Probleme consultații 25 noiembrie 2017
Șiruri (tablouri unidimensionale)

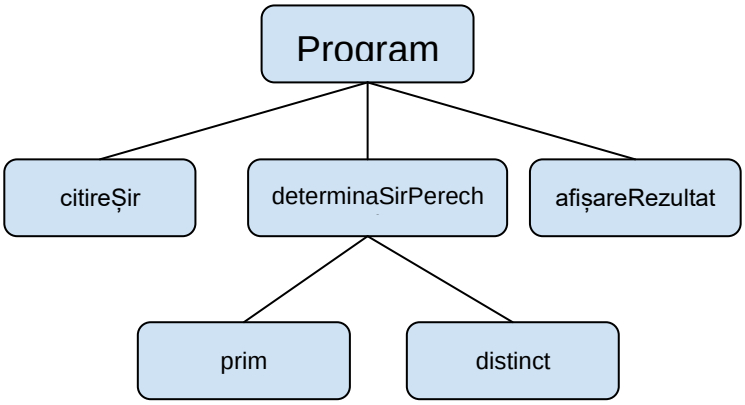
Problema 1

Enunț

Twin primes. Se citesc mai multe numere naturale, până la introducerea numărului 0 și se memorează într-un sir. Să se găsească toate perechile de numere prime gemene distincte care se regăsesc în șirul dat. Un număr prim *p* este considerat prim geamăn (*twin prime*) dacă *p+2* sau *p-2* este de de asemenea prim. De exemplu {3,5}, {5,7} și {11,13} sunt perechi de numere prime gemene.

Analiză

Identificarea subalgoritmilor



Specificarea subalgoritmilor

A. Subalgoritmul citireȘir(a, n):
Descriere: Citește elementele unui șir până la întâlnirea numărului 0.
Date: -
Rezultate: a - un șir de elemente nenule de lungime n.
$$n \in N, a = \{ a_i | \ i = 1..n, a_i \ \in N, a_i \neq 0\}.$$

B. Funcția prim(p):
Descriere: Verifică dacă numărul *p* este prim.
Date: *p* - număr natural, $p \in N$
Rezultate: true dacă numarul *p* este prim, false altfel.

C. Functia distinct(x, y, rez, m):
Descriere: Verifica dacă elementele x și y sunt deja înregistrate ca fiind o pereche de numere gemene prime.
Date: x, y - numerele pe care le verificăm, $x \in N, y \ \in N$
rez, m - șirul cu perechile de numere gemene prime de lungime *m*, $m \in N$
Rezultate: true dacă perechea nu face parte din șir, false altfel.

D. Subalgoritmul determinăȘirPerechi(a, n, rez, m):
Descriere: Determină șirul care conține perechile de numere gemene prime.
Date: a, n - șirul de elemente nenule de lungime n, $n \in N$
$$a = \{ a_i | \ i = 1..n, a_i \ \in N, a_i \neq 0\}$$
Rezultate: rez, m - șirul cu perechile de numere gemene prime de lungime *m*, $m \in N$
$$rez = \{ (rez_i, \ rez_{i+1}) | \ i = 1,3,5,\dots,m-1, m \in N, rez_i + 2 = rez_{i+1}, rez_i, rez_{i+1} \in N \ - \ prime\}$$

E. Subalgoritmul afișareRezultat(rez, m):
Descriere: Afișează perechile de numere gemene prime din șirul *rez* de lungime *m*.
Date: rez - șirul de perechi de numere gemene prime de lungime *m*.
$rez = \{ rez_i | \ i = 1 \ldots m, rez_i \in N, m \in N \}$
Rezultate: se afișeaza prerechile de numere gemene prime din șirul *rez* dacă *m>0*. Sau mesajul: "Nu s-a gasit nici o pereche!" dacă *m=0*.

Proiectare

```
Subalgoritmul citireȘir(a, n) este:
        n <- 0
        @citeste nr
        cat-timp nr <> 0 executa
                n <- n + 1
                a[n] <- nr
                @citeste nr
        sf-cat-timp
sf-subalgoritm

Functia prim(p) este:
    daca n <=1 atunci
        prim <- false
    sd-daca
    daca n = 2 sau n = 3 atunci
        prim <- true
    sf-daca
    i = 2
    cat-timp i <= p/2 executa          // v1.1 i <= sqrt(p)
        daca p mod i = 0 atunci
            prim <- false
        sf-daca
        i = i + 1
    sf-cat-timp
    prim <- true
sf-functie

Functia prim_v2(p) este:
    daca n <= 1 atunci
        prim <- false
    sd-daca
    daca n <= 3 atunci
        prim <- true
    sf-daca
    daca n mod 2 = 0 sau n mod 3 = 0 atunci  // eliminam multiplii de 2 si 3
        prim <- false
     sf-daca
     i = 5
     cat-timp i * i <= p
        daca p mod i = 0 sau p mod (i+2) = 0 atunci
            prim <- false
```

```
        sf-daca
        i = i + 6
    sf-cat-timp
    prim <- true
sf-functie

Functie distinct(x, y, rez, m) este:
    pentru i <- 1, m-1, pas 2 executa // din 2 in 2 elemente
        daca (rez[i] = x si rez[i+1] = y) sau (rez[i+1] = x si rez[i] = y) atunci
            distinct <- false
        sf-daca
    sf-pentru
    distinct <- true
sf-functie

Functie distinct_v2(x, y, rez, m) este:
 i <- 0
 gasit <- false
 cat-timp  i < n - 1 si not gasit executa
  daca (r[i] == x si r[i + 1] == y) sau (r[i + 1] == x si r[i] == y) atunci
    gasit <- true // am gasit perechea deja memorata
  sf-daca
  i <- i + 2 // verificam urmatoarea pereche
 sf-cat-timp
 distinct <- not gasit
sf-functie

Subalgoritmul determinaSirPerechi(a, n, rez, m) este:
  m <- 0 // presupunem ca nu avem nici un numar prim
  pentru i <- 1, n executa
    daca prim(a[i]) atunci
      pentru j <- i+1, n atunci
        daca (a[j] = a[i] + 2 sau a[j] = a[i] - 2) si distinct(a[i], a[j], rez, m) si prim(a[j]) atunci
          // verificam prima data daca a[j] este mai mare sau mai mic cu doua unitati decat a[i]
          // apoi testam daca nu cumva avem aceasta pereche inregistrata
          // si la final verificam daca a[j] este prim.
          m <- m +2
          rez[m] <- a[j]
          rez[m-1] <- a[i]
        sd-daca
      sf-pentru
    sf-daca
  sf-pentru
sf-subalgoritm

Subalgoritmul afisareRezultat(rez, m) este:
  daca m = 0 atunci
    @tipareste “Nu s-a gasit nici o pereche!”
  altfel
    pentru i <- 1,m-1, pas 2 executa
```

```
        @tipareste ‘{’ rez[i], ‘,’ rez[i+1], ‘ } ’
      sf-pentru
   sf-daca
sf-subalgoritm

Algoritmul TwinPrimes este:
   citireSir(a,n)
   determinaSirPerechi(a,n,rez,m)
   afisareRezultat(rez,m)
sf-algoritm
```

