VARIANTA 11 - *Rezolvare*

Subiectul 1

1.	Numerele cu aceeași paritate dau același rest la împărțirea cu **2**. Prima variantă este incorectă pentru că valorile **a div 2**, **b div 2** și respectiv **a/2**, **b/2** corespund câturilor la împărțirea cu **2**. Un contraexemplu este dat de **a=4**, **b=5**, pentru care expresia e adevărată deși numerele au parități diferite. Varianta **b** este cea corectă. Expresia de la varianta **c** are valoarea **true/1** dacă și numai dacă cel puțin unul dintre numerele reținute de **a** și **b** este par. Contraexemple: pentru **a=1**, **b=2** are valoarea **true/1**, iar pentru **a=3**, **b=5** are valoarea **false/0.** Expresia de la varianta **d** este totdeauna falsă dacă **a** și **b** au aceeași paritate. Pentru precizarea răspunsului **b)** se acordă **4p**.
2.	Înaintea oricărei afișări, **f(3)** apelează **f(1)**. În apelul **f(1)** nu se va intra în bucla **for**, prin urmare nu se vor face alte apeluri recursive; la final se va afișa pe ecran **0**. La revenirea din apelul **f(1)**, în **f(3)** se va afișa **i**, adică **1**, apoi se va trece la următorul pas din **for** și se va apela **f(2)**. Bucla **for** din **f(2)** se va executa doar o dată. În cadrul acesteia se va apela **f(1)**, în cadrul căruia se va afișa **0**, apoi la revenire se va afișa **1**, iar la final se va mai afișa **n**, adică din nou **1**. Apoi se va reveni în apelul **f(3)**, care va afișa valoarea curentă a lui **i**, adică **2**, iar la final, după încheierea buclei, pe cea curentă a lui **n**, adică tot **2** și se va încheia. Prin urmare șirul afișat este **0101122**. Pentru răspunsul **a)** se acordă **4p**.
3.	Conform precizărilor din enunț, pe fiecare poziție în parte se încearcă mai întâi plasarea unei paranteze deschise, iar apoi, dacă acest lucru nu mai e posibil, a uneia închise. Astfel, pentru **n=6** se generează în ordine soluțiile: **((()))**, **(()())**, **(())()**, **()(())**, **()()()**. A patra soluție este **()(())**. Pentru răspunsul **a)** se acordă **4 p.**
4.	Numărul maxim de muchii pe care le poate avea un graf neorientat cu **10** vârfuri care **nu** este conex este **36**. La acest rezultat se ajunge păstrând un vârf izolat și luând toate muchiile posibile care nu îl au ca extremitate. Prin urmare, pentru a avea certitudinea că graful este conex trebuie să avem cel puțin **1+36=37** de muchii. Pentru răspunsul **c)** se acordă **4p.**
5.	Pentru fiecare grup de cel mult **4** noduri aflate pe nivelul **2** al arborelui este necesar să avem un tată al acestora pe nivelul **1**. În plus, pe nivelul **0** vom avea rădăcina. Putem construi un arbore cu **7** noduri pe nivelul **2**. Vectorul de tați ar putea fi **(0,1,1,2,2,2,2,3,3,3)**. Putem de asemenea demonstra că e imposibil să avem mai mult de **7** noduri pe ultimul nivel. Dacă am lua, de exemplu, **8** noduri pe nivelul **2**, ar mai fi necesare cel puțin **2** noduri pe nivelul **1** și un nod pe nivelul **0** (rădăcina), deci **11** noduri în total. Pentru răspunsul **b)** se acordă **4p.**

