SUBIECTUL I (20 de puncte)

Pentru fiecare dintre itemii de la 1 la 5, scrieți pe foaia de examen litera corespunzătoare răspunsului corect. Fiecare răspuns corect se notează cu 4 puncte.

1. Variabila `x` este de tip întreg și memorează un număr nenul. Indicați expresia C/C++ cu valoarea `1` pentru orice multiplu al lui `2020` memorat în variabila `x`.

a. `x/(x/2020)==0` b. `x/(x%2020)==0` c. `x%(x%2020)==0` **d.** `x%(x/2020)==0`

2. Subprogramul `f` este definit alăturat. Indicați ce se afișează în urma apelului de mai jos.

`f(2,20);`

```
void f(int x, int y)
{  if(x<y)
     f(2*x-1, y-1);
   cout<<x+y<<" "; | printf("%d ", x+y);
}
```

a. `22 22 23 26` b. `22 22 23 26 33` c. `26 23 22 22` **d.** `33 26 23 22 22`

3. Utilizând metoda backtracking se generează toate posibilitățile de a forma liste de câte `3` locuri izolate distincte din lume, din mulțimea `{Hanging, Meteora, Sumela, Taktsang, Taung Kalat}`, astfel încât pe oricare două poziții alăturate să nu se afle locuri din submulțimea `{Hanging, Sumela, Taung Kalat}`. Două liste diferă prin cel puțin un loc sau prin ordinea acestora. Primele șase soluții generate sunt, în această ordine: `(Hanging, Meteora, Sumela)`, `(Hanging, Meteora, Taktsang)`, `(Hanging, Meteora, Taung Kalat)`, `(Hanging, Taktsang, Meteora)`, `(Hanging, Taktsang, Sumela)`, `(Hanging, Taktsang, Taung Kalat)`. Indicați numărul de soluții generate care au pe prima poziție `Meteora`.

a. `4` **b.** `6` c. `8` d. `10`

4. Un arbore cu rădăcină are `8` noduri, numerotate de la `1` la `8`, și este reprezentat prin vectorul de „tați” `(5,7,5,6,8,5,8,0)`. Indicați frunzele arborelui.

a. `1,2,3,4` b. `1,2,3` c. `1,2,6` d. `4`

5. Un graf orientat cu `5` vârfuri este reprezentat prin matricea de adiacență alăturată. Indicați numărul de vârfuri ale unui subgraf al acestuia care are un număr maxim de vârfuri izolate.

```
0 0 1 0 0
1 0 1 1 0
0 0 0 0 0
0 0 1 0 1
0 0 1 0 0
```

a. `1` **b.** `2` c. `3` d. `4`

SUBIECTUL al II-lea (40 de puncte)

1. **Algoritmul alăturat este reprezentat în pseudocod.**
S-a notat cu `a%b` restul împărțirii numărului natural `a` la numărul natural nenul `b`.

```
citește x,y
   (numere naturale x≤y)
i←x; j←y; s←0
┌cât timp i≤j execută
│┌dacă i%2=0 atunci
│| s←s+j
│└■
│┌dacă j%2=0 atunci
│| s←s+i
│└■
│i←i+1; j←j-1
└■
scrie s
```

a. Scrieți ce se afișează în urma executării algoritmului dacă se citesc, în această ordine, numerele `3` și `12`. **35** **(6p.)**

b. Scrieți două seturi de date din intervalul `[1,10]` care pot fi citite astfel încât, pentru fiecare dintre acestea, în urma executării algoritmului, să se afișeze numărul `20`. **(6p.)**

Doua dintre perechile: (1,9) (2,8) (10,10)

c. Scrieți programul C/C++ corespunzător algoritmului dat. **(10p.)**

d. Scrieți în pseudocod un algoritm, echivalent cu cel dat, înlocuind adecvat structura `cât timp...execută` cu o structură repetitivă de alt tip. **(6p.)**

c)

