MINISTERUL EDUCAȚIEI ȘI CERCETĂRII
ROMÂNIA
UNIVERSITATEA DE MEDICINĂ, FARMACIE, ȘTIINȚE ȘI TEHNOLOGIE „GEORGE EMIL PALADE" DIN TÂRGU MUREȘ
FACULTATEA DE INGINERIE ȘI TEHNOLOGIA INFORMAȚIEI

Algoritmi fundamentali

Curs 4

Tehnici pentru elaborarea algoritmilor

Dr. ing. Kiss Istvan

istvan.kiss@umfst.ro

Cuprins

1. Tehnici pentru elaborarea algoritmilor

- *Elaborare algoritm = proiectare algoritm*
- Întreaga activitate depusă de la enunțarea problemei până la realizarea algoritmului corespunzător rezolvării acestei probleme.
  - Specificarea problemei
  - Descrierea metodei alese pentru rezolvarea problemei
  - Proiectarea propriu-zisă. Constă în:
    - Descompunerea problemei în subprobleme
    - Obținerea algoritmului principal
    - Obținerea subalgoritmilor apelați
    - Descrierea algoritmului principal și a subalgoritmilor
    - Precizarea denumirilor și semnificațiilor variabilelor folosite
  - Verificarea algoritmului

2. Subalgoritmi

- Orice problemă poate fi descompusă în subprobleme.
- Algoritmul de rezolvare aparținând subproblemei se numește subalgoritm.
- Pentru a descrie un subalgoritm vom folosi următoarea formă:

Subalgoritm *nume(*lpf*)* **este**:

Date locale

Instructiune

Instructiune

...

Sf. nume

- ***nume*** *– numele subalgoritmului*
- ***lpf*** *– lista parametrilor formali*

2. Subalgoritmi

- Exemplu: subalgoritm pentru determinarea maximului dintre elementele unui vector $X=(x_1, x_2,...,x_n)$.

Subalgoritm *maxim(n,X,max)* **_este_***:*

max:=x_1*;*

pentru *i:=2;n* **_executa_**

daca x_i *> max* **_atunci_**

max:=x_i*;*

sf.daca

sf.pentru

Sf.maxim

Se observă faptul că după apelul subalgoritmului rezultatul vom regăsi în variabila maxim!!!

2. Subalgoritmi de tip funcție

- Exemplu1: subalgoritm pentru determinarea maximului dintre elementele unui vector $X=(x_1, x_2,...,x_n)$.

Functia _maxim(n,X)_ **_este_**_:_

max:=x_1;

pentru _i:=2;n_ **_executa_**

daca _x_i > max_ **_atunci_**

max:=x_i;

sf.daca

sf.pentru

maxim:=max;

Sf.maxim

Se observă faptul că după apel funcția returnează rezultatul!!! _Ex.: Afișează maxim(n,X);_

2. Subalgoritmi de tip funcție

- Exemplu2:

FUNCŢIA *numar(x)* ***ESTE****:*

   DACĂ *x<0.2* ***ATUNCI*** *numar:=2* ***ALTFEL***

      DACĂ *x<0.5* ***ATUNCI*** *numar:=3* ***ALTFEL***

         DACĂ *x<0.9* ***ATUNCI*** *numar:=4*

            ALTFEL *numar:=5*

         SFDACĂ

      SFDACĂ

   SFDACĂ

SF*-numar*

- ***Se observă faptul că după apel funcția returnează rezultatul!!!*** *Ex.: Afișează numar(0.7);*

3. Apel de subalgoritm

- **CHEAMĂ** nume(lpa) **sau** nume(lpa)

- Exemplu3:

ALGORITMUL *NUMĂRĂZILE* ***ESTE****:*

CITEŞTE *zi, luna, an;*

nr:=zi;

DACĂ *luna>1* ***ATUNCI***

PENTRU *i:=1, Luna-1* ***EXECUTĂ*** *nr:=nr+NRZILE(i)* ***SFPENTRU***

SFDACĂ

DACĂ *luna>2* ***ATUNCI***

DACĂ *BISECT(an)* ***ATUNCI*** *nr:=nr+1* ***SFDACĂ***

SFDACĂ

TIPĂREŞTE *nr;*

SFALGORITM

- *NRZILE(i)* furnizează numărul zilelor existente în luna *i* a unui an nebisect;
- *BISECT(an)* adevărată dacă anul dintre paranteze este bisect.

