Pregătire pentru examenul de Bacalaureat şi examenul de admitere la Facultatea de Matematică şi Informatică

Soluţii probleme sesiunea 2

13 ianuarie 2018

INFORMATICĂ

Tema 2: Algoritmi de prelucrare a datelor simple şi exemple de utilizare a subprogramelor (funcţii şi proceduri)

- Prelucrări asupra cifrelor (extragerea şi analiza cifrelor unui număr natural)
- Prelucrări ale unor secvenţe de valori preluate secvenţial (fără să fie stocate în structuri de tip tablou).
- Prelucrări asupra punctelor date prin coordonate carteziene
- Prelucrări cu numere complexe

1. Se asociază unui număr natural, n, valoarea A(n) ca fiind suma factorialelor cifrelor lui n (de exemplu, pentru n=318, A(n)=3!+1!+8!).
   a. Pentru o valoare n dată să se calculeze A(n). Calculaţi numărul de operaţii de înmulţire efectuate. Propuneţi o variantă care să utilizeze un număr cât mai mic de operaţii de înmulţire.
   b. Pentru o valoare naturală k (k<9999) să se determine cel mai mic număr natural n cu proprietatea că A(n)=k.

Rezolvare.

a. Intrucât valoarea factorialului trebuie calculată pentru fiecare cifră a numărului n este indicat să se definească o funcţie care are un parametru de intrare (un număr natural) şi returnează valoarea factorialului. Funcţia va fi apelată pentru fiecare cifră a numărului n.

Subalgoritm calcul factorial	Funcţie C

`Întreg fact(întreg c)`
`  Întreg i,f`
`  f←1`
`  pentru i←2,c execută`
`  f←f*i`
`  sfârşit_pentru`
`  returnează f`
	`int fact(int c)`
	`{int i,f=1;`
	`  for(i=2;i<=c;i++)`
	`  f=f*i;`
	`  return f;`
	`}`

Observaţii	**Funcţie Pascal**

• c este parametru de intrare
• i şi f sunt variabile locale
	`Function fact(c:integer):integer;`
	`Var i,f:integer;`
	`Begin`
	`  f:=1`
	`  for i:=2 to c do`
	`  f:= f*i;`
	`  fact:=f;`
	`End;`

Pentru calculului lui A(n) este de asemenea util să se definească o funcţie care primeşte un număr natural $n=c_m c_{m-1}\ c_{m-2}\ldots\ c_1 c_0$ ca parametru de intrare şi returnează suma $A(n)= c_m!+c_{m-1}!+c_{m-2}!+\ldots+c_1!+c_0!$. Cifrele numărului n vor fi extrase succesiv, începând cu cifra unităţilor c_0, prin împărţire la 10. Dacă se apelează pentru fiecare cifră funcţia pentru calculul factorialului atunci numărul total de operaţii de înmulţire efectuate este $g(c_m) + g(c_{m-1}) + g(c_{m-2}) +\ldots+ g(c_1) + g(c_0)$ unde g(c)=0 pentru c din {0,1} şi g(c)=c-1 dacă c>1. De exemplu, pentru n=318 se vor efectua 2+0+7=10 înmulţiri. Pentru a

reduce numărul de înmulţiri efectuate ar trebui să se evite repetarea unor operaţii folosind faptul că valoarea factorialului pentru o cifră poate fi obţinută pornind de la valori calculate pentru cifre mai mici. De exemplu 8!=3!*4*5*6*7*8, deci în loc de 7 înmulţiri se efectuează 5 înmulţiri. Pentru a exploata acest lucru valorile factorialelor ar trebui calculate în ordinea crescătoare a valorilor cifrelor (de exemplu pentru 318 s-ar calcula 1!+3!+8!) ceea înseamnă că cifrele ar trebui stocate într-un tablou în ordine crescătoare.

Subalgoritm calcul A(n)	Secvenţă program C

`Întreg calculA (n)`
`  Întreg a`
`  a ← 0`
`  cât timp n>0 execută`
`  a ← a+fact(n MOD 10)`
`  n ← n DIV 10`
`  sfârşit_cât_timp`
`  returnează a`
	`int calculA(int n)`
	`{int a=0;`
	` while (n>0)`
	`  {a=a+fact(n%10);`
	`  n=n/10;`
	`  }`
	` return a;`
	`}`

Observaţii	**Funcţie Pascal**

• n este parametru de intrare
• a este variabilă locală
• Program complet[1]: P1.cpp, P1.pas
	`Function calculA(n:integer):integer;`
	`Var a:integer;`
	`Begin`
	`  a:=0;`
	`  while n>0 do`
	`  begin`
	`  a:=a+fact(n MOD 10);`
	`  n:=n DIV 10;`
	`  end;`
	`  calculA:=a;`
	`End;`

b. Este o problemă de căutare care necesită analiza succesivă a valorilor naturale, începând cu 0, până la întâlnirea primei valori n care are proprietatea că A(n)=k.

Subalgoritm căutare	Funcţie C

`Întreg caut(k)`
`  Întreg i, imax`
`  Boolean există`
`  i ← 1`
`  există←Fals`
`  cât timp (există=Fals) şi`
`  (i<=imax)`
`  dacă k=CalculA(i)`
`  atunci există←Adevărat`
`  altfel i ← i+1`
`  sfârşit_dacă`
`  sfârşit_cât_timp`
`  dacă există=Adevărat`
`  atunci`
`  returnează i`
`  altfel`
`  returnează -1`
`  sfârşit_dacă`
	`int caut(int k)`
	`{int i=1, exist;`
	` i=1;`
	` exist=0;`
	` while ((exist==0)&&(i<INT_MAX))`
	`  {if(k==calculA(i))`
	`  exist=1;`
	`  else`
	`  i++;`
	`  }`
	` if (exist)`
	`  return i;`
	` else`
	`  return -1;`
	`}`

[1] Implementarile in C++ sunt în arhiva rezolvariC.zip şi au fost testate folosind DevCpp (http://www.bloodshed.net/) iar implementările în Pascal sunt în arhiva rezolvariPascal.zip şi au fost testate folosind https://www.ideone.com/

Observaţii	Funcţie Pascal

• Pentru anumite valori ale lui k numărul n cu proprietatea A(n)=k este foarte mare
• Se limitează căutarea prin specificarea unei valori maxime
• Pt varianta în C, INT_MAX e constantă definită în limits.h
• Program complet: P1.cpp, P1.pas
	`Function caut(k:integer):integer;`
	`Var i,imax:integer;`
	`  exist:boolean;`
	`Begin`
	`  i:=1;`
	`  exist:=false;`
	`  imax:=32767;`
	`  while (exist=false)and(i<imax) do`
	`  if k=calculA(i)`
	`  then exist:=true`
	`  else i:=i+1;`
	`  if exist=true`
	`  caut:=i`
	`  else`
	`  caut:=-1;`
	`End;`

2. Determinaţi cel mai mic şi cel mai mare număr constituit din k cifre (k<5) având proprietatea că suma cifrelor este S. *Date de test:* k=3, S=25; k=4, S=33; k=4, S=12.