Exemple

A. Pentru șirul: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 0
   Programul afișează următoarele perechi:
   {3,5} {5,7} {11,13} {17,19}
B. Pentru șirul: 139 100 107 2 101 139 11 12 103 137 0
   Se afișează perechile:
   {137, 139} {101, 103}
C. Pentru: 11 19 109 311 463 937 1163 2017 2019 0
   Se afișează următorul mesaj:
   Nu s-a gasit nici o pereche!

Implementare c++

```
#include <iostream>
#include <math.h>
using namespace std;

void citireSir(int a[], int &n){
  cout << "Introduceti numerele: ";
  int x;
  n = 0;
  cin >> x;
  while (x > 0) {
    a[n++] = x;
    cin >> x;
  }
}

void afisareRezultat(int r[], int n) {
  if (n == 0) {
    cout << "Nu s-a gasit nici o pereche!" << endl;
  }
  else {
    for (int i = 0; i < n - 1; i += 2) {
      cout << '{' << r[i] << ',' << r[i + 1] << "} ";
    }
    cout << endl;
  }
}
```

```
bool distinct(int x, int y, int r[], int n) {
  for (int i = 0; i < n - 1; i += 2) {
    if ((r[i] == x && r[i + 1] == y) || (r[i + 1] == x && r[i] == y)) {
      return false;
    }
  }
  return true;
}

bool distinct_v2(int x, int y, int r[], int n) {
  int i =0;
  bool gasit = false;
  while (i < n - 1 && !gasit) {
    if ((r[i] == x && r[i + 1] == y) || (r[i + 1] == x && r[i] == y)) {
      gasit = true;
    }
    i+=2;
  }
  return !gasit;
}

bool prim(int p){
  if (p <= 1)
    return false;
  if (p == 2 || p == 3)
    return true;
  int i = 2;
  while (i <= sqrt(p)) {
    if (p % i == 0)
      return false;
    i++;
  }
  return true;
}

void determinaSirPerechi(int a[], int n, int rez[], int &m){
  m = 0;
  for (int i = 0; i < n; i++) {
    if (prim(a[i])) {
      for (int j = i + 1; j < n; j++) {
        if ((a[i] == a[j] + 2 || a[i] == a[j] - 2) && distinct(a[i], a[j], rez, m) && prim(a[j])) {
          rez[m++] = a[i] < a[j] ? a[i] : a[j];
          rez[m++] = a[i] > a[j] ? a[i] : a[j];
        }
      }
    }
  }
}
```

```
int main() {
  int a[1000], rez[1000], n, m;
  citireSir(a, n);
  determinaSirPerechi(a, n, rez, m);
  afisareRezultat(rez, m);
}
```

Implementare pascal

```
Program TwinPrime;
uses Math;
type Sir = array[1..1000] of Integer;
Var a,rez: Sir;
    n,m: Integer;
procedure citireSir(var a: Sir; var n: Integer);
var x: Integer;
begin
  Write('Introduceti numerele: ');
  n:=0;
  Read(x);
  while (x>0) do
  begin
    n:=n+1;
    a[n]:=x;
    Read(x);
  end;
end;

procedure afisareRezultat(a: Sir; n: Integer);
var i: Integer;
begin
  if (n = 0) then
    writeln('Nu s-a gasit nici o pereche!')
  else
    begin
      i:=1;
      while (i<n) do
      begin
        write('{',a[i],',',a[i+1],'} ');
        i := i+2;
      end;
      writeln();
    end;
end;
```

```
function distinct(x, y: Integer; r: Sir; n: Integer): Boolean;
var i: Integer;
   lipseste: Boolean;
begin
 i := 1;
 lipseste := true;
 while (i < n-1) and lipseste do
 begin
  if ((r[i]=x) and (r[i+1]=y)) or ((r[i+1]=x) and (r[i]=y)) then
    lipseste := false;
  i := i + 1;
 end;
 distinct := lipseste;
end;

function prim(p: Integer): Boolean;
var i: Integer;
   gasit: Boolean;
begin
 gasit := true;
 if (p <= 1) then
  gasit := false;
 if (p = 2) or (p = 3) then begin
  gasit := true;
 end else begin
  i:=2;
  while (i<=sqrt(p)) and gasit do
  begin
   if (p mod i = 0) then
    gasit := false;
   i := i+1;
  end;
 end;
 prim := gasit;
end;

procedure determinaSirPerechi(a: Sir; n: Integer; var rez: Sir; var m: Integer);
var i,j: Integer;
begin
 m := 0;
 for i := 1 to n-1 do
  if (prim(a[i])) then
   for j := i+1 to n do
    if (((a[i]=a[j]-2) or (a[i]=a[j]+2)) and
      distinct(a[i],a[j], rez, m) and
      prim(a[j])) then
      begin
       m := m+1;
       if (a[i]<a[j]) then begin
        rez[m] := a[i];
```

```
            m := m+1;
            rez[m] := a[j];
          end else begin
            rez[m] := a[j];
            m := m+1;
            rez[m] := a[i];
          end;
        end;
end;

begin
  citireSir(a,n);
  determinaSirPerechi(a,n,rez,m);
  afisareRezultat(rez,m);
end.
```