Subiectul 2

1.	**a)**	Algoritmul are ca efect afișarea celui mai mare pătrat perfect din intervalul **[a,b]**. Dacă intervalul nu conține niciun astfel de număr, atunci se va afișa **0**. În intervalul **[14,20]** există un singur pătrat perfect, **16**. Aceasta va fi valoarea afișată. Pentru precizarea valorii **16** se acordă **6p**.
	b)	Pentru **a=30**, valorile posibile pentru **b** sunt **31, 32, 33, 34, 35**, deoarece **36** este pătrat perfect și dacă **b** ar fi mai mare strict decât **35**, în intervalul dat ar exista cel puțin un pătrat perfect, prin urmare nu s-ar mai afișa **0.** Rezultatul poate fi găsit printr-o abordare exhaustivă: se încearcă pe rând pentru **b** primele valori mai mari strict ca **a**. Atunci când se ajunge la **36**, se constată că pentru orice valoare mai mare sau egală a lui **b**, nu se va mai putea afișa **0**. Rezultatul poate fi calculat imediat dacă se înțelege prelucrarea din interiorul buclei **pentru**: la sfârșitul lui **cât timp**, **j** va fi cel mai mic număr natural al cărui pătrat e mai mare sau egal cu **i**. Condiția **j*j=i** corespunde de fapt verificării dacă i este pătrat perfect. Pentru precizarea valorilor **31, 32, 33, 34, 35**, se acordă **6p**.
	c)	Un algoritm echivalent cu cel dat este următorul: calculăm **r**=cel mai mare număr care ridicat la pătrat nu depășește **b**. Dacă **p=r*r** e mai mare sau egal cu **a**, atunci **p** va fi rezultatul afișat, altfel scriem **0**. **citește a,b (numere naturale nenule, a<b)** **r←1** **cât timp r*r≤b execută** **r←r+1** **r←r-1** **dacă r*r≥a atunci** **p←r** **altfel** **p←0** **scrie p** Pentru verificarea proprietății de a fi pătrat perfect a numărului determinat se acordă **1 p**., pentru verificarea dacă numărul găsit aparține intervalului **[a,b]** se acordă **1p.**, pentru tratarea cazului în care intervalul nu conține pătrate perfecte se acordă **1p**., iar pentru folosirea corectă a unei singure structuri repetitive se mai acordă **3p.** – în total **6p**.
	d)	Se testează capacitatea de a reprezenta pe hârtie algoritmul dat cu ajutorul unui limbaj de programare studiat. Deși scrierea programelor pe hârtie este o activitate improprie la informatică, prin această cerință se valorifică experiențele anterioare de implementare și testare a programelor pe calculator, experiențe care

		au cimentat cunoștințele privind structura programelor, declararea variabilelor, sintaxa instrucțiunilor programului, regulile de scriere a expresiilor etc. Pentru instrucțiunile corecte de declarare a variabilelor, de citire a datelor, de afișare a rezultatului și de decizie se acordă câte **1p.**, pentru cele două instrucțiuni repetitive se acordă **3p.**, iar pentru cele **4** atribuiri se acordă **2p.** Pentru structura corectă a programului se mai acordă **1p.**, în total **10p**.

2.	Pentru ca **x** să aparțină intervalului **i** este necesar ca **x** să respecte simultan condițiile **x>=i.a** și **x<=i.b**. Secvența de instrucțiuni cerută este:
	
	Limbajul Pascal
	`if (x>=i.a) and (x<=i.b) then`
	`  write('DA')`
	`else`
	`  write('NU')`
	
	Limbajul C/C++
	`if(x>=i.a && x<=i.b)`
	`  printf("DA");`
	`  | cout<<"DA";`
	`else`
	`  printf("NU");`
	`  | cout<<"NU";`
	
	Pentru accesarea corectă a câmpurilor înregistrării se acordă **4** puncte (câte un punct pentru fiecare câmp).
	Pentru verificarea condiției de apartenență la interval se acordă **1** punct.
	Pentru afișarea mesajelor cerute se acordă **1** punct.
	În total, pentru o secvență de instrucțiuni care respectă cerința se acordă toate cele **6 puncte**.

3.	Pentru a completa secvența de instrucțiuni conform cerinței, trebuie observat că o variantă corectă de inițializare a lui **min** este de a-i atribui valoarea primului element al coloanei **j**, **a[0,j]** (respectiv **a[0][j]**). Elementele coloanei **j** sunt de forma **a[i,j]** (respectiv **a[i][j]**). După ce se calculează rezultatul specific fiecărei coloane, reținut în variabila **min**, se actualizează **s**, care crește cu valoarea lui **min**.
	