Rezolvare. In prima etapă se determină cel mai mic număr natural cu k cifre (10^{k-1}) respectiv cel mai mare număr natural cu k cifre (10^k). Este suficient să se calculeze inf=10^{k-1}, după care sup=10*inf-1. Pentru a determina suma cifrelor se defineşte un subalgoritm care are un parametru de intrare (n) şi returnează suma cifrelor.

Subalgoritm suma cifre	Funcţie C

`Întreg sumaCifre(n)`
`  Întreg s`
`  s ← 0`
`  cât timp n>0 execută`
`  s ← s+ n MOD 10`
`  n ← n DIV 10`
`  sfârşit_cât_timp`
`  returnează s`
	`int sumaCifre(int n)`
	`{int s=0;`
	`  while (n>0)`
	`  {s=s+n%10;`
	`  n=n/10;`
	`  }`
	`  return s;`
	`}`

Observaţii	**Funcţie Pascal**

• n este parametru de intrare
• s este variabilă locală
• Program complet: P2.cpp, P2.pas
	`Function sumaCifre(n:integer):integer;`
	`Var s:integer;`
	`Begin`
	`  s:=0;`
	`  while n>0 do`
	`  begin`
	`  s:=s+ n MOD 10;`
	`  n:=n DIV 10;`
	`  end;`
	`  sumaCifre:=s;`
	`End;`

Pentru determinarea numerelor cerute în enunţ este suficient să se descrie două prelucrări repetitive care se opresc în momentul în care se ajunge la o valoare care are suma cifrelor egală cu S. In prima prelucrare se porneşte de la inf şi se incrementează valoarea până la satisfacerea condiţiei, iar în cealaltă se porneşte de la

stop şi se decrementează valoarea până la satisfacerea condiţiei. Cele două valori determinate vor fi transmise către algoritmul apelant prin intermediul unor parametri de ieşire (de tip referinţă).

Subalgoritm caut valori	Funcţie C

cată(întreg k,s,min,max)
  Întreg i,nr,inf,sup
  Boolean final
  inf ← 10^{k-1}
  sup ← 10*inf-1
  min ← -1
  max ← -1
  nr ← inf
  final ← fals
  cât timp (nr<=sup) si
  (final=fals)execută
  dacă sumaCifre(nr)=s
  atunci
  min ← nr
  final ← adevarat
  altfel
  nr ← nr+1
  sfârşit_cât_timp
  nr ← sup
  final ← fals
  cât timp (nr>=inf) si
  (final=fals)execută
  dacă sumaCifre(nr)=s
  atunci
  max ← nr
  final ← adevarat
  altfel
  nr ← nr-1
  sfârşit_cât_timp
	void cauta(int k, int s, int& min, int& max)
	{int i,nr,inf,sup;
	 inf=1;
	 for(i=1;i<k;i++)
	  inf=inf*10;
	 sup=10*inf-1;
	 min=-1; max=-1;
	 nr=inf;
	 while (nr<=sup)
	  if (sumaCifre(nr)==s)
	  {min=nr;
	  break;
	  }
	  else
	  nr=nr+1;
	 nr=sup;
	 while (nr>=inf)
	  if (sumaCifre(nr)==s)
	  {max=nr;
	  break;
	  }
	  else
	  nr=nr-1;
	}

Observaţii	**Procedura Pascal**

• k şi s sunt parametri de intrare
• min şi max sunt parametri de ieşire
• dacă nu există număr cu k cifre care să aibă suma cifrelor egală cu S (de exemplu dacă S e mai mare decât 9k) atunci valorile parametrilor de ieşire vor fi -1
• în C parametrii de ieşire se specifică ca fiind adrese de variabile:
  ○ la definire: int *min
  ○ la apel: &min
• în C++ parametrii de iesire se pot specifica şi ca referinţe la variabile:
  ○ la definire: int& min
  ○ la apel: min
• In Pascal parametrii de ieşire se specifică prin `var`
	Procedure cauta(k:integer; s:integer; var min:integer; var max:integer)
	var i,nr,inf,sup:integer, final:boolean;
	begin
	 inf:=1;
	 for i:=1 to k-1 do inf:=inf*10;
	 sup:=10*inf-1;
	 min:=-1; max:=-1;
	 nr:=inf;
	 final:=false;
	 while (nr<=sup) and (final=false) do
	  if (sumaCifre(nr)=s) then
	  begin
	  min:=nr;
	  final:=true;
	  end
	
	  else
	  nr:=nr+1;
	 nr:=sup;

• Program complet: P2.cpp, P2.pas	final:=false;
	while (nr>=inf) and (final=false) do
	if (sumaCifre(nr)=s) then
	begin
	max:=nr;
	final:=true;
	end
	
	else
	nr:=nr-1;
	end;

3. Se pune problema generării unei valori naturale n pornind de la 0 şi folosind doar următoarele două operaţii: (i) dublarea valorii curente; (ii) incrementarea valorii curente. De exemplu 5=(0+1)*2*2+1 se poate obţine aplicând secvenţa de operaţii: incrementare; dublare; dublare; incrementare iar 12=(((0+1)*2+1)*2)*2 se obţine aplicând: incrementare, dublare, incrementare, dublare, dublare.
   a. Pentru o valoare naturală n determinaţi o secvenţă de operaţii care permite generarea valorii.
   b. Determinaţi numărul minim de operaţii de incrementare şi dublare care permit generarea valorii n.

