Botezatu Marius, clasa 11-”D”

Tehnici de elaborare a algoritmilor

Iterativitate sau recursivitate

Cuprins

Introducere

- In timpul dezvoltarii informaticii s-a stabilit faptul ca multe probleme de o importanta practica pot fi rezolvate cu ajutorul unor metode standard, denumite tehnici de programare: recursia,trierea, s.a.
- Una din cele mai des intalnite tehnica de programare este recursia, adica autoapelarea unui subprogram fie de el insusi, fie de alt subprogram. Definirea corecta a unui algoritm recursiv presupune ca in procesul derularii calculelor trebuie sa exista: 1)cazuri elementare, care se rezolva direct.

2)cazuri care nu se rezolva direct, insa procesul de calcul in mod obligatoriu progreseaza spre un caz elementar.

- Orice algoritm recursiv poate fi transcris intr-un algoritm iterativ si invers.
- Iterativitatea este procesul prin care rezultatul este obţinut ca urmare a execuţiei repetate a unui set de operaţii, de fiecare dată cu alte valori de intrare. Numărul de iteraţii poate fi necunoscut sau cunoscut, dar determinabil pe parcursul execuţiei.

Dezavantajele si avantajele metodei iterative/recursive

Nr.crt.	Caracteristici	Iterativitate	Recursivitate
1.	Necesarul de memorie	mic	mare
2.	Timpul de executie	acelasi	acelasi
3.	Structura programului	complicata	simpla
4.	Volumul de munca necesar pentru scrierea programului	mare	mic
5.	Testarea si depanarea programului	simpla	complicata

5 probleme folosind subprograme iterative

1)Acest program sorteaza numerele in ordine crescatoare folosind algoritmul “bubble sort”. Spre exemplu, daca ii dam numerele 4,9,1,3 , el le va sorta si va afisa 1,3,4,9.

```
type
  array1 = array[1..500] of integer;

var
  a: array1;
  i, n: integer;

procedure sortare(a: array1; n: integer);
var
```

```
  i, j, m: integer;
begin
  writeln('Cate numere sunt?'); readln(n);
  for i := 1 to n do
    readln(a[i]);   //citeste toate numerele si le pune intr-un array
  for i := 1 to n do     //pentru a sorta toate numerele in ordine crescatoare programul e nevoit sa treaca prin toate numerele de n-1 ori, de n ori.
    for j := 1 to n - 1 do    //subprogramul trece prin toate numerele de n-1 ori
      if a[j] > a[j + 1] then  //compara primul numar cu al doilea, daca al doilea e mai mic atunci programul le schimba cu locul.
      begin
        m := a[j + 1];
        a[j + 1] := a[j];  //are loc schimbarea primului numar cu al doilea
        a[j] := m;
      end;
  writeln();
  for i := 1 to n do
    write(a[i], ' ');
end;

begin
  sortare(a, n);
End.
```

2) Acest program verifica daca numarul introdus este prim sau nu.

```
var
  numar: integer;
```

```
function prime(numar1: integer): boolean;
var
  i: integer;
begin
  for i := 2 to numar1 div 2 do
    if numar1 mod i = 0 then begin  //daca restul este 0 atunci numarul nu este prim, prime devine false  si va iesi din subprogram
      prime := false;
      exit;
    end;
  prime := true;  //daca a ajuns pana aici inseamna ca numarul e prim
end;

begin
  readln(numar);
  if prime(numar) = false then writeln('Numarul nu este prim')
  else writeln('Numarul este prim');
End.
```

3) Acest program calculeaza numarul literelor de fiecare tip intr-un string. Spre exemplu daca dam stringul: “marius”, programul va afisa “m=1,a=1,r=1,i=1,u=1,s=1”.

```
type
  alfabet = array[1..26] of integer;  //fiecare “celula” al arrayului este o litera,conform alfabetului, array[1] este a, array[2] este b, array[3] este c, etc. Initial toate aceste “celule” sunt 0. Adica array[1]=0, array[2]=0,etc.

var
  s: string;
```

```
  a: alfabet;


procedure calcul;
var
  i: integer;
begin
  readln(s);
  for i := 1 to length(s) do
  begin
    if ((ord(s[i]) <= 122) and (ord(s[i]) >= 97)) or ((ord(lowcase(s[i])) <= 122) and
(ord(lowcase(s[i])) >= 97)) then  //se uita daca caracterul din string este o litera
      a[ord(s[i]) - 96] := a[ord(s[i]) - 96] + 1;  //ord(s[i])-96  calculeaza locul literei s[i] in alfabet si
adauga la locul corespunzator in array +1. Spre exemplu, daca litera data este “a” atunci el va
adauga la array[1]+1. Daca este “b” el va adauga la array[2]+1.
  end;
  for i := 1 to 26 do
    if a[i] <> 0 then writeln(chr(i + 96), '=', a[i]);  //aici el afiseaza pe ecran toate celulele arrayului
care nu sunt = 0.
end;


begin
  calcul;
End.
```