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

int main()
{
unsigned int x,y,i,j,s;
cin>>x>>y;
i=x;j=y;s=0;
while(i<=j)
{
        if(i%2==0)s=s+j;
        if(j%2==0)s=s+i;
        i++;
        j=j-1;
}
cout<<s;
return 0;
}
```

d)

```
citește x,y
   (numere naturale x≤y)
i←x; j←y; s←0
┌~~cât timp i≤j execută~~ repeta
│┌dacă i%2=0 atunci
│| s←s+j
│└■
│┌dacă j%2=0 atunci
│| s←s+i
│└■
│i←i+1; j←j-1
└■ pana cand i>j
scrie s
```

2. Variabila **ob** memorează simultan următoarele date despre un obiectiv turistic din evidența unei agenții: un cod și o perioadă din an în care se recomandă să fie vizitat, formată din zile consecutive, între două date (ziua și luna de început, respectiv ziua și luna finală). Expresiile C/C++ de mai jos au ca valori numere naturale și reprezintă codul obiectivului, ziua și luna de început, respectiv ziua și luna finală a perioadei recomandate pentru vizitarea acestui obiectiv. Scrieți definiția unei structuri cu eticheta **obiectiv**, care permite memorarea datelor despre un obiectiv turistic, și declarați corespunzător variabila **ob**.

`ob.cod  ob.dataInceput.zi  ob.dataInceput.luna  ob.dataFinal.zi  ob.dataFinal.luna`

(6p.)

```
struct data{
        unsigned int zi, luna;
        }
```

```
struct obiectiv{
        unsigned int cod;
        struct data dataInceput, dataFinal;
        }ob;
```

3. Variabilele **i** și **j** sunt de tip întreg, iar variabila **a** memorează un tablou bidimensional cu **4** linii și **5** coloane, numerotate începând de la **0**, cu elemente numere întregi, inițial toate nule. Fără a utiliza alte variabile decât cele menționate, scrieți o secvență de instrucțiuni C/C++ astfel încât, în urma executării acesteia, variabila **a** să memoreze tabloul alăturat. **(6p.)**

20	16	12	8	4
19	15	11	7	3
18	14	10	6	2
17	13	9	5	1

```
for(i=3;i>=0;i--)
        for(j=4;j>=0;j--)
                if(j==4) a[i][j]=4-i;
                else a[i][j]=a[i][j+1]+4;
```

SUBIECTUL al III-lea (30 de puncte)

1. Subprogramul **maxim** are un singur parametru, **n**, prin care primește un număr natural (**n**$\in[0,10^9]$). Subprogramul returnează cea mai mare cifră impară din scrierea acestuia, sau **-1** dacă nu există astfel de cifre. Scrieți definiția completă a subprogramului.

Exemplu: dacă **n=5672883**, subprogramul returnează **7**. **(10p.)**

```
int maxim(unsigned long n)
{
        int c,cmax=0;
        while(n>0)
        {
                c=n%10;
                n=n/10;
                if(c%2==1 && c>cmax) cmax=c;
        }

        if(cmax==0) return -1;
             else return cmax;

}
```

2. Într-un text cu cel mult 10^2 caractere cuvintele sunt formate din litere mici ale alfabetului englez și sunt separate prin câte un spațiu. Scrieți un program C/C++ care citește de la tastatură un text de tipul menționat, pe care îl modifică în memorie, duplicând fiecare cuvânt format numai din vocale. Cuvântul duplicat este separat prin câte un spațiu de cuvintele vecine. Textul transformat este afișat pe ecran, iar dacă nu există astfel de cuvinte, se afișează pe ecran mesajul **nu exista**.

Exemplu: dacă textul citit este **oaia aia alba e a ei**

se obține textul **oaia oaia aia aia alba e e a a ei ei** **(10p.)**

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

int main()
{
  char str[101],sir[210],spatiu[2];
```

```
   char *p;
   int i,n,gasit,j;
   char v[50][20];
   cin.get(str,100);
   n=0;
   p = strtok(str, " ");
   while( p != NULL )
   {
     n++; strcpy(v[n],p);
     p = strtok(NULL, " ");
   }
//for(i=1;i<=n;i++) cout<<v[i]<<' ';
spatiu[0]=' ';spatiu[1]='\0';
strcpy(sir,"");
for(i=1;i<=n;i++)
{
          j=0;
          while(j<strlen(v[i])&&strchr("aeiou",v[i][j])!=0)j++;
          if(j==strlen(v[i])){
                                                    strcat(sir,spatiu);
                                                    strcat(sir,v[i]);
                                                    strcat(sir,spatiu);
                                                    strcat(sir,v[i]);
                                          }
                                                    else
                                                    {
                                                    strcat(sir,spatiu);
                                                    strcat(sir,v[i]);
                                                    }
          }

cout<<sir;
return 0;
}
```