Rezolvare. Numărul minim de operaţii care permit obţinerea valorii n pornind de la 0 este dat de următoarea relaţie de recurenţă:

$$op(n) = \begin{cases} 1 & n = 1 \\ 2 & n = 2 \\ op\left(\frac{n}{2}\right) + 1 & n \text{ par} \\ op(n-1) + 1 & n \text{ impar} \end{cases}$$

Pentru determinarea secvenţei de operaţii se porneşte de la valoarea n iar dacă n e par se împarte la 2 şi se afişează „dublare”, iar dacă n e impar se decrementează şi se afişează „incrementare”. Prelucrarea se repetă până se ajunge la valoarea 0. Dezavantajul acestei variante de rezolvare este că operaţiile se afişează în ordinea inversă a aplicării lor. Pentru a se afişa operaţiile în ordinea aplicării se poate folosi o funcţie recursivă în care afişarea este plasată după apelul recursiv.

Subalgoritm operaţii	Funcţii C

Operatii (întreg n,nrinc, nrdub)
nrinc ← 0; nrdub ← 0;
cât timp n>0 execută
dacă n MOD 2=0 atunci
afişare "dublare"
n ← n DIV 2; nrdub← nrdub+1;
altfel
afişare "incrementare"
n ← n+1; nrinc ← nrinc+1;
sfârşit_cât_timp
	void operatiiIterativ(int n, int *nrinc, int *nrdub)
	{*nrinc=0;
	*nrdub=0;
	while(n>0)
	if (n%2==0)
	{cout<<"dublare";n=n/2; (*nrdub)++;}
	else
	{cout<<"incrementare";n=n-1;
	(*nrinc)++;}
	}
	
	void operatiiRecursiv(int n, int& nr)
	{nr=nr+1;
	if (n==1) {cout<<"incrementare"<<endl; }
	else
	if (n%2==0)
	{operatiiRecursiv(n/2,nr);
	cout<<"dublare"<<endl; }

	`else {operatiiRecursiv(n-1,nr); cout<<"incrementare"<<endl;} }`
Observaţii	Proceduri Pascal

• n este parametru de intrare, iar nrinc, nrdub, respectiv nr sunt parametrii de ieşire (pot fi specificaţi ca adrese (int *nr) sau ca referinţe (int& nr)
• pe lângă modificarea parametrilor de ieşire funcţiile au şi un efect *colateral*: afişează operaţia efectuată
• în funcţie de numărul de operaţii solicitat se pot contariza diferit sau cumulat operaţiile de incrementare şi cele de dublare
• Program complet: P3.cpp, P3.pas
	`Procedure operatiiIterativ(n:integer;var nrinc,nrdub:integer); Begin nrinc:=0; nrdub:=0; while n>0 do if n MOD 2=0 then begin n:=n DIV 2; writeln('dublare'); nrdub:=nrdub+1; end else begin n:=n-1; writeln('incrementare'); nrinc:=nrinc+1; end; End;`
	
	`Procedure operatiiRecursiv(n:integer; var nr:integer); Begin nr:=nr+1; if n=1 then writeln('incrementare') else if n MOD 2=0 then begin operatiiRecursiv(n DIV 2,nr); writeln('dublare'); end else begin operatiiRecursiv(n-1,nr); writeln('incrementare'); end End;`

4. Se citeşte o secvenţă de n perechi de valori reale (x_1, y_1), (x_2, y_2) ..., (x_n, y_n) care reprezintă coordonatele vârfurilor unui poligon convex $P_1P_2...P_n$ (punctul P_i are coordonatele (x_i, y_i)) specificate în ordine trigonometrică. Să se determine:
   a. Perimetrul poligonului.
   b. Aria poligonului.
   c. Coordonatele vârfurilor celui mai mic dreptunghi având laturile paralele cu axele de coordonate care conţine poligonul.
   d. Numărul efectiv de laturi ale poligonului (se presupune că pot exista mai mult de două puncte consecutive coliniare).

   Date de test: $P_1(2,5)P_2(1,4)P_3(1,2)P_4(2,1)P_5(5,1)P_6(8,1)P_7(9,3)P_8(8,5)P_9(5,5)$

Rezolvare.

a. Perimetrul poligonului se determină calculând suma distanţelor dintre vârfurile consecutive: perimetru=$d(P_1,P_2)+ d(P_2,P_3)+... +d(P_{n-1},P_n)+ d(P_n,P_1)$. In acest scop este utilă definirea unei funcţii care primeşte coordonatele a două puncte ca parametri de intrare şi returnează valoarea distanţei euclidiene. Intrucât coordonatele punctelor sunt furnizate succesiv şi se doreşte evitarea stocării tuturor valorilor este suficient să se reţină coordonatele lui P_1 (x1, y1) şi coordonatele punctului curent (xcrt, ycrt) şi ale celui anterior (xant, yant).

Subalgoritmi pentru distanta si calcul perimetru	Funcţii C

`Real dist(real x1,y1,x2,y2)`
`returnează`
`sqrt((x1-x2)*(x1-x2)+`
`(y1-y2)*(y1-y2))`
 	
`Real perimetru(întreg n)`
`Intreg i`
`Real x1, y1, xant, yant, xcrt, ycrt, p`
`p←0`
`citire x1,y1`
`xcrt←x1; ycrt←y1`
`pentru i←2,n execută`
`xant←xcrt; yant←ycrt`
`citire xcrt,ycrt`
`p←p+dist(xant,yant,xcrt,ycrt)`
`sfârşit_pentru`
`p=p+dist(xcrt,ycrt,x1,y1)`
`returnează p`
	`float dist(float x1, float y1, float x2, float y2)`
	`{return`
	`(sqrt((x1-x2)*(x1-x2)+`
	`(y1-y2)*(y1-y2)));}`
	
	`float perimetru(int n)`
	`{int i;`
	`float x1, y1, xant, yant, xcrt, ycrt, p;`
	`p=0;`
	`cout<<"x1="; cin>>x1; xcrt=x1;`
	`cout<<"y1="; cin>>y1; ycrt=y1;`
	`for(i=2;i<=n;i++)`
	`{xant=xcrt; yant=ycrt;`
	`cout<<"x"<<i<<"=";cin>>xcrt;`
	`cout<<"x"<<i<<"="; cin>>ycrt;`
	`p=p+dist(xant,yant,xcrt,ycrt);`
	`}`
	`p=p+dist(xcrt,ycrt,x1,y1);`
	`return p;`
	`}`

Observaţii	Funcţie Pascal

• Program complet: P4.cpp, P4.pas	`Function dist(x1,y1,x2,y2:real):real;`
	`Begin`
	`Dist:= sqrt((x1-x2)*(x1-x2)+`
	`(y1-y2)*(y1-y2)));`
	`End;`
	