Problemă pentru temă: Să se proiecteze funcțiile *NRZILE(i)* şi *BISECT(an)*!

3. Apel de subalgoritm – Ex.4

Se dau trei mulţimi de numere:

$A = \{ a_1, a_2, \dots , a_m \}$

$B = \{ b_1, b_2, \dots , b_n \}$

$C = \{ c_1, c_2, \dots , c_p \}$

Se cere să se tipărească în ordine crescătoare elementele fiecărei mulţimi, precum şi a mulţimilor A U B, B U C, C U A.

Avem nevoie de subalgoritmi:

- *CITMUL(m,A);*
- *REUNIUNE(m,A,n,B,k,R);*
- *TIPMUL(m,A);*
- *ORDON(m,A);*

3. Apel de subalgoritm – Ex.4

ALGORITMUL *OPER-MULTIMI* ***ESTE****:*

  CHEAMĂ *CITMUL(m,A);*

  CHEAMĂ *CITMUL(n,B);*

  CHEAMĂ *CITMUL(p,C);*

  CHEAMĂ *TIPORDON(m,A);*

  CHEAMĂ *TIPORDON(n,B);*

  CHEAMĂ *TIPORDON(p,C);*

  CHEAMĂ *REUNIUNE(m,A,n,B,k,R);*

  CHEAMĂ *TIPORDON(k,R);*

  CHEAMĂ *REUNIUNE(n,B,p,C,k,R);*

  CHEAMĂ *TIPORDON(k,R);*

  CHEAMĂ *REUNIUNE(p,C,m,A,k,R);*

  CHEAMĂ *TIPORDON(k,R);*

SFALGORITM

3. Apel de subalgoritm – Ex.4

SUBALGORITM *CITMUL(n,M)* ***ESTE****:* *{Citeşte n şi M}*

  CITEŞTE *n;* *{n=nr. elementelor mulţimii}*

  CITEŞTE *(m_i,i=1,n);* *{M=mulţimea cu elementele m_1,m_2,...,m_n}*

SF*-CITMUL*

SUBALGORITM *ORDON(n,M)* ***ESTE****:* *{Ordonează crescător cele n}*

  REPETĂ *{elemente ale mulţimii M}*

    ind:=0; *{Cazul M este ordonată}*

    PENTRU *i:=1;n-1* ***EXECUTĂ***

      DACĂ *m_i>m_{i+1}* ***ATUNCI*** *{schimbă ordinea celor}*

        t := m_i; *{două elemente}*

        m_i:=m_{i+1}; m_{i+1}:=t;

        ind:=1; *{Cazul M nu era ordonată}*

      SFDACĂ

    SFPENTRU

  PÂNĂCÂND *ind=0* ***SFREP***

SF*-ORDON*

3. Apel de subalgoritm – Ex.4

SUBALGORITM *REUNIUNE(m,A,n,B,k,R)* ***ESTE****:*

{ R := A U B }

{ k = numărul elementelor mulţimii R }

k:=m; R := A;