	Limbajul Pascal
	`s:=0;`
	`for j:=0 to n-1 do`
	`  begin`
	`  min := a[0,j];`
	`  for i:=1 to m-1 do`
	`  if a[i,j] < min`
	`  then`
	`  min:=a[i, j];`
	`  s:=s+min`
	`  end;`
	`writeln(s);`
	
	Limbajul C/C++
	`s=0;`
	`for (j=0; j<n; j++){`
	`  min=a[0][j];`
	`  for (i=1; i<m;i++)`
	`  if (a[i][j]<min)`
	`  min=a[i][j];`
	`  s=s+a[i][j];`
	`}`
	`printf("%d", s);`
	`  | cout << s;`
	
	Se acordă câte **2** puncte pentru fiecare completare corectă a punctelor de suspensie, înlocuirea cu **a[i,j]** (respectiv **a[i][j]**) fiind punctată o singură dată – în total **6 puncte**.

Subiectul 3

1. **a)** Pentru a calcula partea întreagă inferioară a lui radical din **n**, care este de fapt rezultatul pe care trebuie să îl returneze funcția **radical**, este suficientă o căutare secvențială a numărului cerut. Mai precis, vom găsi cel mai mic număr **r**, al cărui pătrat depășește **n** (cât timp **r*r** e prea mic, îl incrementăm pe **r**) și vom returna **r-1** (care este cu siguranță cel mai mare număr natural al cărui pătrat e mai mic sau egal cu **n**).

Problema admite o soluție mai eficientă, bazată pe căutarea binară, dar aceasta nu e recomandată în condițiile unui examen scris, implementarea fiind ceva mai dificilă.

Pentru antet corect se acordă **1 punct**, pentru determinarea rezultatului cerut se acordă **2 puncte**,iar pentru transmiterea (returnarea) acestuia **1 punct** – în total **4 puncte**.

Limbajul Pascal	Limbajul C/C++

`function radical(n:longint): longint;`
`var r:longint;`
`begin`
`  r:=0;`
`  while r*r<=n do`
`  inc(r);`
`  radical:=r-1;`
`end;`
	`int radical(int n)`
	`{`
	`  int r=0;`
	`  while(r*r<=n)`
	`  r++;`
	`  return r-1;`
	`}`

b) Programul principal cerut poate fi scris foarte ușor folosind un apel al funcției cerute la punctul **a)**. Se reține în variabila **rad** rezultatul lui **radical(x)** și apoi, pentru a trece de la partea întreagă inferioară la cea superioară, se incrementează **rad** dacă **x** nu este pătrat perfect. Rezultatul cerut este **rad*rad**.

Limbajul Pascal	Limbajul C/C++

`var x,rad:longint;`
 	
`begin`
`  read(x);`
`  rad:=radical(x);`
`  if rad*rad<x then`
`  inc(rad);`
`  write(rad*rad)`
`end.`
	`#include <stdio.h>`
	`int main()`
	`{`
	`  int x,rad;`
	`  scanf("%d",&x);`
	`  rad=radical(x);`
	`  if(rad*rad<x)`
	`  rad++;`
	`  printf("%d",rad*rad;`
	`  return 0;`
	`}`

Pentru declararea corectă a variabilelor se acordă **1** punct, pentru operațiile de intrare/ieșire se acordă **1** punct, pentru un apel corect și util al funcției de la punctul **a)** se acordă **2 puncte**, pentru determinarea rezultatului cerut (inclusiv tratarea cazului particular în care **x** este pătrat perfect) se acordă **1** punct, iar pentru corectitudinea globală a programului se acordă **1 punct** – în total **6 puncte**.

2. Există bineînțeles mai multe variante de rezolvare. Cea pe care o prezentăm mai jos are avantajul că poate fi adaptată ușor și altor cerințe asemănătoare legate de prelucrarea cuvintelor dintr-un text.

