

**Laborator 2**  
**1. Pointeri**  
**2. Stive si cozi implementate dinamic**

**1. Pointeri**

**1. Tablouri de pointeri și pointeri la tablouri**

Pointerii fiind variabile, pot fi folosiți pentru a forma alte tipuri de date compuse. Spre exemplu se pot forma tablouri de pointeri. Sintaxa generală de utilizare este :

*Tip* \*tablou[dim] ;

*Exemple:*

1. Declarația `char *s[25]`; reprezintă un tablou de 25 pointeri la caracter.
2. Sortarea unor șiruri de caractere:

```
#include<iostream>
#include<string.h>
using namespace std;
void sort_lines(char *tp[], int n)
{
int i, sort=0;
char *temp;
while (!sort)
{
sort=1;
for(i=0;i<n-1;i++)
    if(strcmp(tp[i],tp[i+1])>0)
    {
strcpy(temp,tp[i]); strcpy(tp[i],tp[i+1]);strcpy(tp[i+1],temp);
sort=0;
}
}
}
main()
{
int i;
char *sir[]={{"manual"},{"carte"},{"mare"},{"12345"},{"4542"}};
sort_lines(sir,5);
for(i=0;i<5;i++)
    cout<<sir[i]<<' ';
cout<<endl;
}
```

## PROIECTAREA ALGORITMILOR

### Laborator 2

Pentru a arăta că folosim pointeri la tablouri (și nu tablouri la pointeri) avem pentru declarație sintaxa următoare:

*Tip (\*p) [dim] ;*

*Diferența dintre pointerii la tablouri de un tip și tablourile de pointeri la același tip este modul de stabilire a unității de deplasare.*

În acest sens, sunt concludente următoarele exemple:

3. În definiția

a) `int (*x) [50];`

x este un pointer la un tablou de 50 întregi.

b) `int* x[50];`

x este un tablou de 50 pointeri la întregi.

Se observă diferența de semnificație care are loc o dată cu folosirea parantezelor.

4. Fie declarația

`short (*n) [10];`

În acest exemplu *n* reprezintă adresa unui tablou de 10 întregi. Chiar dacă acest lucru este inițial sinonim cu adresa de început a tabloului, diferența este esențială: *aritmetica* acestora cu numerele întregi. Astfel, incrementarea lui *n* va avea ca efect deplasarea cu 10 întregi de tip short (deci saltul este de 20 de octeți), pe când în cazul unui pointer la întreg deplasarea ar fi fost de 2 octeți.

Astfel, în programul:

`/* deplasarea pentru un tablou la pointer */`

```
#include<iostream>
using namespace std;
int main()
{
short *a, (*b) [10], c[10];
a=c;
b=c;
cout<<“\nInainte de incrementare:\n Adresa pointer la intreg:<<a<<”\t Adresa pointer la tablou
de intregi: ”<<b;
a++;
b++;
cout<<“\nDupa incrementare:\n Adresa pointer la intreg:”<<a<<”\t Adresa pointer la tablou de
intregi: ”<<b;
}
```

*S-au obținut rezultatele:*

Înainte de incrementare:

Adresa pointer la întreg: FFE2

Adresa pointer la tablou de întregi: FFE2

După incrementare:

Adresa pointer la întreg: FFE4

Adresa pointer la tablou de întregi: FFF6

5. Declarațiile următoare sunt corecte:

`void matr(int *a[], int n); /* a este un tablou de pointeri */`

`void matr(int **a, int n);`

În ambele antete, primul parametru poate fi folosit pentru matrici de dimensiune variabilă.

## 2. Tratarea tablourilor bidimensionale folosind pointeri

Numele unui tablou bidimensional, la fel ca numele unui tablou unidimensional, are ca valoare *adresa primului element al tabloului*.

Fie, de exemplu, declarația:

*tip* *tab*[...][...];

Atunci ***tab*** are ca valoare adresa lui *tab*[0][0].

Dacă în cazul tablourilor unidimensionale, *tab+n* este adresa elementului *tab*[*n*], adică &*tab*[*n*], în cazul tablourilor bidimensionale, expresia *tab+n* este *adresa elementului tab*[*n*][0].