Problema 2

Enunț

Tribonacci. Se citesc mai multe numere naturale, până la introducerea numărului 0 și se memorează într-un sir. Să se determine subșirul de lungime maximă care reprezintă un subșir din seria tribonacci. Seria tribonacci este o generalizare a seriei fibonacci cu deosebirea că fiecare termen este suma celor trei termeni precedenți.

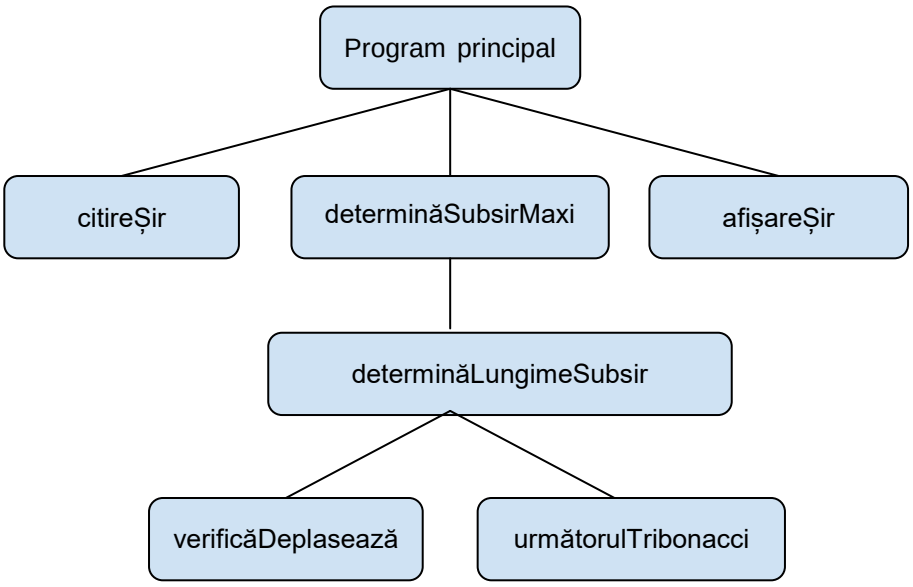
Termenii secvenței tribonacci sunt definiți astfel:

$T_0 = 0, T_1 = T_2 = 1, T_n = T_{n-1} + T_{n-2} + T_{n-3}, unde\ n \geq 3.$

Începutul secvenței este: 0, 1, 1, 2, 4, 7, 13, 24, 44, 81, 149, ...

Analiză

Identificarea subalgoritmilor



Specificarea subalgoritmilor

Subalgoritmul citireȘir(a, n):

Descriere: Citește elementele unui șir până la întâlnirea numărului 0.

Date: -

Rezultate: a - un șir de elemente nenule de lungime n.

$$n \in N, a = (a_1, a_2, \ldots, a_n), unde\ \ a_i \in N, a_i \neq 0,\ i = 1 \ldots n.$$

Subalgoritmul determinăLungimeSubsir(a, n, index):

Descriere: Determină lungimea unei subsecvențe de numere tribonacci începând de la poziția index din șirul dat.

Date: a, n - șirul de elemente nenule de lungime n, $n \in N$

$$a = (a_1, a_2, \ldots, a_n), unde\ \ a_i \in N, a_i \neq 0, \qquad i = 1 \ldots n.$$

index - poziția din șirul a, de la care să înceapă căutarea subsecvenței.

Rezultate: lungimea subșir sau 0 dacă nu exista un astfel de subșir.

Subalgoritmul verificăDeplasează(val, tribonacciVal, index, len):

Descriere: Determină dacă valoarea găsită este o valoare tribonacci, în caz afirmativ incrementează lungimea si deplasează indexul curent cu o unitate.

Date: val - valoarea din sir.

tribonacciVal - valoarea curenta din seria tribonacci.

index - pozitia curenta din sir.

len - lungimea curenta.

Rezultate: true daca valorile se potrivesc, false altfel.

index - noua pozitie in sir.

len - noua lungime.

Subalgoritmul următorulTribonacci(tn3,tn2,tn1,tn):

Descriere: Determină următoarele valori din subsir.

Date: tn3, tn2, tn1, tn - valorile curente.

