `float` etc.).

Putem folosi operatorul de mai multe ori într-o instrucțiune:

```
cin >> lungime >> latime;
```

fiind echivalentă cu:

```
cin >> lungime;
cin >> latime;
```

Trebuie să fim atenți atunci când folosim cei doi operatori, pentru că `cin` poate fi folosit doar în combinație cu >>, iar `cout` doar cu <<.

Dacă într-o instrucțiune de afişare putem folosi constante, variabile şi chiar expresii foarte complicate, într-o instrucțiune de intrare nu putem folosi decât nume de variabile. Aceasta pentru că o instrucțiune de intrare trebuie să precizeze clar unde se stochează valoarea unei date de intrare. Doar numele de variabile referă locații de memorie ale căror valori pot fi modificate în timpul execuției unui program.

Atunci când introducem o dată de la tastatură, trebuie să ne asigurăm că tipul introdus şi cel aşteptat de program se potrivesc. Un număr întreg este forțat automat la un număr real. Operația inversă, însă, poate conduce la rezultate eronate.

Atunci când extrage valori dintr-un stream, operatorul >> ignoră orice spațiu de la început. De asemenea, ignoră caracterul care marchează sfârşitul liniei. Apoi, operatorul >> procedează la extragerea valorilor din stream-ul de intrare. Dacă data aşteptată este un `char`, intrarea se întrerupe după primul caracter. Dacă este vorba de un `int` sau `double`, intrarea se întrerupe la primul caracter care nu se potriveşte ca tip de dată, cum ar fi un spațiu.

Exemplu

```
int i, j, k;
char ch;
float x;
```

Instrucțiunea	**Data**	**Conținutul variabilei după intrare**
`cin >> i;`	`32`	`i = 32`

`cin >> i >> ch >> x;`	`25 A 16.9`	`i = 25`
		`ch = 'A'`
		`x = 16.9`

`cin >> i >> j >> x;`	`12 8`	`i = 12`
		`j = 8`
		Programul aşteaptă al treilea număr

`cin >> i >> x;`	`46 32.4 15`	`i = 46`
		`x = 32.4`
		15 este păstrat pentru o intrare ulterioară

Caracterul *newline*

Fiecare linie are un caracter invizibil care marchează sfârşitul său – caracterul *newline*. Pentru a determina următoarea valoare de intrare, operatorul >> trece peste caracterul *newline*, în cazul în care acesta există.

Atunci când utilizăm tastatura, caracterul *newline* este introdus prin apăsarea lui Return sau Enter. Programul poate genera un *newline* folosind manipulatorul `endl` într-o instrucțiune de ieşire. În C++ putem să ne referim la caracterul *newline* folosind simbolurile `\n`. Deşi `\n` constă din două caractere, el se referă la unul singur – caracterul *newline*. Aşa cum putem păstra litera `A` în variabila `ch` de tip `char` prin instrucțiunea

```
char ch = 'A';
```

tot aşa putem păstra caracterul *newline* într-o variabilă:

```
ch = '\n';
```

Exemplu

Când avem o secvență de citiri, putem introduce valorile în mai multe feluri:

`cin >> i;`	`25 A 16.9\n`	`25\n`	`25A16.9\n`
`cin >> ch;`		`A\n`	Citirea se opreşte când tipul de dată nu mai corespunde
`cin >> x;`		`16.9\n`	

După citire, variabilele vor avea următoarele valori:

```
i = 25
ch = 'A'
x = 16.9
```

Citirea datelor de tip caracter folosind instrucțiunea `get`

Am văzut că operatorul >> ignoră orice spațiu apărut în stream-ul de intrare. Să presupunem că `ch1` şi `ch2` sunt variabile de tip `char` şi că programul execută instrucțiunea

```
cin >> ch1 >> ch2;
```

Dacă stream-ul de intrare este

```
R 1
```

Atunci operatorul de extracție va stoca `R` în `ch1`, ignoră spațiul şi apoi păstrează `1` în `ch2`.

Ce se întâmplă, însă, dacă am fi dorit, de fapt, să introducem trei caractere: `R`, spațiu şi `1`? Cu operatorul de extracție acest lucru nu este posibil.