Această interpretare rezultă din faptul că un tablou bidimensional trebuie considerat ca fiind un tablou de tablouri unidimensionale. În general, un tablou *k*-dimensional este un tablou de tablouri *k-1* dimensionale.

Fie, de exemplu, declarația: *tip tab*[*m*][*n*];

Tabloul *tab* poate fi privit ca un tablou de tablouri unidimensionale. Astfel, *tab*[0], *tab*[1], ..., *tab*[*m-1*] sunt cele *m* elemente ale lui *tab* și fiecare este un pointer spre câte un tablou unidimensional de *n* elemente. De aici rezultă că *tab*[0] are ca valoare adresa lui *tab*[0][0], *tab*[1] are ca valoare adresa lui *tab*[1][0], iar în general ***tab*[*i*] are ca valoare adresa lui *tab*[*i*][0]**. Cum ***\*(tab+i)* are ca valoare chiar *tab*[*i*]**, rezultă că ***tab*[*i*][0] are aceeași valoare cu *\*(tab+i)***.

De asemenea, *tab*[*i*]+*j* are ca valoare adresa lui *tab*[*i*][*j*]. Deci *tab*[*i*][*j*] are aceeași valoare ca și expresia *\*(tab[i]+j)*, iar aceasta din urmă are aceeași valoare cu expresia *\*(\*(tab+i)+j)*.

Înseamnă că atribuirile *\*(\*(tab+i)+j)=x* și *tab*[*i*][*j*]=*x* sunt echivalente.

Dacă *tab* este un tablou unidimensional declarat prin: *tip tab*[...]; atunci *tab* are tipul *tip\** adică este pointer spre *tip*.

Dacă *tab* este un tablou bidimensional declarat prin *tip tab*[*m*][*n*]; atunci *tab* are tipul *tip (\*)*[*n*]; adică pointer spre un tablou de *n* elemente de tipul *tip*.

Menționez că în declarația de mai sus *m* și *n* sunt expresii constante.

Să presupunem că *tab* este parametru la apelul funcției *f*: ...*f*(*tab*). În acest caz, parametrul formal al funcției *f* poate fi declarat în următoarele moduri: ...*f*(*tip t*)[*n*] sau ...*f*(*tip(\*)*)[*n*].

Fie declarația: *tip \*tab1*[*n*];

În acest caz, *tab1* este un tablou unidimensional de pointeri spre tipul *tip*, adică fiecare element din cele *n* ale tabloului *tab1* este un pointer spre *tip*. Tablourile *tab* și *tab1* nu trebuie confundate. Tabloului *tab* i se alocă o memorie de *m\*n\*sizeof(tip)* octeți. Tabloului *tab1* i se alocă *n\*sizeof(tip\*)* octeți.

Dacă o funcție *q* are la apel ca parametru pe tabloul *tab1*: ...*q*(*tab1*) atunci parametrul formal al funcției *q* poate fi declarat în următoarele moduri: ...*q*(*tip \*p*[]) sau ...*q*(*tip \*\*p*).

### Exemple:

1. Fie declarațiile:

```
double tab[2][3]={{0,1,2},{3,4,5}}
```

```
double *p;
```

Atunci avem:

INSTRUCȚIUNI	ELEMENTUL AFIȘAT	VALOAREA ELEMENTULUI
p=tab[0]; cout<<*p;	tab[0][0]	Zero
p=tab[1]; cout<<*p;	tab[1][0]	3
cout<<*(*(tab));	tab[0][0]	Zero
cout<<*(*(tab+1)+2));	tab[1][2]	5

1. Considerăm funcția  $f$  definită astfel:

```
void f (double (*p)[3])
{
  int i,j;
  for (i=0;i<2;i++)
  for(j=0;j<3;j++)
    cout<<p[i][j];
}
```

La apelul  $f(tab)$ ; unde  $tab$  este tabloul definit la exemplul anterior, se afișează valorile elementelor lui  $tab$  separate de câte un spațiu:

0 1 2 3 4 5.

Expresia  $p[i][j]$  poate fi înlocuită cu  $*(*(p+i)+j)$ .

2. Fie declarațiile:

```
double *t[2];
double t0[3]={10,11,12};
double t1[3]={13,14,15};
și atribuirile t[0]=t0; t[1]=t1;
Definim funcția  $f1$  astfel:
void f1(double *p[])
{
  ...
}
```

unde corpul funcției  $f1$  coincide cu al funcției  $f$ .