Rezultate: tn3, tn2, tn1, tn - valorile deplasate cu o poziție,

$t_n = t_{n-1} + t_{n-2} + t_{n-3}$

$t_{n-3} = t_{n-2};\ t_{n-2} = t_{n-1};\ t_{n-1} = t_n$

Subalgoritmul determinăSubsirMaxim(a, n, rez, m):

Descriere: Determină subșirul de lungime maximă care este de asemenea subșir în seria tribonacci.

Date: a, n - șirul de elemente nenule de lungime n, $n \in N$

$$a = (a_1, a_2, \dots, a_n), unde\ a_i \in N, a_i \neq 0, \qquad i = 1 \dots n.$$

Rezultate: rez, m - subșirul de lungime m, din șirul a, care de asemenea este subșir tribonacci, $m \in N$

$rez = (rez_1, rez_2, \dots, rez_m)\ unde\ \exists k \in N, a.i. rez_1 = T_k, rez_2 = T_{k+1}, rez_3 = T_{k+2}\ si\ rez_i = rez_{i-1} + rez_{i-2} + rez_{i-3},\quad i = 4 \dots m, m \in N, si\ T_k\ -\ aparțin\ secvenței\ tribonacci$

Subalgoritmul afișareȘir(rez, m):

Descriere: Afișează șirul *rez* de lungime *m*.

Date: rez - subșirul de lungime *m*, parte din seria tribonacci.

$rez = (rez_1, rez_2, \dots, rez_m), rez_i \in N, m \in N$

Rezultate: se elementele șirul *rez* dacă *m>0*. Sau mesajul: “Nu s-a gasit nici un subsir!” dacă *m=0*.

Proiectare

```
Subalgoritmul citireȘir(a, n) este:
    n <- 0
    @citeste nr
    cat-timp nr <> 0 executa
        n <- n + 1
        a[n] <- nr
        @citeste nr
    sf-cat-timp
sf-subalgoritm

Subalgoritmul afisareRezultat(rez, m) este:
  daca m = 0 atunci
    @tipareste “Nu s-a gasit nici un subsir!”
  altfel
    pentru i <- 1, m executa
      @tipareste rez[i]
    sf-pentru
  sf-daca
sf-subalgoritm
```

```
Subalgoritmul următorulTribonacci(t0, t1, t2, tn) este:
  tn <- t2+t1+t0;
  t0 <- t1;
  t1 <- t2;
  t2 <- tn;
sf-subalgoritm

Funcția verificăDeplasează(val, tribonacciVal, index, len)
 daca val = tribonacciVal atunci
   index <- index +1
   len <- len +1
   verificăDeplasează <- true
 sd-daca
 verificăDeplasează <- false
sf-funcție

Subalgoritmul determinaLungimeSubsir(a, n, index)
  len <- 0 // presupunem ca nu avem nici un subsir valid
  t0 <- 0
  t1 <- t2 <- 1
  verificăDeplasează(a[index], t1, index, len)
  verificăDeplasează(a[index], t2, index, len)
  următorulTribonacci(t0, t1, t2, t3)
  cat-timp a[index] > t3 si not verificăDeplasează(a[index], t3, index, len) executa
    următorulTribonacci(t0, t1, t2, t3) // sarim peste numerele mai mici
   sf-cat-timp
  cat-timp a[index] = t3 si verificăDeplasează(a[index], t3, index, len) executa
    următorulTribonacci(t0, t1, t2, t3) // numaram cate numere se regasesc in serie
   sf-cat-timp
  determinaLungimeSubsir <- len
sf-subalgoritm

Subalgoritmul determinaSubsirMaxim(a, n, rez, m)
  i <- 0
  m <- 0  // sirul rezultat initial este vid
  maxLen <- 0 // lungimea maxima gasita
  indiceStart <- 0 // pozitia de unde incepe subsirul maxim
  cat-timp i<n executa
    len <- determinaLungimeSubsir(a, n, i)
    daca len > maxLen atunci
       maxLen <- len
       indiceStart <- i
    sf-daca
    daca len > 2 atunci
       i <- i + len
    altfel
       i <- i + 1
    sf-daca
  sf-cat-timp
```

```
  daca maxLen > 0 atunci
    m <- maxLen
    pentru i = 1..m executa
      rez[i] <- a[i + indiceStart]
    sp-pentru
  sf-daca
sf-algoritm

Algoritmul Tribonacci este:
  citireSir(a,n)
  determinaSubsirMaxim(a,n,rez,m)
  afisareSir(rez,m)
sf-algoritm
```

Exemple

A. Pentru valorile: 1 2 3 4 5 6 7 8 9 10 0
   Programul afișează următorul mesaj:
   Subsirul de lungime maxima este: 1 2 len: 2
B. Pentru șirul: 1 2 3 1 1 1 1 1 2 4 7 13 24 44 81 149 274 1 1 2 4 7 0
   Se afișează:
   Subsirul de lungime maxima este: 1 1 2 4 7 13 24 44 81 149 274 len: 11
C. Pentru șirul: 1 2 3 1 1 1 1 1 2 7 13 24 44 81 149 274 1 1 2 4 7 0
   Se afișează:
   Subsirul de lungime maxima este: 7 13 24 44 81 149 274 len: 7
D. Pentru: 100 101 102 103 1000 1001 2017 0
   Se afișează următorul mesaj:
   Nu s-a gasit nici un subsir!