Vom folosi funcția `get` care încarcă următorul caracter fără a ignora spațiile. Acest apel de funcție arată astfel:

```
cin.get(ch);
```

Specificăm numele `istream`-ului, adică `cin`, apoi punem semnul . (punct) urmat de numele funcției şi lista ei de parametri. Apelul funcției `get` foloseşte sintaxa apelului funcțiilor `void` şi nu a celor care întorc o valoare. Acest apel de funcție este, deci, o instrucțiune de sine stătătoare. Parametrul funcției `get` trebuie să fie o variabilă. Acesta este un exemplu prin care se apelează o funcție din clasa `istream` – funcția `get` – pentru un obiect al acestei clase – obiectul `cin`.

Putem folosi următoarele trei apeluri ale lui `get`:

```
cin.get(ch1);
cin.get(ch2);
cin.get(ch3);
```

sau

```
cin >> ch1;
cin.get(ch2);
cin >> ch3;
```

Dacă stream-ul de intrare este tot

```
R 1
```

atunci variabilele `ch1`, `ch2` şi `ch3` vor stoca

```
ch1 = 'R';
ch2 = ' ';
ch3 = '1';
```

Exemplu

```
cin >> ch1;    A B\n          ch1 = 'A';
cin >> ch2;    CD\n           ch2 = 'B';
cin >> ch3;                   ch3 = 'C';

cin.get(ch1);  A B\n          ch1 = 'A';
cin.get(ch2);  CD\n           ch2 = ' ';
cin.get(ch3);                 ch3 = 'B';

cin >> ch1;    A B\n          ch1 = 'A';
cin >> ch2;    CD\n           ch2 = 'B';
cin.get(ch3);                 ch3 = '\n';
```

Ignorarea caracterelor folosind funcția `ignore`

Funcția `ignore` este asociată cu tipul de dată `istream`, fiind o funcție definită în această clasa `istream`. Este folosită pentru a „sări” peste caractere din stream-ul de intrare. Este o funcție cu doi parametri, iar un exemplu de apel este următorul:

```
cin.ignore(200, '\n');
```

Primul parametru este o expresie `int`, iar al doilea una `char`.

Acest apel al funcției spune calculatorului să ignore următoarele 200 de caractere de intrare sau să sară până întâlneşte caracterul *newline*, în funcție de care dintre ele apare prima.

Exemplu

```
cin >> i >> j;           957 34 1235\n     i = 957;
cin.ignore(100, '\n');   128 96\n          j = 34;
cin >> k;                                  k = 128;

cin >> ch;               A 22 B 16 C 19\n  ch = 'A';
cin.ignore(100, 'B');                      i = 16;
cin >> i;

cin.ignore(2, '\n');     ABCDEF\n          ch = 'C';
cin >> ch;
```

Intrări şi ieşiri interactive

Un *program interactiv* este unul prin care utilizatorul comunică direct cu calculatorul. Multe dintre programele scrise de noi vor fi interactive, dând şi o anumită etichetă codului.

Atunci când dorim să cerem utilizatorului să introducă o dată în program, este util să îi afişăm înainte un mesaj prin care să îi explicăm ce trebuie să introducă. De asemenea, programul ar trebui să tipărească toate datele introduse pentru ca utilizatorul să se poată verifica. Aceasta se numeşte *tipărire în ecou*.

Programul de mai jos arată cum poate fi scris un cod interactiv.

Exemplu

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

int main()
{
     int codNumeric;
```

```
        int cantitate;
        double pretUnitar;
        double pretTotal;

        cout << "Introduceti codul numeric al produsului
    comandat:" << endl;
        cin >> codNumeric;

        cout << "Introduceti cantitatea:" << endl;
        cin >> cantitate;

        cout << "Introduceti pretul unitar pentru acest
    produs:" << endl;
        cin >> pretUnitar;

        pretTotal = cantitate * pretUnitar;

        cout.setf(ios::fixed, ios::floatfield);

        cout << "Produsul " << codNumeric
             << ", cantitatea " << cantitate
             << ", pretul unitar " << setprecision(2) <<
    pretUnitar
             << " lei " << endl;
        cout << "Total: " << pretTotal << endl;

        return 0;
    }
```

În timpul rulării acestui program se va tipări câte un text care va arăta utilizatorului care este următoarea valoare aşteptată.

Exemplu