PENTRU *j:=1,n* ***EXECUTĂ***

ind:=0; *{Ipoteza* b_j *nu e in A}*

PENTRU *i:=1;m* ***EXECUTĂ***

DACĂ b_j=a_i ***ATUNCI*** *ind:=1 {*b_j *este in A}* ***SFDACĂ***

SFPENTRU

DACĂ *ind=0* ***ATUNCI*** *k:=k+1;* r_k:=b_j ***SFDACĂ***

SFPENTRU

SF*-REUNIUNE*

3. Apel de subalgoritm – Ex.4

***SUBALGORITM** TIPMUL(n,M) **ESTE**:* *{ Tipăreşte cele n elemente }*

* **PENTRU** i:=1;n **EXECUTĂ*** *{ ale mulţimii M }*

*   **TIPĂREŞTE*** m_i

* **SFPENTRU***

***SF**-TIPMUL*

***SUBALGORITM** TIPORDON(n,M) **ESTE**:* *{ Ordonează şi tipăreşte }*

* **CHEAMĂ** ORDON(n,M);* *{ elementele mulţimii M }*

* **CHEAMĂ** TIPMUL(n,M);*

***SF**-TIPORDON*

4. Avantajele subalgoritmilor

- Modularizarea problemei
- Reutilizarea aceleași secvențe de algoritm
- Claritate și depanare mai ușoară

- Din punctul de vedere al calculatorului?

5. Tehnici pentru elaborarea algoritmilor

- *Elaborare algoritm = proiectare algoritm*
- Întreaga activitate depusă de la enunțarea problemei până la realizarea algoritmului corespunzător rezolvării acestei probleme.
  - Specificarea problemei
  - Descrierea metodei alese pentru rezolvarea problemei
  - Proiectarea propriu-zisă. Constă în:
    - Descompunerea problemei în subprobleme
    - Obținerea algoritmului principal
    - Obținerea subalgoritmilor apelați
    - Descrierea algoritmului principal și a subalgoritmilor
    - Precizarea denumirilor și semnificațiilor variabilelor folosite
  - Verificarea algoritmului

5. Proiectare descendentă (top-down)

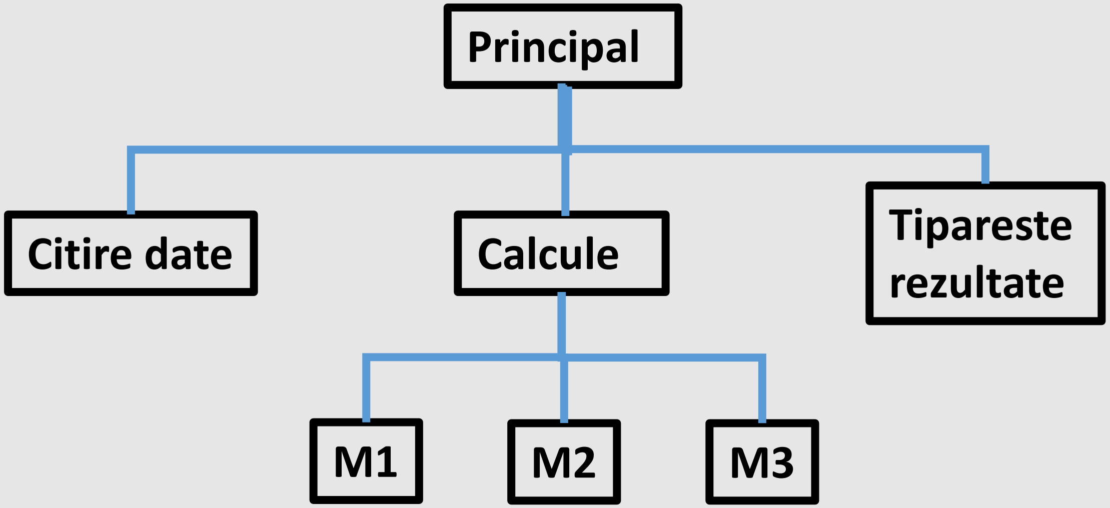
- Se porneste de la Problema pe care descompune în subprobleme rezolvabile separat.
- Pune accent pe algoritmul principal și nu pe subalgoritmi.
- Descompune și pe subalgoritmi în subalgoritmi.
- Definește problema principală, subproblemele și conexiunea între subprobleme.
- Permite lucrul în echipe – subechipele proiectează algoritmi pentru rezolvarea subproblemelor.