Implementare c++

```
#include <iostream>
using namespace std;

void citireSir(long a[], int &n) {
  cout << "Introduceti numerele: ";
  long x;
  n = 0;
  cin >> x;
  while (x > 0) {
    a[n++] = x;
    cin >> x;
  }
}
```

```
void afisareSir(long a[], int n){
  if (n == 0) {
    cout << "Nu s-a gasit nici un subsir!" << endl;
  } else {
    cout << "Subsirul de lungime maxima este: ";
    for (int i = 0; i < n; i++) {
      cout << a[i] << ' ';
    }
    cout << "len: " << n << endl;
  }
}

void urmatorulTribonacci(long &t0, long &t1, long &t2, long &t3){
  t3 = t2 + t1 + t0;
  t0 = t1;
  t1 = t2;
  t2 = t3;
}

bool  verificaDeplaseaza(long value, long tribonacciValue, int &index, int &len){
  if (value == tribonacciValue) {
    index++;
    len++;
    return true;
  }
  return false;
}

int determinaLungimeSubsir(long a[], int n, int index){
  int l = 0;
  long t0, t1, t2, t3;
  t0 = 0;  t1 = t2 = 1;
  verificaDeplaseaza(a[index], t1, index, l);
  verificaDeplaseaza(a[index], t2, index, l);
  urmatorulTribonacci(t0, t1, t2, t3);
  while (index<n && a[index] > t3 && !verificaDeplaseaza(a[index], t3, index, l)){
    urmatorulTribonacci(t0, t1, t2, t3);
  }
  while (index<n && a[index] == t3 && verificaDeplaseaza(a[index], t3, index, l)){
    urmatorulTribonacci(t0, t1, t2, t3);
  }
  return l;
}

void constuiesteTribonacci(long t[], int tn){
  t[0] = 0;
  t[1] = t[2] = 1;
  for (int i = 3; i < tn; i++)
    t[i] = t[i - 1] + t[i - 2] + t[i - 3];
}
```

```
int determinaLungimeSubsir_v2(long a[], int n, int index){
  int start = index;
  int tn = 50;
  long t[tn];
  constuiesteTribonacci(t, tn);
  int i=0;
  while (i<tn && a[start] > t[i]){
    i++;
  }
  while (start<n && i<tn && a[start] == t[i]){
    i++;
    start++;
  }
  return start - index;
}

void determinaSubsirMaxim(long a[], int n, long rez[], int &m){
  int i = 0;
  m = 0;
  int maxLen = 0;
  int indiceStart = 0;
  while (i < n) {
    int len = determinaLungimeSubsir(a, n, i);
    cout << "a[" << i << "]:" << a[i] << " len: " << len << endl;
    if (len > maxLen) {
      maxLen = len;
      indiceStart = i;
    }
    if (len > 2)
      i += len;
    else
      i++;
  }
  if (maxLen > 0) {
    m = maxLen;
    for (i = 0; i < m; i++) {
      rez[i] = a[i + indiceStart];
    }
  }
}


int main(){
  long a[1000], rez[1000];
  int n, m;
  citireSir(a, n);
  determinaSubirMaxim(a, n, rez, m);
  afisareSir(rez, m);
}
```

Implementare pascal

```
Program Tirbonacci;
type Sir = array[1..1000] of Longint;
Var a,rez: Sir;
   n,m: Integer;

procedure citireSir(var a: Sir; var n: Integer);
var x: Integer;
begin
  Write('Introduceti numerele: ');
  n:=0;
  Read(x);
  while (x>0) do
  begin
    n:=n+1;
    a[n]:=x;
    Read(x);
  end;
end;

procedure afisareSir(a: Sir; n: Integer);
var i: Integer;
begin
  if (n = 0) then
    writeln('Nu s-a gasit nici un subsir!')
  else
    begin
      write('Subsirul de lungime maxima este: ');
      for i := 1 to n do
      write(a[i],' ');
      writeln(' len: ',n);
    end;
end;

procedure urmatorulTribonacci(var t0,t1,t2,t3: Longint);
begin
  t3 := t2 + t1 + t0;  t0 := t1;  t1 := t2;  t2 := t3;
end;

function verificaDeplaseaza(val, tribonacciVal: Longint; var index,len: Integer): Boolean;
var gasit: Boolean;
begin
  gasit := false;
  if (val = tribonacciVal) then begin
    index := index + 1;
    len := len + 1;
    gasit := true;
```

```
  end;
  verificaDeplaseaza := gasit;
end;

function determinaLungimeSubsir(a: Sir; n, index: Integer): Integer;
var t0,t1,t2,t3: Longint;
    len: Integer;
    gasit: Boolean;
begin
  len := 0;  t0 := 0;  t1 := 1;  t2 := 1;
  verificaDeplaseaza(a[index], t1, index, len);
  verificaDeplaseaza(a[index], t2, index, len);
  urmatorulTribonacci(t0, t1, t2, t3);
  while (index<n) and (a[index] > t3) and not verificaDeplaseaza(a[index], t3, index, len) do begin
    urmatorulTribonacci(t0, t1, t2, t3);
  end;
  while (index<n) and (a[index] = t3) and verificaDeplaseaza(a[index], t3, index, len) do begin
    urmatorulTribonacci(t0, t1, t2, t3);
  end;
  determinaLungimeSubsir := len;
end;

procedure construiesteTribonacci(var t: Sir; n: Integer);
var i: Integer;
begin
  t[1] := 0;  t[2] := 1;  t[3] := 1;
  for i := 4 to n do
    t[i] := t[i-1] + t[i-2] + t[i-3];
end;

function determinaLungimeSubsir_v2(a: Sir; n, index: Integer): Integer;
var i: Integer;
    start: Integer;
    lenTribonacci: Integer;
    tribonacci: Sir;
begin
  start := index;
  lenTribonacci := 50;
  construiesteTribonacci(tribonacci, lenTribonacci);
  i := 1;
  while (i<lenTribonacci) and (a[start] > tribonacci[i]) do begin
    i := i + 1;
  end;
  while (start < n) and (i<lenTribonacci) and (a[start] = tribonacci[i]) do begin
    i := i + 1;
    start := start + 1;
  end;
  determinaLungimeSubsir_v2 := start - index;
end;
```