Limbajul Pascal:

```
var s:string;
    i,p,u:integer;

function litera(c:char):boolean;
var mica,mare:boolean;
begin
  mica:=((ord(c)>=ord('a'))and(ord(c)<=ord('z')));
  mare:=((ord(c)>=ord('A'))and(ord(c)<=ord('Z')));
  litera:=(mica or mare)
end;

function vocala(c:char):boolean;
var voc:string;
begin
  voc:='AEIOUaeiou';
  vocala:=(pos(c,voc)<>0)
end;

function literamica(c:char):boolean;
begin
  literamica:=((ord(c)>=ord('a'))
                and(ord(c)<=ord('z')))
end;

function majuscula(c:char):char;
var nr:integer;
begin
  nr:=ord(c)-ord('a');
  majuscula:=char(ord('A')+nr)
end;

begin
  readln(s);
  for i:=1 to length(s) do
    if litera(s[i]) then begin
        if (i=1)or(not litera(s[i-1]))
          then p:=i;
        if (i=length(s))or(not litera(s[i+1]))
          then begin
            u:=i;
            if (vocala(s[p]))and(vocala(s[u])) then
              while p<=u do begin
                  if literamica(s[p])
                    then s[p]:=majuscula(s[p]);
                  inc(p)
                end
          end
      end;
    writeln(s)
end.
```

Limbajul C/C++:

```
#include <stdio.h>
#include <string.h>
```

```
int literamica(char c){
  if(c>='a' && c<='z')
    return 1;
  return 0;
}
int litera(char c){
  if(c>='a' && c<='z')
    return 1;
  if(c>='A' && c<='Z')
    return 1;
  return 0;
}
int vocala(char c){
  char voc[]="AEIOUaeiou";
  if(strchr(voc,c)!=NULL)
    return 1;
  return 0;
}
char majuscula(char c){
  return c-'a'+'A';
}
int main(){
  int n,i,p,u;
  char s[101];
  gets(s);
  for(i=0;s[i]!=0;i++)
    if(litera(s[i])){
      if(i==0||!litera(s[i-1]))
        p=i;
      if(!litera(s[i+1])){
        u=i;
        if(vocala(s[p])&&vocala(s[u]))
          while(p<=u){
            if(literamica(s[p]))
              s[p]=majuscula(s[p]);
            p++;
          }
      }
    }
  puts(s);
  return 0;
}
```

Atât în limbajul **Pascal**, cât și în **C/C++** putem proceda în felul următor: parcurgem șirul de caractere **s** și, pentru fiecare cuvânt, reținem în variabila de tip întreg **p** poziția la care începe fiecare cuvânt, și în variabila **u,** poziția la care se termină.

Vom prelucra doar caracterele de tip literă. Condiția de a ne afla la începutul unui cuvânt este ca înaintea caracterului curent să nu se afle o literă, sau caracterul curent să fie primul din întregul șir. Condiția de a ne afla pe ultima poziție din cuvânt este ca după caracterul curent să nu se mai afle o literă sau să fim la sfârșitul șirului de caractere. Practic, interpretăm cuvintele ca fiind secvențe continue formate exclusiv din litere.

Apoi, în cazul în care cuvântul curent este unul dintre cele care trebuie prelucrate, facem acest lucru (în cazul nostru transformăm în majuscule) de la **p** la **u**. Această strategie generală poate fi adaptată aproape la orice problemă care cere prelucrarea cuvintelor. Nu depinde nici de numărul de separatori, nici de tipul lor, nici de existența unor restricții legate de începutul sau sfârșitul textului.