```
Introduceti codul numeric al produsului comandat
4671
Introduceti cantitatea:
10
Introduceti pretul unitar pentru acest produs:
272.55
Produsul 4671, cantitatea 10, pretul unitar 272.55 lei
Total: 2725.50
```

Volumul informației afişate depinde de cel care va folosi programul. Dacă el este destinat unor persoane nefamiliarizate cu calculatorul, mesajele vor fi mai detaliate. Dacă programul este folosit frecvent de aceeaşi persoană, putem da mesaje mai scurte sau se pot introduce mai multe valori pe aceeaşi linie.

Intrări şi ieşiri neinteractive

Deşi tindem să dăm exemple de programe interactive, multe dintre programele folosite în viața reală sunt neinteractive. Acestea sunt programe care prelucrează cantități mari de date, greu de introdus interactiv fără erori. Pentru acest tip de programe, datele se păstrează în fişiere de date pregătite anterior. Aceasta permite verificare şi corectarea datelor înainte de rularea programului.

4.2 Intrări şi ieşiri din fişiere

Fişierul este o zonă pe un suport de stocare, de exemplu hard disc, desemnată printr-un nume, care păstrează o colecție de date, de exemplu codul programului scris cu un editor.

Un program poate citi datele dintr-un fişier în aceeaşi manieră în care le citeşte de la tastatură. Se poate scrie în fişier la fel cum se scrie pe ecran.

Atunci când programul ajunge să lucreze cu cantități mari de date, este de preferat să folosim fişierele. Acestea pot fi scrise cu un editor de texte, corectate şi salvate pe disc. Totodată, nu suntem obligați să introducem toate datele dintr-o singură dată.

Modul de utilizare a fişierelor

Pentru a folosi un fişier în operațiile I/O trebuie să parcurgem patru paşi.

1. Cerem preprocesorului să includă fişierul header `fstream`;
2. Folosim instrucțiuni de declarare pentru a declara stream-urile;
3. Pregătim fişierele pentru citire şi scriere prin instrucțiunea `open`;
4. Specificăm numele stream-ului în fiecare instrucțiune de citire sau de scriere.

Includerea fişierului header `fstream`

Vom modifica programul care măsoară consumul unui autoturism la 100 km pentru ca să citească datele dintr-un fişier.

Exemplu

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

int main()
{
     float cant1;
     float cant2;
     float cant3;
     float cant4;
     float indicatiePlecare;
     float indicatieSosire;
     float litriPerKm;

     ifstream inConsum;
     ofstream outConsum;

     inConsum.open("incons.dat");
     outConsum.open("outcons.dat");

     inConsum >> cant1 >> cant2 >> cant3 >> cant4
              >> indicatiePlecare >> indicatieSosire;

     litriPerKm = (cant1 + cant2 + cant3 + cant4)*100.0/
                  (indicatieSosire - indicatiePlecare);

     outConsum << "Consumul este " << litriPerKm
               << " litri per km." << endl;
```

```
  return 0;
}
```

Mai întâi folosim directiva de preprocesare

```
#include <fstream>
```

Prin acest header se definesc două noi tipuri de date, `ifstream` şi `ofstream`. Cele două tipuri de dată reprezintă, primul, un stream de caractere provenind de la un fişier, iar al doilea un stream de caractere care conduce către un fişier. Toate operațiile valabile pentru un `istream`: `>>`, `get` sau `ignore` sunt valabile şi pentru tipul `ifstream`. Operațiile folosite pentru un `ostream`: `<<`, `endl`, `setw`, `setprecision` se pot folosi şi pentru un `ofstream`.

Declararea stream-urilor pentru fişiere

Obiectele de tip stream se declară la fel ca orice variabilă

Exemplu

```
int unInt;
float unFloat;
ifstream unFisier;
ofstream altFisier;
```

Obiectele `cin` şi `cout` nu trebuie declarate în fiecare program pentru că ele sunt declarate în fişierul `iostream`, citirile de la tastatură şi scrierile pe ecran fiind operații frecvente. Spre deosebire de ele, stream-urile pentru lucrul cu fişiere trebuie declarate în program pentru că fiecare aplicație foloseşte propriile fişiere de date. Pentru programul nostru declarăm două stream-uri:

```
ifstream inConsum;
ofstream outConsum;
```

De notat că `ifstream` se utilizează doar pentru fişiere de intrare, iar `ofstream` doar pentru fişiere de ieşire. Prin intermediul unui obiect `ifstream` se pot face doar citiri, iar prin intermediul unui obiect `ofstream` se pot face doar scrieri.

