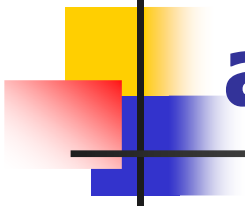
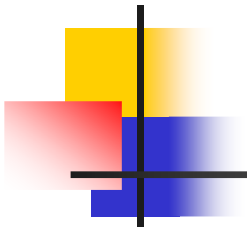
Proiectarea şi dezvoltarea sistematică a programelor de mari dimensiuni



1. **Programarea calculatoarelor – de la teorie la practică**
2. **Stilul de programare**
3. **Metoda detalierilor în paşi succesivi**
4. **Exemplu de program**
5. **Concluzii**



Programarea calculatoarelor
de la teorie la practică

- Predarea disciplinelor de programare se bazează în foarte mare măsură pe exemple de programe.

- Succesul unui curs de programare depinde, în final, şi de modul în care au fost alese exemplele din acel curs.

- Majoritatea exemplelor prezintă studenților programul direct în forma lui finală, trecând peste detaliile de realizare ale acestuia.

Programarea calculatoarelor
de la teorie la practică

- Activitatea de programare nu se restrânge la contemplarea unor programe gata făcute, ci presupune conceperea de noi programe.

- Este greşită impresia că programarea necesită doar cunoaşterea unui limbaj de programare, trecerea de la idei la program realizându-se pe bază de intuiție.

- Un curs de programare adevărat trebuie să prezinte metode de proiectare şi implementare, iar exemplele selectate trebuie să ilustreze un stil de programare şi să scoată în evidență stadiile prin care trece un program în procesul dezvoltării sale.

Stilul de programare
Elemente caracteristice

Unul dintre elementele care contribuie la realizarea eficientă a programelor de calitate este **stilul de programare**.

Stilul de programare se poate caracteriza prin :

- **aspectul general** al programului;
- **claritatea** şi **lizibilitatea** programului;
- **structurarea** şi **modularizarea** programului în conformitate cu funcțiile şi operațiile care se implementează;
- **robustețea** programului : calitatea sa de a continua chiar şi la apariția unor erori;
- **mentenabilitatea** programului : uşurința cu care poate fi modificat şi îmbunătățit ulterior.

Stilul de programare

Pârghii de acţiune

1. **Organizarea programului**: definirea subprogramelor (funcțiilor) în conformitate cu principalele activități şi operații desprinse din problema care urmează a fi rezolvată.
2. **Organizarea datelor**: trebuie să concorde cu structura obiectelor din realitatea problemei care se rezolvă.
3. **Comunicarea între subprograme**: se recomandă să se realizeze în special prin mecanismul parametri-argumente și numai în mod excepțional prin alte variante.
4. **Testarea şi tratarea** prin program a **unor categorii de erori**, de exemplu cele de intrare/ieşire. Unele limbaje, inclusiv C, permit astfel de operații ceea ce determină îmbunătățirea robusteții programelor.

Stilul de programare

Pârghii de acţiune

5. **Alegerea numelor simbolice** (identificatorilor) din program astfel încât să se facă o legătură directă cu semnificația entității respective în problema de rezolvat. În acest fel se poate uşura urmărirea şi înțelegerea programului.
6. **Utilizarea comentariilor**: reprezintă cea mai simplă **metodă de documentare** a programelor, utilă chiar şi autorilor programului, dacă îl reanalizează după un anumit timp. În practica programării se utilizează linii de comentarii distincte prin care se descriu funcțiile programului, funcția fiecărui subprogram, datele de intrare şi cele de ieşire, indicații de utilizare a programului etc. De asemenea, se obişnuieşte ca prelucrările mai importante sau mai dificile din program să fie însoțite de comentarii explicative.

Stilul de programare
Pârghii de acţiune

7. **Formatul liber** de redactare al liniilor sursă. Această facilitate, prezentă în majoritatea limbajelor de programare moderne, permite punerea în concordanță a textului programului cu organizarea şi semnificația sa. Utilizarea **indentării** duce la creşterea lizibilitatăți şi clarități programului. Indentarea reprezintă deplasarea spre dreapta a unui bloc de date sau de instrucțiuni, care aparține unei structuri, față de marginea stângă a elementelor structurii înconjurătoare.