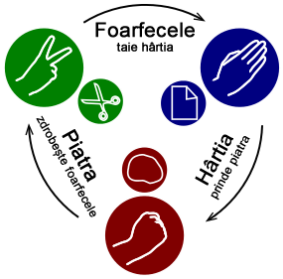
```
procedure determinaSubsirMaxim(a: Sir; n: Integer; var rez: Sir; var m: Integer);
var i, maxLen, indiceStart, len: Integer;
begin
  i := 1;
  m := 0;
  maxLen := 0;
  indiceStart := 0;
  while (i <= n) do
  begin
    len := determinaLungimeSubsir_v2(a,n,i);
    writeln('a[',i,']: ',a[i],' len: ', len);
    if (len > maxLen) then begin
      maxLen := len;
      indiceStart := i;
    end;
    if (len > 2) then begin
      i := i + len;
    end else
      i := i + 1;
  end;
  if (maxLen > 0) then begin
    m := maxLen;
    for i := 1 to m do
      rez[i] := a[i + indiceStart - 1];
  end;
end;

begin
  citireSir(a,n);
  determinaSubsirMaxim(a,n,rez,m);
  afisareSir(rez,m);
end.
```

Problema 3

Enunț

Problema “Piatra, hartie, foarfeca”

Piatră, hârtie, foarfecă („jan-ken-pon” în limba japoneză, „rochambeau” în limba franceză) este o metodă de tragere la sorți, în care se folosesc gesturi făcute cu mâna pentru a decide cine a câștigat. Cei care participă (de obicei 2) spun „Piatră!” „Hârtie!” „Foarfecă!” de două ori, dar a doua dată, când spun „foarfecă” (sau în loc de a spune cuvântul „foarfecă”), fiecare jucător face unul din următoarele gesturi cu mâna: un pumn, o palmă deschisă sau extinzând degetul arătător și cel lung ca o foarfecă. Dacă ambii jucători au același gest, se repetă până gesturile diferă. Piatra fiind în stare să strice foarfeca, va câștiga dacă aceste două gesturi au fost făcute. Dacă s-a făcut foarfecă și hârtie, câștigă foarfeca (căci poate tăia hârtia), iar dacă se face piatră și hârtie, câștigă hârtia, căci poate împacheta piatra.



Sa se simuleze jocul “Piatra, hartie, foarfeca” printr-un algoritm cu urmatoarele restrictii:

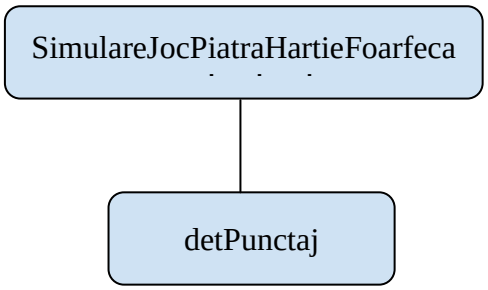
1) Joc intre 2 jucatori.
2) Un joc dureaza 3 runde (cu gesturi diferite).
3) La fiecare runda jucatorul castigator primeste 1 punct.
4) Punctele se cumuleaza la cele 3 runde. Castiga jucatorul cu punctajul maxim.
5) Mapare valori in algoritm: piatra=0, hartie=1, foarfeca=2.
6) Valorile celor 2 jucatori la fiecare runda se citesc de la tastatura sau se determina aleator (0,1 sau 2). Presupunem ca exista un al treilea jucator care are rol de albitru iar acesta introduce datele in aplicatie doar daca cei 2 jucatori au respectat regulile jocului (ambii arata deodata semnul ales).

Dupa terminarea jocului se va afisa pe ecran cine este jucatorul castigator!

Analiză

Identificarea subalgoritmilor

- sublgoritm care determina si incrementeaza punctajul jucatorului castigator la o runda
- algoritmul care simuleaza jocul, citeste valorile celor doi jucatori si stabileste la fiecare runda jucatorul castigator al rundei, iar la final castigatorul jocului.



Specificarea subalgoritmilor

Subalgoritmul detPunctaj(rundaJ1, rundaJ2:Integer; var pJ1, pJ2:Integer);
Descriere: determina care dintre valorile celor doi jucatori este castigatoare, incrementand cu 1 valoarea pJ1 sau pJ2
Date: rundaJ1, rundaJ2, pJ1, pJ2
Rezultate:pJ1,pJ2

- rundaJ1, rundaJ2 – reprezinta valoarile “aratate” de cei doi jucatori
- pJ1, pJ2 reprezinta valorile punctajelor celor doi jucatori inainte de runda curenta, valori ce vor fi modificate in functie de valorile arata de cei doi jucatori. Vezi mai jos regulile de acordare punctaj castigator.