	`Function perimetru(n:integer):real;`
	`Var i:integer;`
	`x1,y1,xant,yant,xcrt,ycrt,p:real;`
	`begin`
	`p:=0;`
	`write('x1='); readln(x1); xcrt:=x1;`
	`write('y1='); readln(y1); ycrt:=y1;`
	`for i:=2 to n do`
	`begin`
	`xant:=xcrt; yant:=ycrt;`
	`write('x',i,'=');readln(xcrt);`
	`write('y',i,'=');readln(ycrt);`
	`p:=p+dist(xant,yant,xcrt,ycrt);`
	`end;`
	`p:=p+dist(xcrt,ycrt,x1,y1);`
	`perimetru:=p;`
	`end;`

b. Poligonul fiind convex, aria poligonului poate fi calculată ca fiind suma ariilor tuturor triunghiurilor care au un vârf în P_1 şi celelalte două vârfuri în vârfuri consecutive ale poligonului: aria=arieTriunghi(P_1,P_2,P_3)+ arieTriunghi(P_1,P_3,P_4)+…+ arieTriunghi(P_1,P_{n-1},P_n). Aria unui triunghi poate fi calculată pornind de la lungimile laturilor (a,b,c) folosind formula lui Heron. Pentru un triunghi $P_1P_{i-1}P_n$ având laturile
a = dist(P_1,P_{i-1}), b = dist(P_{i-1},P_i), c= dist(P_i,P_1) formula de calcul este
arieTriunghi($P_1P_{i-1}P_n$)=sqrt(p*(p-a)*(p-b)*(p-c)) unde p=(a+b+c)/2 este semiperimetrul triunghiului.
Având în vedere faptul că punctele se preiau succesiv este necesar să se reţină coordonatele lui P1 (x1,y1), ale punctului curent (xcrt,ycrt) şi ale punctului anterior (xant, yant). Pentru a evita recalcularea unor distanţe este util să se reţină distanţele d(P1,Pant) şi d(P1,Pcrt).

Subalgoritmi pentru calcul arii	Funcţii C

Real arieTr(real a,b,c)
p←(a+b+c)/2
returnează
sqrt(p*(p-a)*(p-b)*(p-c))
 	
Real arie(întreg n)
Intreg i
Real x1, y1, xant, yant, xcrt, ycrt,a
a←0
citire x1,y1
citire xcrt, ycrt
dP1Pcrt←dist(x1,y1,xcrt,ycrt);
pentru i←3,n execută
xant←xcrt; yant←ycrt
citire xcrt,ycrt
dP1Pcrt←dist(x1,y1,xcrt,ycrt);
a←a+arieTr(dP1Pant,
dist(xant,yant,xcrt,ycrt),
dP1Pcrt);
sfârşit_pentru
returnează a
	float semiper(float a, float b, float c)
	{return((a+b+c)/2);}
	
	float arieTr(float a, float b, float c)
	{float p;
	p=semiper(a,b,c);
	return(sqrt(p*(p-a)*(p-b)*(p-c)));
	}
	
	float arie(int n)
	{int i;
	float x1, y1, xant, yant, xcrt, ycrt, a;
	float dP1Pant,dP1Pcrt;
	a=0;
	cout<<"x1="; cin>>x1;
	cout<<"y1="; cin>>y1;
	cout<<"x2="; cin>>xcrt;
	cout<<"y2="; cin>>ycrt;
	dP1Pcrt=dist(x1,y1,xcrt,ycrt);
	for(i=3;i<=n;i++)
	{xant=xcrt; yant=ycrt;
	dP1Pant=dP1Pcrt;
	cout<<"x"<<i<<"="; cin>>xcrt;
	cout<<"x"<<i<<"="; cin>>ycrt;
	dP1Pcrt=dist(x1,y1,xcrt,ycrt);
	a=a+arieTr(dP1Pant,
	dist(xant,yant,xcrt,ycrt),
	dP1Pcrt);
	}
	return a;
	}

Observaţii	Funcţie Pascal

• Program complet: P4.cpp, P4.pas	Function arieTr(a,b,c:real):real;
	Var p:real;
	Begin
	p:=(a+b+c)/2;
	arieTr:= sqrt(p*(p-a)*(p-b)*(p-c));
	End;
	
	Function arie(n:integer):real;
	Var i:integer;
	x1,y1,xant,yant,xcrt,ycrt,p:real;
	begin
	a:=0;

	write('x1='); readln(x1);
	write('y1='); readln(y1);
	write('xcrt='); readln(xcrt);
	write('ycrt='); readln(ycrt);
	dP1Pcrt:=dist(x1,y1,xcrt,ycrt);
	for i:=3 to n do
	begin
	xant:=xcrt; yant:=ycrt;
	write('x',i,'=');readln(xcrt);
	write('y',i,'=');readln(ycrt);
	dP1Pant:=dP1Pcrt;
	dP1Pcrt:=dist(x1,y1,xcrt,ycrt);
	a:=a+arieTr dP1Pant,
	dist(xant,yant,xcrt,ycrt),
	dP1Pcrt);
	end;
	arie:=a;
	end;

c. Cel mai mic dreptunghi (având laturile paralele cu axele de coordonate) are colţul din stânga jos în punctul de coordonate (xmin,ymin) iar colţul din dreapta sus în punctul de coordonate (xmax,ymax) unde xmin este cea mai mică abscisă întâlnită, ymin este cea mai mică ordonată întâlnită, xmax este cea mai mare abscisă întâlnită, ymax este cea mai mare ordonată întâlnită. Prin urmare este suficient ca xmin şi xmax să se iniţializeze cu x1 iar ymin şi ymax cu y1 după care, pentru fiecare nou punct, se compară coordonatele acestuia cu xmin, xmax, ymin, ymax şi se fac actualizări dacă este cazul.