Metoda detalierilor în paşi succesivi
(Step Wise Refinement – SWR)

Cu cât problema este mai complexă, trecerea de la enunțul problemei la program este mai dificilă. În aceste situații este util ca atât elaborarea algoritmului cât şi scrierea programului să se facă treptat, printr-un proces de **detaliere succesivă** în mai mulți paşi :

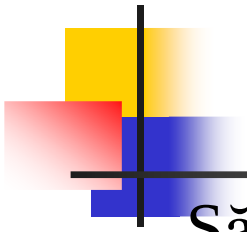
- **Pasul 1** : Se scrie *funcția principală(main)* utilizând, pe cât posibil, apeluri de funcții corespunzătoare operațiilor de bază care se disting în această fază. Se descriu structurile de date aferente programului. Funcțiile apelate din funcția principală se definesc doar la modul generic (prototip urmat de /*comentarii*/).
- **Paşii 2, 3, ...** : Se dezvoltă funcțiile apelate direct din funcția principală (pasul 2), apoi funcțiile apelate din acestea din urmă (pasul 3) ş.a.m.d., până la elaborarea, în amănunt, a întregului program.

Metoda detalierilor în paşi succesivi
(Step Wise Refinement – SWR)

Metoda detalierilor în paşi succesivi este o metodă de proiectare-programare **descendentă** (top-down).

Metoda SWR porneşte de la general şi dezvoltă detaliile din aproape în aproape.

În principiu, în fiecare pas se dezvoltă funcțiile care au apărut (fiind doar enunțate) în pasul anterior. Dacă operațiile implementate în pasul curent sunt suficient de complexe, este posibil să fie enunțate noi funcții care vor fi dezvoltate (detaliate) în pasul următor.



Exemplu de program

Enunţul problemei

Să se scrie un program pentru realizarea unui top al melodiilor. Persoanele care participă la alcătuirea topului se împart în 4 categorii, după sex si vârstă (mai tineri de 20 de ani şi peste 20 de ani).

Fiecare persoană nominalizează, în ordine, 5 melodii preferate, identificate prin titlu. Datele privind persoanele participante la realizarea topului se citesc de pe mediul de intrare sau dintr-un fişier. Datele pentru o persoană vor fi de forma:

nume prenume sex(m sau f) vârsta
melodie1
melodie2
melodie3
melodie4
melodie5

Exemplu de program
Enunţul problemei

Să se afiseze:

1. Lista melodiilor în ordinea popularității lor.

2. Listele câștigătorilor (cei care au ghicit melodiile câștigatoare), şi anume: 4 liste separate cuprinzând numele si prenumele persoanelor care au menționat pe prima pozitie a preferințelor una dintre cele mai solicitate 3 melodii la categoria lor.

Program exemplu

Pasul 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define NMEL 5

typedef struct pers{
  char * nume;
  char * prenume;
  char  sex;
  int varsta;
  char * melodii[NMEL];
} persoana;
```

Program exemplu
Pasul 1

```
typedef struct m {
  char * titlu;
  int punctaj;
} melodie;

persoana * citire(int *np){
// citeste datele din intrare si le pune intr-o lista
}
void afisare_test(persoana *plista, int np){
// afisararile de test sunt foarte importante, pentru a descoperii
// eventualele greseli in faza incipienta
}
```

Program exemplu
Pasul 1

```
melodie * top_melodii(persoana *plista, int np){
// creaza o lista cu melodiile si voturile aferente fiecarei melodii
// ordoneaza lista de melodii descrescator, functie de nr. voturilor
// afiseaza lista de melodii
}

void castigatori(melodie *ptop, persoana *plista, int np){
 // se apeleaza functia afiseaza_castigatori pentru cele 4 categorii
 // metoda are avantajul de a modifica foarte usor parametrii listelor
  // de exemplu vom putea imparti in 3 grupe de varsta lista
  // corespunzatoare fiecarui sex doar apeland de 6 ori functia
  // afiseaza_castigatori
}
```

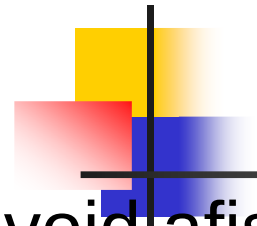
Program exemplu

Pasul 1

```
int main() {
  int npers;
  melodie * top;
  persoana * lista=NULL; /* lista persoanelor */
  lista=citire(& npers); //citeste datele si le pune in lista
  afisare_test(lista, npers);
  top=top_melodii(lista, npers); //topul ordonat al melodiilo
  castigatori(top, lista, npers); //afiseaza cele 4 liste
  return 0;
}
```

Program exemplu

Pasul 2



```
void afisare_test(persoana *plista, int n){
  int i,j;
  for( i=0; i<n; i++ ){
      printf("\n persoana %d", i);
      printf("\n Nume: %s  Prenume: %s sex:%c v:%d \n",
          plista[i ].nume, plista[ i ].prenume, plista[ i ].sex,
          plista[i ].varsta);
      for( j=0; j<NMEL; j++ )
           printf("%s  ",plista[ i ].melodii[ j ]);
  }
}
```

Program exemplu

Pasul 2