Codificare in program: piatra = 0 , hartie=1, foarfeca =2
Reguli de acordare punctaj castigator la o runda

piatra >foarfeca ==> 0>2 => pJ1++
piatra < hartie ==> 0<1 ==> pJ2++

foarfeca >hartie ==> 2>1 => pJ1++
foarfeca <piatra ==> 2<0 => pJ2++

hartie <foarfeca ==> 1<2 => pJ2++
hartie >piatra ==> 1>0 => pJ1++

Proiectare

Subalgoritm detPunctaj(rundaJ1, rundaJ2, pJ1, pJ2) este:

```
Daca(rundaJ1=0) atunci
   Daca(rundaJ2=2) atunci
      pJ1←pJ1+1
   altfel
      Daca(rundaJ2=1)atunci
         pJ2←pJ2+1
       sfDaca
   sfDaca
altfel
  Daca(rundaJ1=2) atunci
   Daca (rundaJ2=1) atunci
      pJ1←pJ1+1
   altfel
      Daca(rundaJ2=0)atunci
          pJ2←pJ2+0
      sfDaca
  sfDaca
 altfel
  Daca (rundaJ1=1) atunci
   Daca(rundaJ2=2) atunci
      pJ2←pJ2+1
   altfel
      Daca(rundaJ2=0)atunci
         pJ1←pJ1+1
     sfDaca
  sfDaca
sfDaca
sfSubalgoritm;
```

```
Algoritmul SimulareJocPiatraHartieFoarfeca este:
   rundaJ1←-1;   rundaJ2←-1;  nRunde←0;
   pJ1←0;pJ2←0;
   incaJoaca←true;   numarRunda←1;

   CatTimp(incaJoaca) executa
        Afiseaza('Dati valoare jucator 1:'); Citeste(rundaJ1);
        Afiseaza('Dati valoare jucator 2:'); Citeste(rundaJ2);
        CatTimp(rundaJ1=rundaJ2) executa
            Afiseaza('Dati valoare jucator 1:');
            Citeste(rundaJ1);
            Afiseaza('Dati valoare jucator 2:');
            Citeste(rundaJ2);
          SfCatTimp
         detPunctaj(rundaJ1,rundaJ2,pJ1,pJ2);
         nRunde←nRunde+1;
         if(nRunde=3) then
            incaJoaca← false;
sfCatTimp
   if(pJ1>pJ2) then
      Afiseaza('A castiga J1')
   else
      Afiseaza('A castiga J2');
end.
```

Exemple

A. Pentru datele introduse: 0 2 2 1 1 0
   Programul afișează:
   "A castigat J1"

B. Pentru datele introduse: 0 1 2 0 1 2
   Programul afișează:
   "A castigat J2"

C. Pentru datele introduse: 0 2 2 1 0 1
   Programul afișează:
   "A castigat J1"

Implementare – varianta C++

```
#include <iostream>
using namespace std;

void detPunctaj(int rundaJ1, int rundaJ2, int  &pJ1, int &pJ2){
 if(rundaJ1==0)
   if(rundaJ2==2)
      pJ1=pJ1+1;
   else
      if(rundaJ2==1)
         pJ2=pJ2+1;
 else
  if(rundaJ1==2)
   if (rundaJ2==1)
```

```
      pJ1=pJ1+1;
   else
      if(rundaJ2==0)
         pJ2=pJ2+0;
 else
  if (rundaJ1==1)
   if(rundaJ2==2)
      pJ2=pJ2+1;
   else
      if(rundaJ2==0)
         pJ1=pJ1+1;
}

int main(){
   bool incaJoaca;
   int rundaJ1, rundaJ2, numarRunda, pJ1,pJ2, nRunde;
   rundaJ1=-1;   rundaJ2=-1;   nRunde=0;   pJ1=0;pJ2=0;  incaJoaca=true;   numarRunda=1;
   while(incaJoaca)  {
        cout<<"\n Dati valoare jucator 1:";
        cin>>rundaJ1;
        cout<<"\n Dati valoare jucator 2:";
        cin>>rundaJ2;
        while(rundaJ1==rundaJ2){
           cout<<"\n Dati valori diferite ale jucatorilor:";
           cout<<"\n Dati valoare jucator 1:";
           cin>>rundaJ1;
           cout<<"\n Dati valoare jucator 2:";
           cin>>rundaJ2;
         }
         detPunctaj(rundaJ1,rundaJ2,pJ1,pJ2);
         nRunde=nRunde+1;
         if(nRunde==3)
           incaJoaca= false;
   }// while

   if(pJ1>pJ2)
      cout<<"\n A castiga J1";
   else
      cout<<"\n A castiga J2";

   return 0;
}
```

Implementare – variant Pascal

```
Program PiatraHartieFoarfeca;
var incaJoaca:Boolean;
   rundaJ1, rundaJ2, numarRunda, pJ1,pJ2, nRunde:Integer;

procedure detPunctaj(rundaJ1, rundaJ2:Integer; var pJ1, pJ2:Integer);
begin
 if(rundaJ1=0) then
   if(rundaJ2=2) then
      pJ1:=pJ1+1
   else
      if(rundaJ2=1) then
         pJ2:=pJ2+1
```

```
 else
  if(rundaJ1=2) then
   if (rundaJ2=1) then
      pJ1:=pJ1+1
   else
      if(rundaJ2=0) then
          pJ2:=pJ2+0
 else
  if (rundaJ1=1) then
   if(rundaJ2=2) then
      pJ2:=pJ2+1
   else
      if(rundaJ2=0) then
         pJ1:=pJ1+1
end;

begin
   rundaJ1:=-1;
   rundaJ2:=-1;
   nRunde:=0;
   pJ1:=0;pJ2:=0;
   incaJoaca:=true;
   numarRunda:=1;

   while(incaJoaca)do
      begin
         writeln('Dati valoare jucator 1:'); Read(rundaJ1);
        writeln('Dati valoare jucator 2:'); Read(rundaJ2);
        while(rundaJ1=rundaJ2) do
         begin
            writeln('Dati valoare jucator 1:');
            readln(rundaJ1);
            writeln('Dati valoare jucator 2:');
            readln(rundaJ2);
         end;
         detPunctaj(rundaJ1,rundaJ2,pJ1,pJ2);
         nRunde:=nRunde+1;
         if(nRunde=3) then
            incaJoaca:= false;
      end;
   if(pJ1>pJ2) then
      writeln('A castiga J1')
   else
      writeln('A castiga J2');
end.
```