Subalgoritmi pentru determinare dreptunghi	Funcţie C

dreptunghi(întreg n, real xmin, ymin, xmax ymax)
Intreg i
Real x,y
citire x,y
xmin←x; xmax←x
ymin←y; ymax←y
pentru i←2,n execută
citire x,y
dacă x<xmin atunci
xmin←x
sfârşit_dacă
 	
dacă x>xmax atunci
xmax←x
sfârşit_dacă
dacă y<ymin atunci
ymin←y
sfârşit_dacă
dacă y>ymax atunci
ymax←y
sfârşit_dacă
 	
sfârşit_pentru
	void dreptunghi(int n, float& xmin, float &ymin, float& xmax, float& ymax)
	{int i;
	float x,y;
	cout<<"x1="; cin>>x;
	cout<<"y1="; cin>>y;
	xmin=xmax=x; ymin=ymax=y;
	for(i=2;i<=n;i++)
	{cout<<"x"<<i<<"="; cin>>x;
	cout<<"x"<<i<<"="; cin>>y;
	if (x<xmin) xmin=x; if (x>xmax) xmax=x;
	if (y<ymin) ymin=y; if (y>ymax) ymax=y;
	}
	}

Observaţii	Procedură Pascal
• xmin, ymin, xmax, ymax sunt specificaţi ca parametri de ieşire • Program complet: P4.cpp, P4.pas	`Procedure dreptunghi (n:integer; var xmin,ymin,xmax,ymax:real); Var i:integer; x,y:real; begin write('x='); readln(x); write('y='); readln(y); xmin:=x; xmax:=x; ymin:=y; ymax:=y; for i:=2 to n do begin write('x='); readln(x); write('y='); readln(y); if x<xmin then xmin:=x; if x>xmax then xmax:=x; if y<ymin then ymin:=y; if y>ymax then ymax:=y; end; end;`

d. Primele două puncte definesc o primă latură. Fiecare nou punct va defini o nouă latură atât timp cât nu este coliniar cu ultimele două puncte preluate. Pentru a verifica faptul că trei puncte P(x1,y1), P(x2,y2) şi P(x3,y3) sunt coliniare se poate folosi condiţia: dacă (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1)=0 atunci punctele sunt coliniare. In condiţiile în care coordonatele punctelor sunt preluate succesiv, pentru a verifica condiţia de coliniaritate este necesar să fie salvate, pe lângă coordonatele punctului curent (xcrt,ycrt) şi coordonatele ultimelor două puncte: (xant,yant), respectiv (xant2,yant2). Intrucât ultima latură poate fi determinată de P_n şi P_1 se verifică separat dacă P_{n-1}, P_n şi P_1 sunt coliniare sau nu.

Subalgoritmi pentru verificare coliniaritate şi determinare număr de laturi	Funcţie C

`dreptunghi(întreg n, real xmin, ymin, xmax ymax)`
`Intreg i`
`Real x,y`
`citire x,y`
`xmin←x; xmax←x`
`ymin←y; ymax←y`
`pentru i←2,n execută`
`  citire x,y`
`  dacă x<xmin atunci`
`  xmin←x`
`  sfârşit_dacă`
 	
`  dacă x>xmax atunci`
`  xmax←x`
`  sfârşit_dacă`
`  dacă y<ymin atunci`
`  ymin←y`
`  sfârşit_dacă`
`  dacă y>ymax atunci`
`  ymax←y`
`  sfârşit_dacă`
 	
`sfârşit_pentru`
	`int coliniar(float x1, float y1, float x2, float y2, float x3, float y3)`
	`{float eps=0.00001;`
	`  if(fabs((x2-x1)*(y3-y1)-`
	`  (y2-y1)*(x3-x1))<eps)`
	`  return(1);`
	`  else`
	`  return(0);`
	`}`
	
	`int nrLaturi(int n)`
	`{int i, nr;`
	`  float x1,y1,xcrt,ycrt;`
	`  float xant,yant,xant2,yant2;`
	`  cout<<"x1="; cin>>x1; xant=x1;`
	`  cout<<"y1="; cin>>y1; yant=y1;`
	`  cout<<"x2="; cin>>xcrt;`
	`  cout<<"y2="; cin>>ycrt;`
	`  nr=1;`
	`  for(i=3;i<=n;i++)`
	`  {xant2=xant; yant2=yant;`
	`  xant=xcrt; yant=ycrt;`
	`  cout<<"x"<<i<<"="; cin>>xcrt;`
	`  cout<<"x"<<i<<"="; cin>>ycrt;`

```
   if (coliniar
       (xant2,yant2,xant,yant,xcrt,ycrt)
       ==0)
    nr++;
  }
if(coliniar(xant,yant,xcrt,ycrt,x1,y1)==0)
nr++;
return nr;
}
```

Observaţii

- în cazul datelor de tip real valorile sunt aproximative, deci o condiţie de egalitate se specifică printr-o inegalitate: val=0 e înlocuit cu |val|<eps cu eps o valoare suficient de mică
- Program complet: P4.cpp, P4.pas

Funcţii Pascal

```
function
coliniare(x1,y1,x2,y2,x3,y3:real):boolean;
var r:boolean;
eps:real;
begin
 eps:=0.00001;
 if abs((x2-x1)*(y3-y1)-(y2-y1)*(x3-x1))<eps
 then r:=true else r:=false;
 coliniare:=r;
end;

function nrLaturi(n:integer):integer;
Var i,nr:integer;
x1,y1,xant,yant,xcrt,ycrt,xant2,yant2:real;
begin
 write('x1='); readln(x1); xant:=x1;
 write('y1='); readln(y1); yant:=y1;
 write('xcrt='); readln(xcrt);
 write('ycrt='); readln(ycrt);
 nr:=1;
 for i:=3 to n do
  begin
   xant2:=xant; yant2:=yant;
   xant:=xcrt; yant:=ycrt;
   write('x',i,'=');readln(xcrt);
   write('y',i,'=');readln(ycrt);
   if coliniare (xant2,yant2,xant,yant,
                 xcrt,ycrt)=false
   then nr:=nr+1;
  end;
 if coliniare(xant,yant,xcrt,ycrt,x1,y1)=false
 then nr:=nr+1;
 nrLaturi:=nr;
end;
```

5. Se consideră polinomul de grad n cu coeficienţi reali: $P(z)=c_nz^n + c_{n-1}z^{n-1} + \ldots + c_1z + c_0$ şi se pune problema calculului valorii polinomului pentru un număr complex z=a+bi specificat prin partea sa reală (a) şi coeficientul părţii sale imaginare (b). Valorile coeficienţilor se citesc succesiv începând cu c_n.
   Date de test: $z=2+3i$, $P(z)=2z^3-3z^2+z-6$.

Rezolvare. Structura generală a algoritmului care permite evaluarea unui polinom este:

```
v←cn
pentru i←n-1,1 cu pas -1 execută
  v←produs(v,z)
  v←suma(v,ci)
sfârşit_pentru
```

Prin urmare este suficient să se definească subalgoritmi pentru calculul sumei şi produsului a două numere complexe. Pentru două numere complexe $z_1=a_1+I\ b_1$ si $z_2=a_2+I\ b_2$ suma este $s= (a_1+a_2)+I\ (b_1+b_2)$ iar produsul este $p= (a_1a_2-b_1b_2)+I(a_1b_2+a_2b_1)$. In implementare numerele complexe pot fi specificate ca perechi de valori reale corespunzătoare părţii reale respectiv coeficientului părţii imaginare. Cele două componente pot fi implementate ca variabile aleatoare independente sau ca elemente ale unei structuri cu două câmpuri (`struct` în C, `record` în Pascal). Aici este ilustrată prima variantă.

