Lab. 1 - Obiectivele si principiile Ingineriei Programarii (IP) - Objectives and Principles of Software Engineering

Notiuni de baza

- **1. Obiectivele IP**

Obiectivul final al IP este trecerea de la activitati de elaborare in care domina si continua sa domine intr-o proportie mare stilul artizanal, stilul de intuitie, de creatie de tip "arta" la o aplicatie sistematica care sa asigure o anumita *calitate garantata* a programelor cit si un *cost cit mai scazut* in elaborare si intretinere.

Obiective generale ale IP

a) *Adaptabilitatea programelor* reprezinta posibilitatea de a face modificari in mod controlat in aplicatie. Ea depinde de mai multi factori:
-posibilitatea de a transfera programe pe un alt tip de calculator sau pe un alt sistem de operare (portabilitate)
-posibilitatea de a adauga noi functii/metode sau clase
-ideea de a imbogati performantele sistemului sau ale programelor (schimbind metoda de programare)
-inlaturarea anomaliilor

In telecomunicatii e foarte importanta adaptabilitatea programelor, deoarece acest domeniu se afla intr-o dinamica puternica.

b) *Eficienta programelor* se refera la capacitatea acestor programe de a realiza functii cerute in aumite conditii de eficienta. Eficienta se masoara prin timp de calcul si prin memorie ocupata.

c) *Fiabilitatea programelor* reprezinta evitarea, depistarea si inlaturarea erorilor in programare in fazele de conceptie si implementare, precum si masuri de restabilire a functionarii.

d) *Perceptibilitatea* reprezinta prioritatea programelor de a fi usor de inteles si urmarit de programator altul decit cel care l-a creat. Pentru aceasta se folosesc diagrame Booch, scheme logice, diagrame UML, etc. In afara de aceasta un program trebuie sa fie usor de utilizat de catre beneficiar.

♦ 2. Principiile generale ale IP

a) Principiul modularizarii

Modularizarea e de mai multe tipuri :
-*modularizare functionala* – adica un modul functioneaza independent de celelalte
-*modularizare structurala* – structurile de date care sint combinatii de date elementare legate intre prin relatii
-*modularitate procedurala* – ordinea de executie a operatiilor
-*modularitate obiectuala* – relatiile dintre ierarhii, clase
-*modularitate bazata pe componente si servicii* – relatiile dintre componente in vederea definirii si descoperirii de servicii

b) Principiul abstractizarii identifica problemele generale si omite detalile neesentiale

c) Principiul localizarii se refera la a pune intr-o vecinatate fizica elementele care au o legatura intre ele.

d) Principiul incapsularii informatiei (al ascunderii informatiei) se refera la scoaterea in evidenta a trasaturilor esentiale, ascunzind detalii ce nu afecteaza celelalte parti componente ale unui sistem.

e) Principiul uniformitatii (de asigurare a consistentei programului) ne spune sa folosim notatii sugestive care sa nu fie ambigue

f) Principiul completitudinii ne spune ca aplicatia trebuie sa nu omita nici un element constructiv esential

g) Principiul confirmabilitatii se refera la necesitatea ca informatiile pentru verificarea corectitudinii programului sa fie explicit formulate

♦ 3. Elaborarea unei aplicatii pornind de la principiile si obiectivele IP

Exemplu: Sa se scrie un program care citeste un numar real pozitiv, subunitar, calculeaza si afiseaza radacina patrata din numarul respectiv cu o eroare mai mica de 10^{-10} .

Programul de fata e implementat in limbajul C/C++ si nu foloseste functia *sqrt()* din biblioteca standard. Pentru a extrage radacina patrata dintr-un numar cuprins in intervalul (0,1), se poate folosi metoda iterativa a lui Newton de mai jos.

Fie sirul:

X[0], X[1], ... , X[n], ...

Unde X[n+1]=0.5(X[n]+a/X[n]), pentru n= 0,1,2... ***rel(1)***

Se demonstreaza ca sirul de mai sus este convergent si are limita egala cu radical din *a*. Convergenta sa e rapida pentru *a* subunitar. In acest caz se poate lua X[0]=1.