5. Proiectare descendentă (top-down)

- Problema
  - Subproblema *1*
    - Subproblema *1*
    - Subproblema *2*
  - Subproblema *2*
  - ...
  - Subproblema *n*
    - Subproblema *1*
    - Subproblema *2*
    - Subproblema *3*
    - Subproblema *4*
    - ...

5. Proiectare descendentă (top-down)

- Conexiunile între subprobleme se reprezintă prin arbore de programare. Ex.:



5. Proiectare descendentă (top-down)

- Este cunoscută și sub denumire ”**Divide et impera**”
- Paradigma ”**Divide et impera**” este folosită și la împărțirea datelor în grupe mai mici de date (ex.: QuickSort)
- În cărți proiectarea descendentă apare și sub denumirea **stepwise-refinement.**

6. Proiectare ascendentă (bottom-up)

- Folosește subalgoritmii existenți (și librăriile limbajului de programare) în proiectare de subalgoritmi pentru a ajunge în final la algoritmul de rezolvare.
- Este important întocmirea unei diagrame de apel a subalgoritmilor ca și în cazul proiectării top-down.

- *Dezavantaj*: erorile subalgoritmilor vor fi observate numai în faza de integrare.

7. Proiectare modulară

- **Modulul** este o unitate structurală de sine stătătoare, fie algoritm, subalgoritm, program, subprogram, unitate de program.
- Un modul poate fi format din mai multe submodule.
- Fiecare modul trebuie să aibă un rol bine precizat.
- Avantaje: reducerea complexității problemei, reducerea probabilității greșelilor de programare, testare ușoară, lucrul în echipă, reutilizabilitate.

8. Probleme

1. Să fie următoarea funcție definită în domeniul numerelor reale:

$$f(x)=\begin{cases}\sqrt[3]{x+2}, & x\geq 10\\ |3x|, & x<10\end{cases}$$

Să se proiecteze un algoritm pentru calculul funcției. x introdus de la tastatură. Precizia de calcul radical să nu depășească 10^{-4}.

8. Probleme - rezolvare

- Vom avea nevoie de subalgoritm pentru radical de ordin 3 și valoare absolută.

- **Algoritmul principal**:

```
Algoritmul calculF este:
        real x, f;
        citeste x, eps;
        daca x>=10 atunci
                f:=radical3(x+2,eps);
        altfel
                f:=absolut(3*x);
        sf.daca
        afiseaza f;
Sf. calculF
```

- **Subalgoritmi**:

1. Functia **radical3**(a,eps);
2. Functia **absolut**(a);

- **Variabile folosite**:
  - x citit de la tastatura
  - f valoarea functiei
  - a parametru formal
  - eps precizie de calcul radical
  - r1, r2 variabile auxiliare in subalgoritm radical3.

8.1. Radical de ordin 3

$$\sqrt[3]{x}$$

- Metoda: Algoritmul se bazează pe relația de recurență:

$$r_1 = 1; \quad r_i = (2 \cdot r_{i-1} + x/(r_{i-1} \cdot r_{i-1}))/3, daca\, i > 1$$

- Precizia:

$$\left| r_i - r_{i-1} \right| < \varepsilon$$

- Vom proiecta un algoritm care foloseste variabila *r1* pentru a retine termenul calculat la pasul precedent si *r2* pentru termenul care urmeaza sa fie calculate.

8.1. Radical de ordin 3

```
Functia radical3(a,eps) este:
        real r1, r2:=1;
        repeta
                r1=r2;
                r2=(2*r1+a/(r1*r1))/3;
        panacand absolut(r2-r1)>=eps;
        radical3:=r2;
Sf.radical3
```

Se observă faptul că functia absolut este de folos si in algoritmul radicalului!!!

8.2. Valoare absoluta

Functia absolut(a) este:

```
        daca a>=0 atunci
                absolut:=a;
        altfel
                absolut:=-a;
        sf.daca
Sf.absolut
```

Urmatorul pas de proiectare este testarea solutiei!