Varianta C cu variabile reale si parametrii de tip referinţă (program P5.c)	Varianta C cu structuri (program P5v2.c)

`void suma(float areal, float aimag, float breal, float bimag, float& sreal, float& simag)`
`{sreal=areal+breal;`
`simag=aimag+bimag;}`
 	
`void produs(float areal, float aimag, float breal, float bimag, float& preal, float& pimag)`
`{preal=areal*breal-aimag*bimag;`
`pimag=areal*bimag+aimag*breal;}`
 	
`char oper(float vimag)`
`{if (vimag>0) return('+'); else return('-');}`
 	
`void evaluarePolinom()`
`{`
`float zreal, zimag, c, vreal, vimag;`
`int n,i;`
`cout<<"n="; cin>>n;`
`cout<<"z - parte reala="; cin>>zreal;`
`cout<<"z - parte imagin="; cin>>zimag;`
`cout<<"c"<<n<<"="; cin>>vreal;`
`vimag=0;`
`for(i=n-1;i>=0;i--)`
`{produs(vreal,vimag,zreal,zimag,`
`vreal, vimag);`
`cout<<"c"<<i<<"="; cin>>c;`
`suma(vreal,vimag,c,0,vreal,vimag);`
`}`
 	
`cout<<vreal<<oper(vimag)<<fabs(vimag)<<"i";`
`}`
	`typedef struct{float real; float imag;} complex;`
	`complex suma(complex a, complex b)`
	`{complex s;`
	`s.real=a.real+b.real;`
	`s.imag=a.imag+b.imag;`
	`return(s);`
	`}`
	
	`complex produs(complex a, complex b)`
	`{complex p;`
	`p.real=a.real*b.real-a.imag*b.imag;`
	`p.imag=a.real*b.imag+a.imag*b.real;`
	`return(p);`
	`}`
	
	`char oper(float vimag)`
	`{if (vimag>0) return('+');`
	`else return('-');}`
	
	`void evaluarePolinom()`
	`{`
	`complex z,v,c;`
	`float creal;`
	`int n,i;`
	`cout<<"n="; cin>>n;`
	`cout<<"z - parte reala="; cin>>z.real;`
	`cout<<"z - parte imaginara="; cin>>z.imag;`
	`cout<<"c"<<n<<"="; cin>>c.real;`
	`c.imag=0;`
	`v=c;`
	`for(i=n-1;i>=0;i--)`
	`{v=produs(v, z);`
	`cout<<"c"<<i<<"="; cin>>c.real;`
	`v=suma(v, c);`
	`}`
	
	`cout<<v.real<<oper(v.imag)<<fabs(v.imag)<<"i";`
	`}`

Variantă în Pascal

```
program P5;
var zreal, zimag, c, vreal, vimag:real;
    n,i:integer;
procedure suma(areal,aimag,breal,bimag:real; var sreal,simag:real);
begin
  sreal:=areal+breal;
  simag:=aimag+bimag;
end;
procedure produs(areal,aimag,breal,bimag:real; var preal,pimag:real);
begin
  preal:=areal*breal-aimag*bimag;
  pimag:=areal*bimag+aimag*breal;
end;
function oper(vimag:real):char;
begin
 if vimag>0 then oper:='+' else oper:='-';
end;
begin
  write('n='); readln(n);
  write('z-parte reala:'); readln(zreal);
  write('z-parte imaginara:'); readln(zimag);
  write('c',n,'='); readln(c);
  vreal:=c; vimag:=0;
  i:=n-1;
  while i>=0 do
   begin
     produs(vreal,vimag,zreal,zimag,vreal,vimag);
     write('c',i,'='); readln(c);
     suma(vreal,vimag,c,0,vreal,vimag);
     i:=i-1;
   end;
  writeln(vreal,oper(vimag),abs(vimag),'i');
end.
```

6. Se citeşte succesiv o secvenţă de n valori reale care conţine cel puţin o valoare pozitivă. Să se determine indicele de început şi indicele de sfârşit ale unei subsecvenţe de elemente consecutive care satisface:

   a. Este cea mai lungă subsecvenţă crescătoare din secvenţa dată.
   b. Suma elementelor subsecvenţei este maximă (în raport cu sumele altor subsecvenţe din secvenţă).

   Date de test: -1,2,1,-1,3,4,5,-3,1,4
   2,-3,1,3,4,-2,1,2,3,-5

Rezolvare.

a. Intrucât valorile sunt introduse succesiv, se reţin doar ultimele două valori (xant şi xcrt). Dacă valoarea curentă este mai mare decât valoarea anterioară atunci se incrementează variabila care conţine numărul curent de valori consecutive crescătoare. Dacă valoarea curentă este mai mică decât valoarea anterioară atunci se reţin valorile care identifică cea mai lungă subsecvenţă crescătoare (indicele de start şi lungimea) după care se resetează indicele de început şi lungimea curentă a unei secvenţe crescătoare.

Subalgoritm subsecventa crescatoare	Funcţie C

lgMaxima(întreg n, indMax, lgMax)
Intreg i, lgCresc, indCresc
Real x, xant
citire x
lgCresc←1; lgMax←1
indCresc←1; indMaxCresc←1
pentru i←2,n execută
  xant←x
  citire x
  dacă xant>x atunci
  dacă lgCresc>lgMax atunci
  lgMax←lgCresc;
  indMaxCresc←indCresc
  sfârşit_dacă
  lgCresc←1; indCresc←i;
  altfel
  lgCresc←lgCresc+1
  sfârşit_dacă
sfârşit_pentru
	void lgMaxima(int n,int& indMaxCresc,int& lgMax)
	{float x, xant;
	 int lgCresc;
	 int indCresc;
	 cout<<"x"<<1<<"="; cin>>x;
	 lgCresc=lgMax=1; indCresc=indMaxCresc=1;
	 for(int i=2;i<=n;i++)
	  {
	  xant=x;
	  cout<<"x"<<i<<"="; cin>>x;
	  if (xant>x)
	  {if (lgCresc>lgMax)
	  {lgMax=lgCresc;
	  indMaxCresc=indCresc;};
	  lgCresc=1; indCresc=i;}
	  else {lgCresc=lgCresc+1;}
	  }
	}