		Se acordă **1 punct** pentru declararea corectă a variabilelor, **1 punct** pentru citirea datelor (trebuie avut în vedere faptul că șirul de caractere poate conține spații), **1 punct** pentru afișarea textului modificat. Se acordă **2 puncte** pentru parcurgerea șirului de caractere, **2 puncte** pentru identificarea cuvintelor care trebuie transformate și **2 puncte** pentru modificarea acestora (aceste 2 puncte se acordă numai dacă șirul de caractere este transformat în memorie, conform cerinței, nu și pentru varianta în care rezultatul cerut este doar afișat). Pentru corectitudinea globală a programului se acordă **1 punct** – în total **10 puncte.**
4.	**b)**	Pentru a reține șirul numerelor citite de pe cea de-a doua linie a fișierului vom utiliza un vector **v**, care va fi ordonat crescător (conform precizării din enunț). Apoi vom parcurge vectorul de la stânga la dreapta și pentru fiecare indice **i** vom determina cel mai mare **j** cu proprietatea că **v[i]+v[j]≤x**. Având în vedere faptul că vectorul **v** este ordonat crescător, rezultă că pentru orice **k** natural, cu **i<k≤j**, avem **v[i]+v[k]≤x**, deci numărul de perechi cu proprietatea cerută, pentru care **v[i]** este primul termen (mai mic) este **j-i**; acesta se va aduna la variabila **nr**, în care vom păstra rezultatul final. Ne vom opri atunci când **i** va ajunge mai mare sau egal cu **j**, pentru că de la această valoare a lui **i** încolo nu va trebui să mai adunăm nimic la **nr** (**v[i]** a devenit prea mare). Eficiența algoritmului, ca timp de executare, constă în faptul că **j** va face o singură parcurgere de la stânga la dreapta. Ținând cont de faptul că **v** este ordonat crescător, observăm că pe măsură ce **i** crește, **j** trebuie să scadă (sau eventual să rămână pe loc). Deci, atât **i**, cât și **j** fac cel mult o parcurgere completă a șirului. Pentru o descriere coerentă a metodei se acordă **1 punct**, iar pentru justificarea eficienței, încă **1 punct** – în total **2 puncte.** Menționăm că descrierea coerentă a problemei nu presupune să povestim programul, ci ideea de prelucrare. În acest sens e mai bine să explicăm că "parcurgem", "numărăm", "căutăm primul element care …" etc. decât să dăm detalii tehnice cum ar fi "folosim un **for**", "incrementăm variabila întreagă **t**…", etc.

	a)	**Limbajul Pascal**
		`var n,x,i,j,nr:longint;`
		`  v:array[1..100000] of longint;`
		`  fin:text;`
		
		`begin`
		`  assign(fin,'bac.in');`
		`  reset(fin);`
		`  readln(fin,n,x);`
		`  for i:=1 to n do`
		`  read(fin,v[i]);`
		`  close(fin);`
		`  nr:=0;`
		`  i:=1;`
		`  j:=n;`

```
  while i<j do
    begin
      while (j>i)and(v[j]+v[i]>x) do
        dec(j);
      nr:=nr+j-i;
      inc(i)
    end;
  write(nr)
end.
```

Limbajul C/C++

```
#include <stdio.h>
int main()
{
  FILE *fin;
  int n,x,v[100000],i,j,nr;
  fin=fopen("bac.in","r");
  fscanf(fin,"%d%d",&n,&x);
  for(i=0;i<n;i++)
    fscanf(fin,"%d",&v[i]);
  fclose(fin);
  nr=0;
  i=0;
  j=n-1;
  while(i<j)
  {
    while(j>i&&v[j]+v[i]>x)
      j--;
    nr+=j-i;
    i++;
  }
  printf("%d",nr);
  return 0;
}
```

Se acordă **1 punct** pentru operaţiile cu fişiere (declarare, nume corect şi deschidere pentru citire), încă **1 punct** pentru citirea **tuturor** numerelor din fişier, **3 puncte** pentru un algoritm principial corect, şi **1 punct** pentru corectitudinea formală (declararea variabilelor, structura programului, sintaxa instrucțiunilor, etc.).

Se acordă **2 puncte** pentru alegerea unui algoritm eficient ca timp de executare (O(**n**)) – în total **8 puncte**.

Menționăm că prin *algoritm principial* corect se înțelege un algoritm care, în intenție, urmărește să rezolve corect problema, deși poate să prezinte scăpări logice sau confuzii.