4) Acest program creeaza o lista unidimensionala cu 3 noduri

```
type
  adresacelula = ^celula;
  celula = record
```

```
    a: string;
    b: adresacelula;
  end;

var
  k, m, n: adresacelula;

procedure afisare;
begin
while k<>nil do begin
  writeln(k^.a); //afisarea listei
  k:=k^.b;
  end;
end;

begin
  new(k); new(m);
  k:=nil;
  m^.a := 'marius';
  m^.b := nil;
  k := m;
  n:=m;
  new(m);
  m^.a := 'botezatu';
  m^.b := nil;       //aici pursisimplu cream nodurile
  n^.b:=m;
  n:=m;
```

```
  new(m);
  m^.a := 'salut';
  m^.b := nil;
  n^.b := m;

  afisare;
End.
```

5) Un program care afiseaza toate numerele pare pana la “n”.

```
var
  n: integer;

procedure panalan;
var
  i: integer;
begin
  for i := 1 to n do
    if i mod 2 = 0 then write(i, ' '); //daca numarul mod 2 = 0 inseamna ca e par, il afiseaza
end;

begin
  readln(n);
  panalan;
End.
```

5 programe folosind subprograme recursive

1) Acest program contine un subprogram recursiv care calculeaza suma S(n)=1+3+5+…+(2*n-1)

```
var
  numar: integer;

function recursiv(numar1: integer): integer;
begin
  if numar1 = 1 then recursiv := 1
  else recursiv := (2 * numar1 - 1) + recursiv(numar1 - 1); //functia se autoapeleaza pana cand numar = 1
end;

begin
  readln(numar);
  writeln(recursiv(numar));  //afisam rezultatul
End.
```

2) Cu ajutorul subprogramului noi aflam numerele lui Fibonacci

```
var
  numar: integer;

function recursiv(n: integer): integer;
begin
  if n = 0 then recursiv := 0
  else if n = 1 then recursiv := 1
  else recursiv := recursiv(n - 1) + recursiv(n - 2);  //Fiecare numar Fibonacci este suma celor doua numere Fibonacci anterioare
end;
```

```
begin
  readln(numar);
  writeln(recursiv(numar));  //afisam rezultatul
End.
```

3) Acest program contine un subprogram recursiv care calculeaza produsul P(n) = 1*4*7*…*(3*n-2)

```
var
  numar: integer;

function recursiv(n: integer): integer;
begin
  if n = 1 then recursiv := 1
  else recursiv := (3 * n - 2) * recursiv(n - 1);
end;

begin
  readln(numar);
  writeln(recursiv(numar));  //afisam rezultatul
End.
```

4) Subprogramul recursiv din acest program ne creeaza un array si afiseaza rezultatele pe ecran

```
type
  array1 = array[1..100] of integer;

var
  numar: integer;
  a: array1;

procedure recursiv(n: integer);
```

```
begin
  if n = 1 then begin
    writeln('a[1]=');
    readln(a[1]);
  end
  else begin
    writeln('a[', n, ']='); //afisam pe ecran valorile fiecarei celule din array
    readln(a[n]);  //aici  introducem valorile
    recursiv(n - 1);
  end;

end;

begin
  readln(numar);  //introducem cat de mare este arrayul
  recursiv(numar);
End.
```

5) Subprogramul din acest program inverseaza un sir de caractere

```
var
  string1: string;
  n,nn: integer;

procedure recursiv(var string2: string; n1,n2: integer);
var
  c1: char;
```

```
begin
  c1 := string2[n1];
  if n1 = (length(string2) div 2) then exit  //daca n1 = (length(string2) div 2) atunci stringul este inversat si putem iesi din subprogram
  else begin
    c1 := string2[n1];
    string2[n1] := string2[n2];
    string2[n2] := c1;  //aici schimbam cu locul caracterele din string de pe pozitiile n1 si n2
    recursiv(string2, n1 - 1,n2+1)
  end;

end;

begin
nn:=1;
  readln(string1);  //introducem stringul pe care dorim sa ill inversam
  n := length(string1);
  recursiv(string1, n,nn);
  writeln(string1); //afisam pe ecran stringul inversat
End.
```

Concluzii

- Ambele tehnici de programare , cea iterativa si cea recursiva, au diferite caracteristici ce le disting. Cele mai semnificative, pe care le putem observa uitandu-ne la programele de mai sus este faptul ca ,de regula, programele iterative sunt mai voluminoase si mai grele de priceput. Insa, acest lucru nu inseamna ca trebuie sa neglijam folosirea tehnicii iterative deoarece pentru unele probleme folosirea ei este mult mai convinabila decat folosirea tehnicii recursive. In plus, necesarul de memorie este mai mare pentru tehnica recursiva, iar testarea si depanarea programelor este mai complicata. Deci putem spune ca ambele tehnici au propriile avantaje si dezavantaje, iar alegerea celei mai bune variante variaza de la caz la caz.

Bibliografie

- Manualul de informatica pentru clasa 11-a