Probleme tip grilă

1) Se dă următorul subalgoritm:

```
Subalgoritm sumaInteresanta(a, n):
    k = 0
    Pentru i ← 1, n execută:
        Daca (i modulo 2 = 0) atunci
            k ← k + a[i]
        SfDaca
    SfPentru
    returnează k
SfSubalgoritm
```

Precizati care dintre secvențele de instrucțiuni de mai jos va produce afisarea numarului 12.
Observație: S-a presupus că indexarea șirurilor începe de la 1.

a. a= [1,2,3,4,5,6], n=6 (corect)
   nr = sumaInteresanta (a, n)
   afișare(nr)
b. a= [6,5,4,3,2,1], n=6
   nr = sumaInteresanta (a, n)
   afișare(nr)
c. a= [2,1,4,3,6,5], n=6
   nr = sumaInteresanta (a, n)
   afișare(nr)
d. nu există un astfel de apel.

2) Se consideră funcția de mai jos.

```
Functia apare(a,n,e) este:
    Daca (n=0) atunci
        apare ← false
    altfel
        Daca a[n]=e atunci
            apare ←true
        altfel
            apare ← apare(a,n-1,e)
        sfDaca
    sfDaca
sfFunctia
```

I. De câte ori se apelează funcția pentru a = [3,4,5,6,7,8], n= 6, e = 2?

a. De 2 ori
b. De 6 ori
c. De 4 ori
d. De 7 ori (corect)

II. Care este rezultatul functiei pentru urmatoarele date:

a. a=[2,4,6,8], n=4, e=6 (T)
b. a=[2,4,6,8], n=4, e= 8 (T)
c. a=[2,4,6,8], n=4, e= 2 (T)
d. a[2,4,6,8], n=4, e=1 (F)

3) Se dă următoarul subalgoritm cu specificarea de mai jos:
Descriere: (de completat de catre elev) ……..
Date de intrare: a, n
Date de iesire: b, m

```
Subalgoritm detNumereInteresante(a, n, b, m):
        m = 0
        Pentru i ← 1, n execută:
                Daca (a[i] modulo 2 = 0) atunci
                        m ← m+1
                        b[m] ← i
                SfDaca
        SfPentru
        returnează k
SfSubalgoritm
```

I. Care este rezultatul execuției apelului **detNumereInteresante**(x,l,y,k) pentru l=4 și x=[2,1,4,3].
   a. k = 2 și y=[2, 4];
   b. Eroare de compilare
   c. k = 2 și y=[1, 3]; (corect)
   d. k = 0 și y=[];

II. Propuneri un set de date de intrare astfel incat in urma apelului **detNumereInteresante**(x,l,y,k) sa se obtina setuld e date de iesire k=4 și y=[1,2,3,4].
Exemplu: l=4, x=[2,2,2,2]

4) Se dă următorul subalgoritm care insereaza pe pozitia pos un element elem. Modificati corpul acestui subalgoritm astfel incat inserarea sa se realizeze:
   a) Dupa pozitia pos. (linia 3, conditia i> (pos+1))
   b) Inainte de pozitia pos. (linia 3, conditia i>= pos)

Subalgoritmul **inserareElem**(a,n,pos, elem) este:

```
1.      n ←n + 1
2.      i ←  n
3.      cat-timp (i > pos) executa
4.              a[i] ←a[i - 1]
5.              i ← i - 1
6.      sf-cat-timp
7.      a[i] ←elem
sf-subalg
```

Probleme tema

Problema 1

Se citesc nr numere naturale care reprezinta punctele accumulate (valori intre 0 si 10) la un concurs (pe pozitia i sunt memorate concurentului ai i-lea din concurs). Sa se determine:

a) Concurentul/concurentii care au primit punctajul cel mai mare.
b) Pentru a imbunatatii punctajul concurentilor se cere cooperarea lor la urmatoarea proba cu un concurent din editiile anterioare, astfel:
    a. adaugarea dupa fiecare punctaj <5 un concurent cu punctaj de valoare 10.
    b. adaugarea dupa fiecare punctaj intre [5,7] un concurent cu punctaj de valoare 8.
    c. adaugarea dupa fiecare punctaj >7un concurent cu punctaj de valoare <5.

Observatie:

a) in 2 variante (se calculeaza maximul si apoi toti cu maxim, sau se calculeaza maxim la parcurgere si memorare)
b) inserarea dupa o pozitia data – functie apelata de trei ori la cele 3 subcerinte

Problema 2

Alte cerinte problema 1

d. Sa se determine daca in sirul initial exista perechi de concurenti vecini cu acelasi numar de puncte acumulate.
e. Sa se determine numarul de triplete (elemente vecine din sir) care au la mijloc un concurent cu punctaj mare (9 sau 10) si la “capete” concurenti cu note mici (valori sub 5).
f. Sa se determine numarul de concurenti a caror punctaj a crescut dupa noua proba. (concurentii primesc tot valori intre 0 si 10).
g. Cat la suta dintre concrentii initiali si-au imbunatatit punctajul?

Nota: *Implementarile de pascal au fost validate cu versionea disponibila online aici: https://www.tutorialspoint.com/compile_pascal_online.php iar implementarile de c++ folosind clang-900.0.37*