Deschiderea fişierelor

Trebuie să pregătim acum fişierele pentru citire sau scriere, deci le *deschidem*.

Ne propunem să citim din stream-ul tip fişier `inConsum` şi să scriem în stream-ul `outConsum`. Deschidem fişierele folosind următoarele instrucțiuni:

```
inConsum.open("incons.dat");
outConsum.open("outcons.dat");
```

Funcția `open` are un singur argument cuprins între ghilimele. Prima instrucțiune este un apel al funcției `open` asociate cu tipul de dată `ifstream`, iar a doua apelează funcția `open` asociată cu `ofstream`.

Funcția `open` asociază variabila stream din program cu un fişier fizic pe disc. Prima instrucțiune realizează o conexiune între obiectul `inConsum` şi fişierul `incons.dat`. A doua instrucțiune leagă obiectul `outConsum` de fişierul `outcons.dat`. `inConsum` este conectat cu `incons.dat` aşa cum `cin` este legat de tastatură.

Mai departe, acțiunea depinde de tipul de stream.

Pentru un fişier de intrare, funcția `open` poziționează *markerul de citire* pe primul element din fişier.

Pentru un fişier de ieşire, funcția verifică dacă acesta există. Dacă există, şterge vechiul conținut al său. Dacă nu există, îl creează. Apoi *markerul de scriere* este aşezat pe prima poziție.

Folosim noțiunile de marker de citire sau de scriere pentru a reprezenta locul din care se va citi dintr-un stream de intrare, respectiv se va scrie într-un stream de ieşire.

Exemplu

```
inConsum                    outConsum
23.2                        _
17.4
19.8
16.7
22451
23544
```

Deschiderea fişierelor trebuie realizată înaintea oricărei utilizări a lor deoarece funcția `open` le pregăteşte pentru citire sau scriere.

Specificarea stream-urilor în instrucțiuni I/O

Pentru citirea şi scrierea din fişier nu va trebui decât să înlocuim obiectele `cin` şi `cout` cu obiectele de tip stream de fişier declarate mai devreme.

Exemplu

```
inConsum >> cant1 >> cant2 >> cant3 >> cant4
         >> indicatiePlecare >> indicatieSosire;
outConsum << "Consumul este " << litriPerKm
          << " litri per km." << endl;
```

Cel mai interesant este că C++ foloseşte o sintaxă uniformă pentru operațiile I/O, indiferent dacă este vorba despre fişiere sau dispozitive I/O.

4.3 Erori de citire

La citirea datelor de la tastatură sau dintr-un fişier pot apărea erori.

Să presupunem că programul ne cere să introducem un număr întreg, iar noi introducem caractere. Operația de intrare eşuează datorită datelor de intrare invalide. În terminologia C++ stream-ul `cin` intră într-o stare de eroare - *fail state*. Dacă un stream intră într-o astfel de stare, orice altă operație ulterioară asupra sa este anulată. Din păcate, calculatorul nu opreşte programul şi nu dă niciun mesaj de eroare în astfel de situații.

De cele mai multe ori erorile de intrare apar din cauza nepotrivirii tipurilor de dată.

Exemplu

```
int i = 10;
int j = 20;
int k = 30;
cin >> i >> j >> k;
cout << “i: “ << i << “j: “ << j << “k: “ << k;
```

Dacă tastăm

```
1234.56 7 89
```

programul afişează

```
i: 1234 j: 20 k: 30
```

Un alt motiv pentru care un stream intră în *fail state* este incercarea de deschidere a unui fişier de intrare care nu există. Să presupunem că avem pe disc fişierul `myfile.dat` şi scriem următoarele instrucțiuni care îşi propun să lucreze cu acest fişier:

```
ifstream inFisier;
inFisier.open(”myfil.dat”);
```

```
inFisier >> i >> j >> k;
```

Datorită scrierii incorecte a numelui de fişier, `inFisier` va intra în fail state. Ca urmare, variabilele `i`, `j` şi k vor avea nişte valori nedeteminate.

Un stream de citire dintr-un fişier intră în *fail state* atunci când se citeşte din fişier dincolo de caracterul `EOF`, cel care marchează finalul fişierului.