Pentru a obtine radacina patrata cu precizia ceruta, este suficient ca diferenta in valoare absoluta dintre doi termeni consecutivi ai sirului sa fie mai mica decit eroarea maxima admisa EPS=10^{-10}. Pentru a rezolva problema cu metoda indicata mai sus la fiecare pas al calculului sunt necesari numai doi termeni ai sirului:

1. - din X[n] se determina X[n+1]

2. – se compara abs(X[n]-X[n+1]) cu EPS=10^{-10}

Daca valoarea absoluta respectiva nu este mai mica decit EPS, atunci se calculeaza X[n+2] folosind ***rel(1***), in care X[n] se inlocuieste cu X[n+1]

Deci se poate reveni la primul punct descris mai sus, dupa ce lui X[n] i se atribuie valoarea X[n+1]. Asadar in loc sa utilizam tabloul *X* este suficient sa folosim doua variabile X1 si X2 care pastreaza in orice moment doi termeni ai sirului de mai sus. Termenii sirului se obtin executand repetat secventa de atribuiri:

```
X1=X2;
X2= 0.5*(1+a/X1);
```

Dupa fiecare executie a acestei secvente, X1 si X2 au ca valori doi termeni ai sirului cautat. Secventa se va repeta atat timp cat:

```
abs(X1-X2) >= EPS
```
rel(2)

Acest test il facem dupa fiecare executie a secventei de atribuiri. Daca el se confirma, atunci se repeta secventa respectiva, altfel valoarea variabilei X2 aproximeaza corect radicalul. Acest proces se realizeaza in implementarea C/C++ printr-o, instructiune *do-while*. Secventa de atribuiri este corpul ciclului iar ***rel2*** reprezinta conditia de continuare a lui. Intrucat primul termen al sirului are valoarea 1, instructiunea repetitiva e precedata de atribuirea X2=1.

Codul programului este:

```
//Radical aproximare Newton pentru numere subunitare
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#define EPS 1e-10

int main() /* citeste pe “a” din intervalul (0,1), calculeaza si afiseaza radacina
patrata din a */
{
	double a, X1, X2, y;
	printf("tastati valoarea lui a (0,1) =");
	if (scanf("%lf", &a) != 1 || a < 0 || a>1) {
		printf("nu s-a tastat un numar in intervalul [0,1]\n");
		_getch();
		exit(1);
	}
	if (a == 0)
		X2 = 0; /* radical din zero este zero*/
	else
		if (a == 1)
			X2 = 1; /* radical din 1 este 1 */
		else { /* “a” este diferit de 0 sau 1 */
			X2 = 1;
			do {
				X1 = X2;
				X2 = 0.5*(X1 + a / X1);
				if ((y = X1 - X2) < 0)
					y = -y; /* y = abs(X1-X2) */
			} while (y >= EPS);
		} /* else */
	printf("\nRadical din a=%g\tsqrt(a) este = %g\n", a, X2);
	_getch();
	 }
```

4. Teme

1. Programul poate fi extins ca aplicatie si pentru numere supraunitare. In acest caz se ia ca prim termen al sirului chiar numarul din care se extrage radical.

O alta posbilitate e de a extrage radical din inversul numarului care este supraunitar care vine inmultit apoi cu numarul initial rezultand radacina patrata. Solutia e buna pentru ca sirul converge rapid pentru valori subunitare.

Implementati noua versiune a aplicatiei care sa permita radical din ori ce numar real folosind Visual Studio, mediul C/C++. Utilizati in calculul radicalului pentru numarul supraunitar ambele variante prezentate. Afisati timpul de conversie. (Obs. folositi din <*time.h*> metoda *clock()* care returneaza timpul in milisecunde). Daca timpul nu apare a fi semnificativ, modificati valoarea preciziei EPS.

Pentru compatibilizarea intre versiunile mediului C++ folositi daca e cazul ca si o prima directiva preprocesor in aplicatiile dezvoltate constructia:

 # define _CRT_SECURE_NO_WARNINGS

Exemplu :

```
//Radical aproximare Newton pentru numere subunitare
//#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#define EPS 1e-10

int main() /* citeste pe “a” din intervalul (0,1), calculeaza si afiseaza radacina
patrata din a */
{
	double a, X1, X2, y;
	printf("tastati valoarea lui a (0,1) =");
	if (scanf_s("%lf", &a) != 1 || a < 0 || a>1) {
		printf("nu s-a tastat un numar in intervalul [0,1]\n");
		exit(1);
	}
	if (a == 0)
		X2 = 0; /* radical din zero este zero*/
	else
		if (a == 1)
			X2 = 1; /* radical din 1 este 1 */
		else { /* “a” este diferit de 0 sau 1 */
			X2 = 1;
			do {
				X1 = X2;
				X2 = 0.5 * (X1 + a / X1);
				if ((y = X1 - X2) < 0)
					y = -y; /* y = abs(X1-X2) */
			} while (y >= EPS);
```

```
            } /* else */
        printf("\nRadical din a=%g\tsqrt(a) este = %g\n", a, X2);
}
```

2. Analizati care din obiectivele si principiile IP le putem regasi in aceasta implementare. Modificati eventual aplicatia astfel incat sa le puteti identifica.

3. Considerati o alta aplicatie (licenta, etc.) in care sa identificati obiectivele si principiile IP, daca le-ati considerat. Analizati posibilitatea de a le introduce.