Observaţii	**Procedură Pascal**

• Indicele de start şi lungimea secvenţei maxime se returnează prin intermediul parametrilor de ieşire corespuzători; indicele de final se obţine din cel de start si lungime
• Ideea de rezolvare bazată pe o singură parcurgere poate fi aplicată şi în cazul în care valorile ar fi stocate într-un tablou
• Program complet: P6.cpp, P6.pas
	procedure lgMaxima (n:integer; var indMaxCresc, lgMax: integer);
	var x, xant:real;
	  lgCresc,indCresc,i:integer;
	begin
	 write('x1='); readln(x);
	 lgCresc:=1; lgMax:=1; indCresc:=1;
	 indMaxCresc:=1;
	 for i:=2 to n do
	 begin
	  xant:=x;
	  write('x',i,'='); readln(x);
	  if (xant>x) then
	  begin
	  if (lgCresc>lgMax) then
	  begin
	  lgMax:=lgCresc;
	  indMaxCresc:=indCresc;
	  end;
	  lgCresc:=1; indCresc:=i;
	  end
	  else lgCresc:=lgCresc+1;
	 end;
	end;

b. Se aplică o idee similară însă resetarea indicelui de start a subsecvenţei se realizează când suma devine negativă. Dacă după adăugarea elementului current suma este pozitivă atunci se incrementează indicele de final al subsecvenţei iar dacă valoarea sumei depăşeşte suma maximă atunci se reţin indicii de start şi de final ai subsecvenţei curente de sumă maximă.

Subalgoritm subsecventa cresacatoare	Funcţie C

sumaMaxima(întreg n, startMax, endMax)
Intreg i, start, end
Real x, suma, sumaMax;
citire x
dacă x<0
suma←0;
start,end,startMax,endMax←2
altfel
suma←x;
start,end,startMax,endMax←1
sumaMax ← suma
pentru i←2,n execută
citire x
dacă suma+x<0 atunci
suma←0; start←i+1
altfel
suma← suma+x; end←i
dacă suma>sumaMax atunci
sumaMax←suma
startMax←start
endMax←end
sfârşit_dacă
sfârşit_dacă
sfârşit_pentru
returnează sumaMax
	float sumaMaxima(int n, int& startMax, int& endMax)
	{int start, end;
	float x, suma, sumaMax;
	cout<<"x"<<1<<"="; cin>>x;
	if (x<0)
	{suma=0;
	start=end=startMax=endMax=2;
	}
	else
	{suma=x;
	start=end=startMax=endMax=1;
	}
	sumaMax=suma;
	for(int i=1;i<=n;i++)
	{cout<<"x"<<i<<"="; cin>>x;
	if (suma+x<0)
	{suma=0; start=i+1;}
	else
	{suma=suma+x; end=i;
	if (suma>sumaMax)
	{sumaMax=suma;
	startMax=start;
	endMax=end;
	}
	}
	}
	return sumaMax;
	}

Observaţii	Funcţie Pascal

• Functia sumaMaxima returnează indicii de început şi de sfârşit prin parametri de ieşire, iar suma maximă prin valoare de retur.
• Program complet: P6.cpp, P6.pas
	function sumaMaxima(n:integer; var startMax, endMax: integer):real;
	var x, suma, sumaMax: real;
	start, iend, i:integer;
	begin
	write('x1='); readln(x);
	if (x<0) then
	begin
	suma:=0;start:=2;iend:=2;
	startMax:=2; endMax:=2;
	end
	else
	begin
	suma:=x; start:=1; iend:=1;
	startMax:=1; endMax:=1;
	end;
	sumaMax:=suma;
	for i:=2 to n do
	begin
	write('x',i,'='); readln(x);
	if (suma+x<0) then
	begin
	suma:=0;
	start:=i+1; iend:=i+1;

```
        end
     else
       begin
        suma:=suma+x; iend:=i;
         if (suma>sumaMax) then
           begin
             sumaMax:=suma;
             startMax:=start;
             endMax:=iend;
           end
       end
   end;
  sumaMaxima:=sumaMax;
end;
```

7. Se consideră un serviciu web la care utilizatorii se conectează/deconectează şi se pune problema determinării numărului maxim de utilizatori conectaţi simultan pornind de la o secvenţă de semnale de forma: 1 (s-a conectat un utilizator), 0 (s-a deconectat un utilizator). De exemplu pentru secvenţa 1,1,1,0,1,0,1,1,1,0,0,1,0,0,0 numărul maxim de utilizatori conectaţi este 5.

Rezolvare. Ideea de rezolvare se bazează pe faptul că numărul de utilizatori este incrementat la citirea unui 1 şi decrementat la citirea unui 0. In momentul in care se intâlneşte un 0, înainte de decrementare se verifică dacă nu trebuie actualizată valoarea maximă curentă. Implementările sunt în P7.cpp respectiv P7.pas

Secvenţa C

```
int s,conect,conectMax;
int n,i;
cout<<"n="; cin>>n;
conect=conectMax=0;
for(i=1;i<=n;i++)
  {cout<<"s"<<i<<"="; cin>>s;
   if(s==1) conect++;
    else
     {if(conect>conectMax)
        conectMax=conect;
      conect--;
     }
   }
cout<<"Nr maxim="<<conectMax;
```

Secvenţa Pascal

```
var n,i,s,conect,conectMax:integer;
begin
  write('n=');readln(n);
  conect:=0; conectMax:=0;
  for i:=1 to n do
   begin
     write('s',i,'='); readln(s);
     if s=1 then conect:=conect+1
     else
      begin
       if conect>conectMax then
          conectMax:=conect;
          conect:=conect-1;
       end
   end;
  writeln('conectMax=',conectMax);
```

8. O secvenţă de n valori reale preluate succesiv (fără a fi stocate într-un tablou): $x_0,x_2,...,x_{n-1}$ trebuie „netezită” prin mediere folosind o „fereastră” de dimensiune 3, adică pornind de la valorile din secvenţă se afişează valorile $y_1,y_2,...,y_{n-2}$ care au proprietatea că $y_i=(x_{i-1}+x_i+x_{i+1})/3$.

Rezolvare. Pentru a calcula valoarea medie trebuie reţinute ultimele două valori anterioare pe lângă valoarea curentă.