La apelul funcției  $f1(t)$ ; se listează valorile elementelor tablourilor  $t0$  și  $t1$ , separate prin câte un spațiu, adică: 10 11 12 13 14 15

Același rezultat se obține dacă antetul lui  $f1$  se schimbă cu:  $void f1(double **p)$ .

În ambele situații, expresia  $p[i][j]$  poate fi schimbată cu  $*(*(p+i)+j)$ .

## Probleme rezolvate cu șiruri de caractere si pointeri

### Problema 1

Scrieți un program C++ care citește de la tastatura doua șiruri de caractere de maximum 100 de litere mici și verifica utilizând apeluri ale funcției **aparitii** daca cele doua siruri sunt anagrame (contin aceleasi litere, dar in ordine diferita). Se cere afișarea mesajului **anagrame** în caz afirmativ și a mesajului **nu sunt anagrame** în caz contrar.

Exemplu: Pentru șirurile **adrian** și **nairda** se afișează **șirurile sunt anagrame**.

```
#include <iostream>
#include<string.h>
using namespace std;
char s[101], t[101], c, *p;
int k, x, ok = 1;
int main()
{
    cin.get(s, 101);
    cin.get();
    cin.get(t, 101);
    if(strlen(s) != strlen(t)) ok = 0; // au lungimi diferite
    else
    {
        for(c = 'a'; c <= 'z' && ok == 1; c++) // fiecare litera din alfabet
        {
            x = k = 0;
            p = strchr(s,c);
            while(p != 0) // se numara de cate ori apare litera c in șirul s
            {
                x++; p = strchr(p + 1, c);
            }
            p = strchr(t, c);
            while(p != 0) // se numara de cate ori apare litera c in șirul t
            {
                k++; p = strchr(p + 1, c);
            }
            if(x != k) ok = 0; // dacă numărul de apariții diferă
        }
    }
    if(ok == 1) cout<<"sirurile sunt anagrame";
    else cout<<"sirurile NU sunt anagrame";
}
```

```

return 0;
}
    
```

Executia programului folosind compilatorul online <https://www.jdoodle.com/online-compiler-c++/>:

The screenshot shows a C++ program in an online compiler. The code reads two strings, s and t, and checks if they are anagrams by counting the frequency of each letter. The execution result shows the output: "sirurile sunt anagrame".

```

1 #include <iostream>
2 #include <string.h>
3 using namespace std;
4 char s[101], t[101], c, "p"; int k, x, ok=1;
5 int main()
6 {
7     cin.get(s, 101);
8     cin.get();
9     cin.get(t, 101);
10    if(strlen(s) != strlen(t)) ok = 0; // au lungimi diferite
11    else
12    {
13        for(c = 'a'; c <= 'z' && ok == 1; c++) // fiecare litera din alfabet
14        {
15            x = k = 0;
16            p = strchr(s, c);
17            while(p != 0) // se numara de cate ori apare litera c in sirul s
18            {
19                x++; p = strchr(p + 1, c);
20            }
21            p = strchr(t, c);
22            while(p != 0) // se numara de cate ori apare litera c in sirul t
23            {
24                k++; p = strchr(p + 1, c);
25            }
26            if(x != k) ok = 0; // daca numarul de aparitii difera
27        }
28    }
29    if(ok == 1) cout << "sirurile sunt anagrame";
30    else cout << "sirurile NU sunt anagrame";
31    return 0;
32 }
    
```

Execution details: GCC 11.1.0, Interactive mode, Stdin Inputs: adrian, nairda. Result: sirurile sunt anagrame. CPU Time: 0.00 sec(s), Memory: 3420 kilobyte(s).

**Problema 2**

Se citește de la tastatura un text format din cuvinte separate între ele prin cate un singur spațiu. Fiecare cuvânt are cel mult 40 de caractere, doar litere mici ale alfabetului englez. Textul are cel mult 200 de caractere. Sa se scrie un program C++ care sa afișeze pe ecran, pe linii separate, doar cuvintele din textul citit care conțin cel mult trei vocale. Se considera vocale: **a, e, i, o, u**.