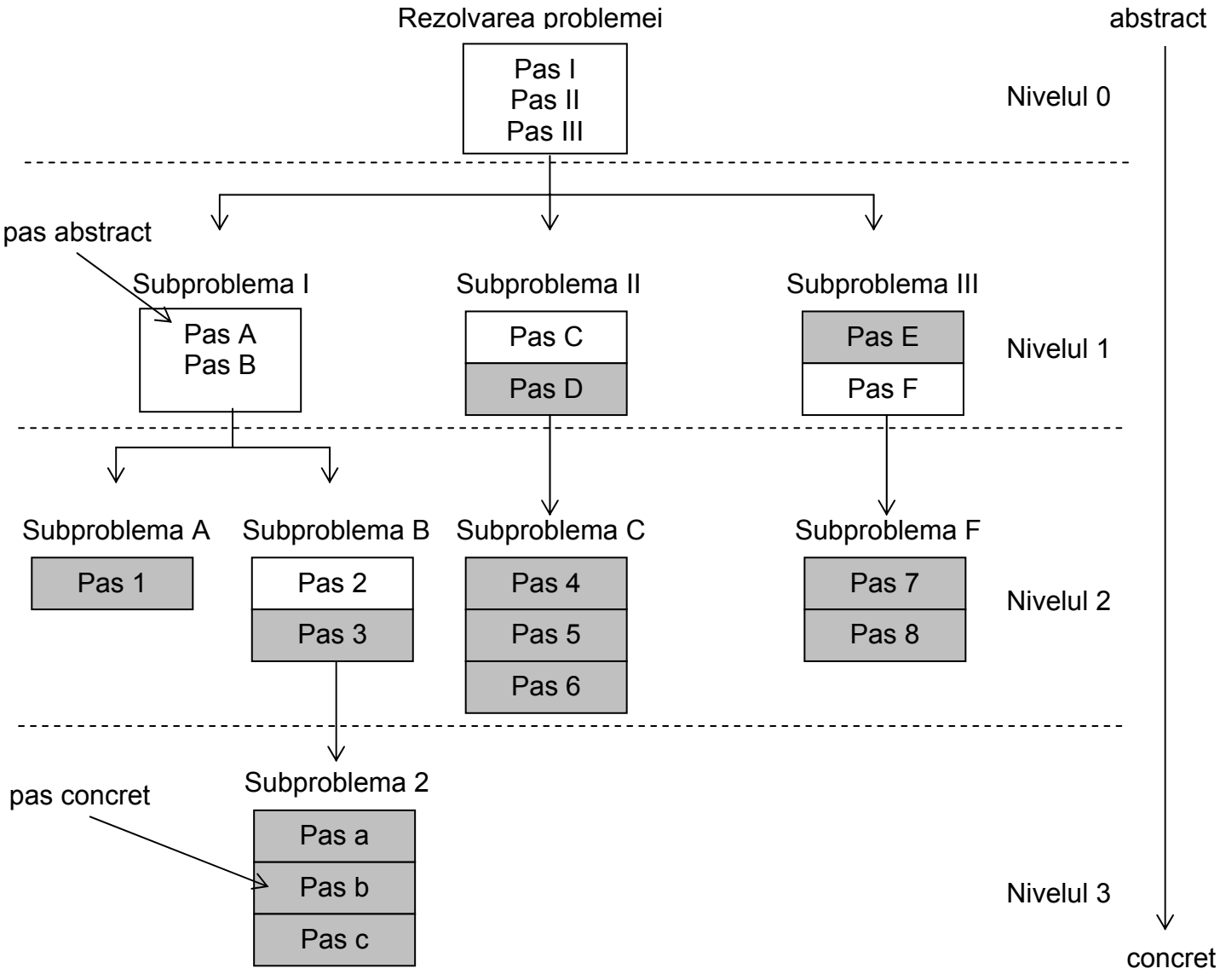
4.4 Scrierea aplicațiilor

Problemele pe care le-am prezentat până acum au fost simple şi uşor de programat. În acest moment putem scrie şi aplicații mai complexe şi de aceea trebuie să vedem cum putem concepe corect o aplicație. Vom vorbi despre descompunerea funcțională şi despre proiectarea orientată pe obiecte.

Descompunerea funcțională

Această este o tehnică de dezvoltare a unei părți a unui program sau chiar a unui program de dimensiuni reduse prin care problema este împărțită în subprobleme mai uşor de rezolvat, soluții care creează o soluție globală a întregii probleme.

Printr-o astfel de descompunere creăm o structură ierarhică numită *structură arborescentă*.