```
melodie *creaza_lista_melodii(persoana *plista,int n, int *nm){
  // creaza o lista cu melodiile si voturile aferente fiecarei melodii
}

void sorteaza_lista_melodii(melodie *ptop, int nm){
  // creaza o lista ordonata descrescator, functie de numarul voturilor,
  // din lista de melodii primita ca parametru
}

void afiseaza_top(melodie *ptop,int nm){
   // afiseaza lista de melodii primita ca parametru
}
```

Program exemplu

Pasul 2

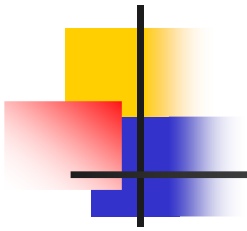
```
melodie * top_melodii(persoana *plista, int np){
   melodie *ptop;
   int nmelodii;
   // creaza o lista cu melodiile si voturile aferente fiecarei melodii
   ptop=creaza_lista_melodii(plista, np, & nmelodii);
   // ordoneaza lista de melodii descrescator, functie de nr.voturi
   sorteaza_lista_melodii(ptop, nmelodii);
   // afiseaza lista de melodii
   afiseaza_top(ptop, nmelodii);
   return ptop;
}
```

Program exemplu

Pasul 2

```
void afiseaza_castigatori(persoana *plista,int
        np,melodie * ptop, char sx, int v1, int v2)
{
 // se parcurge lista de persoane plista si se afiseaza
// persoanele cu sexul sx si varsta cuprinsa intre v1 si
v2, // adica plista[i].sex==sx si v1<=lista[i].varsta<=v2
(pseudocod)
}
```

Program exemplu
Pasul 2

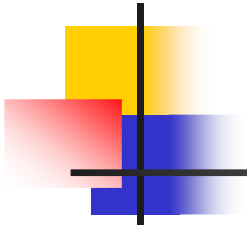


```
void castigatori(melodie *ptop, persoana *plista, int np)
{
    afiseaza_castigatori(plista,np,ptop,'f',1,19);
    afiseaza_castigatori(plista,np,ptop,'f', 20,150);
    afiseaza_castigatori(plista,np,ptop,'m',1,19);
    afiseaza_castigatori(plista,np,ptop,'m', 20,150);
    /* o alta metoda consta in parcurgerea listei de persoane o singura data, si crearea in acest timp a celor 4 liste mai mici, pe care sa le afisam ulterior*/
}
```

Concluzii

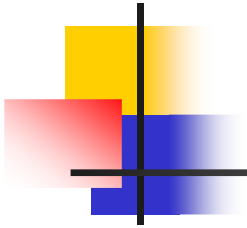
1. Dezvoltarea programelor constă într-o succesiune de paşi de rafinare. La fiecare pas o operație complexă este descompusă într-un număr de operații mai simple. Fiecare rafinare la nivelul unei operații poate fi însoțită de o rafinare a structurilor de date aferente, datele fiind cele care realizează interacțiunea dintre (sub)operații. Rafinarea descrierii operațiilor şi a structurilor de date trebuie să se realizeze în paralel.
2. Gradul de modularitate obținut va determina uşurința sau dificultatea cu care programul va putea fi adaptat la modificările şi extensiile ulterioare, inclusiv la schimbarea mediului de execuție (limbaj de programare, sistem de operare, platformă hardware).

Concluzii



3. Pe parcursul rafinărilor succesive vor fi folosite mai multe notații. Este recomandabil ca notația inițială, care derivă din natura problemei de rezolvat, să fie menținută cât mai mult cu putință. Direcția în care va evolua notația, pe parcursul rafinărilor, depinde, în ultimă instanță, de limbajul de programare în care va fi scris programul în variantă finală. Notația finală se identifică cu limbajul de programare ales pentru implementarea programului. De aceea, limbajul de programare trebuie să permită exprimarea cât mai clară şi naturală a structurilor de date şi de control care apar în procesul de proiectare.

Concluzii



4. Fiecare pas de rafinare implică luarea unui număr de decizii pe baza unor criterii de performanță. Printre aceste criterii se numără : viteza de execuție, spațiul de memorie necesar rulării, claritatea codului, uniformitatea structurilor etc. Programatorul trebuie să fie consecvent în ceea ce priveşte luarea acestor decizii, trebuie să cântărească cu atenție toate aspectele şi să fie pregătit să revină asupra unor decizii anterioare, chiar dacă aceasta înseamnă să reia proiectarea de la început.