3. Fișierul text **bac.txt** conține numere naturale din intervalul **[1,10⁴]**: pe prima linie un număr **n**, pe a doua linie un șir de **n** numere, iar pe fiecare dintre următoarele linii, până la finalul fișierului, câte o pereche de numere, reprezentând extremitățile unui interval închis. Numerele aflate pe aceeași linie a fișierului sunt în ordine crescătoare și sunt separate prin câte un spațiu.

Se cere să se afișeze pe ecran numărul de intervale care nu conțin niciun termen al șirului aflat pe a doua linie a fișierului. Proiectați un algoritm eficient din punctul de vedere al timpului de executare.
Exemplu: dacă fișierul conține numerele alăturate, se afișează pe ecran **3**.
a. Descrieți în limbaj natural algoritmul proiectat, justificând eficiența acestuia. **(2p.)**
b. Scrieți programul C/C++ corespunzător algoritmului proiectat. **(8p.)**

```
5
4 8 9 16 25
1 3
2 5
9 15
5 7
20 100
10 12
```

Problema cere sa se proiecteze un algoritm eficient din punctul de vedere al timpului de executare.
Am folosit urmatoarele variabile:
x,y – pentru fiecare interval [x,y] ce trebuia citit
fr – tablou unidimensional umplut initial cu 0; pentru fiecare valoare ce poate fii continuta in fisier (i=1...10000) v[i]=0 daca i nu se gaseste pe a doua linie in fisier sau v[i]=1 daca i se gaseste pe a doua linie in fisier.
Dupa ce am citit valorile sirului de pe a doua linie din fisier si am initializat corespunzator vectorul fr, vom citi, pe rand intervalele [x,y]; vom pleca cu cautarea de la x si vom incerca sa ajungem la y parcurgand doar elemente fr[i]=0 unde i=x..y; daca gasim un element fr[i]!=0 atunci putem trage concluzia ca [x,y] nu respecta conditia ceruta. In cazul in care i a ajuns la capatul intervalului (y) inseamna ca niciun element din [x,y] nu e gaseste printre elementele sirului aflat pe cel de-al doilea rand in fisier si vom mari numarul de intervale care respecta conditia data cu 1.

Datele din fisier s-au citit doar o data. Verificarea conditiei ceruta pentru interval se realizeaza imediat dupa citirea marginilor de interval; numerele din fisier sunt in ordine crescatoare lucru ce usureaza cautarea.
Singurele instructiuni repetitive imbricate sunt cele care citesc perechile de numere si verifica daca din intervalul respectiv fac parte elemente din sir insa, numarul de repetitii executat de instructiunea repetitiva care verifica existenta numerelor de pe cel de-al doilea sir este relativ, algoritmul avand nevoie, astfel, de putin timp pentru executare.

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

ifstream f("bac.txt");
int main()
{
int x,y,fr[10000],n,nr=0,k;
f>>n;
int i;
for(i=1;i<=10000;i++) fr[i]=0;
for(i=1;i<=n;i++) {f>>k;fr[k]=1;}
while(!f.eof())
{
        f>>x>>y;
        i=x;
        while(i<=y && fr[i]==0)i++;
        if(i>y) {nr++;cout<<x<<' '<<y<<endl;}
}
cout<<nr;
f.close();
return 0;
}
```