Paşii haşurați conțin suficiente detalii pentru a putea fi implementați în C++. Cei nehaşurați trebuie descompuşi în continuare. Fiecare celulă este un modul. Modulele sunt elemente de bază într-o descompunere funcțională.

Pentru conceperea unui modul trebuie să parcurgem următorii paşi:

1. Să schițăm o soluție a problemei
2. Să descriem paşii majori
3. Dacă un pas este suficient de simplu pentru a putea fi implementat, nu mai necesită descompuneri ulterioare
4. Dacă pasul poate fi gândit ca o serie de paşi mai mici, este încă un pas abstract

Proiectarea orientată pe obiecte

Descompunerea funcțională poate fi privită ca o metodă de găsire a soluției unei probleme cu accent pe algoritmi şi acțiunile care trebuie realizate. Datele, în acest caz, joacă un rol secundar.

Proiectarea orientată pe obiecte se focalizează pe entități (obiecte) şi operațiile posibile asupra acestor entități.

Exemplu

O problemă bancară poate avea nevoie de un obiect `contBancar` cu operațiile asociate `DeschideCont`, `ScrieCec` şi `CreeazaDepozit`. Obiectul `contBancar` constă din date – `numarCont` şi `balantaCurenta`.

Primul pas în proiectarea orientată pe obiecte este identificarea obiectelor din problemă şi a operațiilor asociate. Soluția finală va fi exprimată în termeni de obiecte şi operații. Datele joacă aici un rol determinant.

În C++ operațiile asociate cu o clasă sunt scrise ca funcții şi se numesc *funcții membre*. O funcție membră este apelată prin numele unui obiect al clasei urmat de un punct şi de numele funcției cu lista de parametri.

Exemplu

```
contBancar.DeschideCont(1000, "tip1");
```

Datele care compun obiectul se numesc *date membre*. Instanțele unei clase se numesc *obiecte*, în timp ce instanțele tipurilor de date predefinite cum ar fi `int` se numesc variabile.

Proiectarea orientată pe obiecte conduce la un program care foloseşte o colecție de obiecte. Fiecare obiect răspunde de o parte din soluție, iar obiectele comunică între ele prin apelarea funcțiilor membre.

Proiectarea orientată pe obiecte se pretează la scrierea proiectelor mari din următoarele trei motive:

1. Obiectele dintr-un program modelează obiecte din problema de rezolvat;
2. Este posibilă furnizarea şi utilizarea de biblioteci de clase scrise de diverse firme sau persoane independente;
3. Se foloseşte un concept fundamental numit *moştenire* care permite adaptarea unei clase la particularitățile problemei fără a modifica codul scris anterior.

Pentru crearea unei soluții software optime se urmează un proces detaliat pentru obținerea unei *analize* a *cerințelor sistemului* (*requirments*) şi *proiectarea* acestuia (*design*) astfel încât să satisfacă cerințele. Programatorii experimentați cunosc faptul că oricât de simplă ar fi problema pe care o au de rezolvat, timpul

petrecut cu analiza şi proiectarea poate salva foarte mult timp care se poate pierde cu dezvoltarea unui sistem greşit conceput.

Unified Modeling Language (UML)

Există multe procedee pentru analiza şi proiectarea orientată pe obiecte. Pentru totate acestea se foloseşte un limbaj grafic de comunicare a rezultatelor numit *Unified Modeling Language* (*UML*). Prima versiune a acestui limbaj a fost lansată în anul 1996 de către James Rumbaugh, Grady Booch şi Ivar Jacobson de la Rational Software Corporation. În acelaşi timp, organizația non-profit *Object Management Group* (*OMG*) a lansat, la inițiativa companiilor HP, IBM, Microsoft, Oracle şi Rational Software, o propunere de creare a unui limbaj unic de modelare. UML a fost limbajul adoptat de OMG care, începând cu anul 1997 asigură revizuirea permanentă a sa.

UML este o schemă de reprezentare grafică pentru modelarea sistemelor orientate pe obiecte. Una dintre cele mai atractive caracteristici ale sale este flexibilitatea. UML este extensibil şi independent de multele procese de analiză şi proiectare orientate pe obiecte. Tehnologia obiectelor a devenit indispensabilă în industria software, iar UML este din ce în ce mail folosit. Specificațiile complete ale UML sunt disponibile la `http://www.uml.org`.