Secvenţa C

```
float xant2, xant, xcrt;
int n,i;
cout<<"n="; cin>>n;
```

Secvenţa Pascal

```
var xant2, xant, xcrt: real;
    n,i:integer;
begin
```

```
cout<<"x0="; cin>>xant;
cout<<"x1="; cin>>xcrt;
for(i=2;i<n;i++)
{xant2=xant;
 xant=xcrt;
 cout<<"x"<<i<<"="; cin>>xcrt;
 cout<<"y"<<i-1<< "="<<
                  (xant2+xant+xcrt)/3;
}
```

```
 write('n='); readln(n);
 write('x0='); readln(xant);
 write('x1='); readln(xcrt);
 for i:=2 to n-1 do
  begin
   xant2:=xant;
   xant:=xcrt;
   write('x',i,'='); readln(xcrt);
   writeln('y',i-1,'=',
              (xant2+xant+xcrt)/3);
end
```

9. Se consideră relaţia de recurenţă care defineşte o secvenţă de tip Collatz:

$$C(n)=\begin{cases} n/2 & \text{daca } n \text{ este par} \\ 3n+1 & \text{daca } n \text{ este impar} \end{cases}$$

Conjectura Collatz afirmă faptul că pentru orice valoare de pornire *n*, secvenţa va ajunge la *1*. Pentru un număr natural *n* se defineşte *L(n)* ca fiind lungimea secvenţei, adică numărul de elemente din secvenţă până la întâlnirea valorii *1* (inclusiv valoarea 1). Două numere *n1* şi *n2* se consideră Collatz-echivalente dacă *L(n1)=L(n2)*. Pentru două valori naturale *a* şi *b* (*a<b*) să se determine toate perechile de numere Collatz-echivalente din *{a, a+1,...,b}*.

Rezolvare. Problema poate fi descompusă în trei subprobleme: (i) calcul C(n); (ii) calcul L(n); (iii) parcurgere perechi de valori distincte din *{a, a+1,...,b}* şi verificarea proprietăţii.

Functii C

```
int C(int n)
{if (n%2==0) return(n/2);
 else return(3*n+1);
}

int L(int n)
{int k,lg;
 k=n;
 lg=1;
 while (k!=1)
   {k=C(k); lg++;}
 return(lg);
}
int main()
{int a,b,n1,n2;
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 for(n1=a;n1<=b-1;n1++)
  for(n2=n1+1;n2<=b;n2++)
   if(L(n1)==L(n2))  cout<<"("<<n1<<
                  ","<<n2<<")"<<endl;
}
```

Functii Pascal

```
Program P9;
Var a,b,n1,n2:integer;
Function C(n:integer):integer;
Var rez:integer;
Begin
 If n MOD 2 = 0 then rez:= n DIV 2
 Else rez:=3*n+1;
 C:=rez;
End;
Function L(n:integer):integer;
Var k,lg:integer;
Begin
 k:=n; lg:=1;
 while (k<>1) do
 begin k:=C(k); lg:=lg+1; end;
 L:=lg;
End;
Begin
  Write('a='); readln(a);
  Write('b='); readln(b);
  For n1:=1 to b-1 do
    For n2:=n1+1 to b do
      If L(n1)=L(n2) then
                writeln(n1,',',n2);
End.
```

10. Se consideră o funcţie integrabilă f:[a,b]->R şi se pune problema aproximării integralei funcţiei f pe intervalul [a,b] folosind o sumă Riemann "la stânga" cu pasul h(n)=(b-a)/n: S(n)=h(n)(f(a)+f(a+h(n))+…+f(b-h(n))). Fiind date limitele intervalului să se calculeze S(n*) (n* este prima valoare pentru care |S(n*)-S(n*-1)|<0.001).

Rezolvare. Se calculează succesiv S(1), S(2), S(3) ... şi se reţin tot timpul ultima şi penultima valoare calculată. Prelucrarea se opreşte când modulul diferenţei dintre acestea devine mai mic decât 0.001. Funcţia f pentru care se calculează poate fi transmisă ca parametru (`float (*f)(float)`) în C respectiv prin definirea unui tip pentru specificarea functiilor (`type functie=function(x: real):real;`) si transmiterea adresei functiei (@numeFunctie).

Functii C

```
float f1(float x)
{return (x);}

float f2(float x)
{return (sin(x)+cos(x));}

float S(int n, float (*f)(float),float
a, float b)
{float h, suma, x;
 int i;
 suma=0;
 h=(b-a)/n;
 x=a;
 for(i=1;i<=n;i++)
   {suma=suma+(*f)(x);
    x=x+h;
    }
 suma=h*suma;
 return suma;
}

float                  sumaRiemann(float
(*f)(float),float a, float b)
{float h, x, sant, scrt, eps=0.00001;
 int i, n=1;
 h=(b-a)/n;
 scrt=h*f(a);
 do {sant=scrt;
     n++;
     scrt=S(n,f,a,b);
     } while (fabs(scrt-sant)>=eps);
  return (scrt);
}

int main()
{float a,b,rez;
 cout<<"a="; cin>>a;
 cout<<"b="; cin>>b;
 cout<<sumaRiemann(f1,a,b)<<endl;
}
```

Functii Pascal

```
program P10;
type  functie=function(x:  real):real;
{tip functie}
var a,b,rez:real;
function f1(x:real):real;
begin
  f1:=x;  {functia f1(x)=x}
end;

function f2(x:real):real;
begin
  f2:=sin(x)+cos(x);          {functia
f2(x)=sin(x)+cos(x)}
end;

function     S(n:integer;   f:functie;
a:real; b:real):real;
var h, suma, x:real;
    i: integer;
begin
 suma:=0;
 h:=(b-a)/n;
 x:=a;
 for i:=1 to n do
   begin
     suma:=suma+f(x);
     x:=x+h;
   end;
 suma:=h*suma;
 S:=suma;
end;

function        sumaRiemann(f:functie;
a:real; b:real):real;
var h, sant, scrt, eps:real;
    n:integer;
begin
  eps:=0.00001;
  n:=1;
  h:=(b-a)/n;
  scrt:=h*f(a);
  repeat
     sant:=scrt;
```

```
      n:=n+1;
      scrt:=S(n,f,a,b);
   until abs(scrt-sant)>=eps;
   sumaRiemann:=scrt;
end;

begin
 write('a='); readln(a);
 write('b='); readln(b);
 write(sumaRiemann(@f1,a,b));
end.
```