Exemplu:

Pentru textul:

**pentru** examenul de programarea calculatoarelor **se folosesc** fisiere

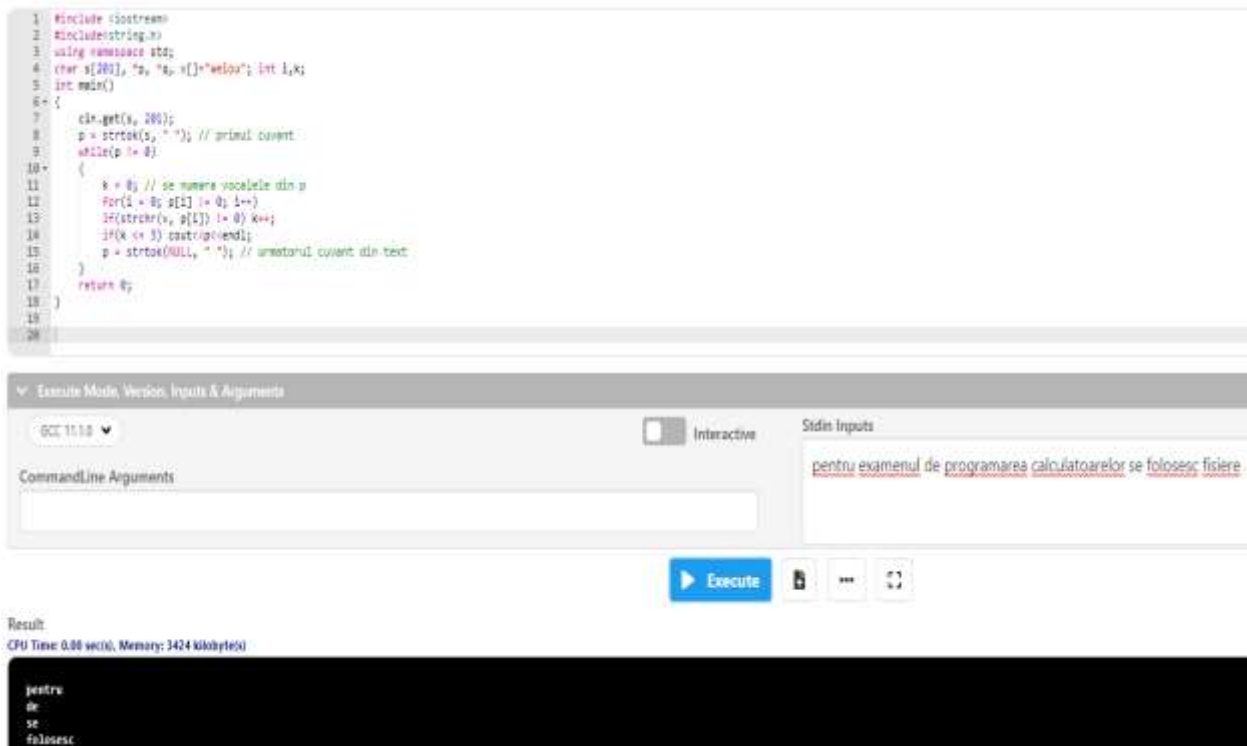
se afiseaza:

**pentru**

de  
se  
folosesc

```
#include <iostream>
#include<string.h>
using namespace std;
char s[201], *p, *q, v[]="aeiou"; int i,k;
int main()
{
    cin.get(s, 201);
    p = strtok(s, " "); // primul cuvnt
    while(p != 0)
    {
        k = 0; // se numara vocalele din p
        for(i = 0; p[i] != 0; i++)
            if(strchr(v, p[i]) != 0) k++;
        if(k <= 3) cout<<p<<endl;
        p = strtok(NULL, " "); // urmatorul cuvnt din text
    }
    return 0;
}
```

Executia programului folosind compilatorul online <https://www.jdoodle.com/online-compiler-c++/>:



**Problema 3**

Sa se scrie un program C++ care citeste de la tastatura un cuvânt de cel mult 20 de litere mici ale alfabetului englez și care să afișeze pe ecran, pe linii diferite, cuvintele obținute prin ștergerea succesivă a vocalelor în ordinea alfabetică a lor (**a, e, i, o, u**). La fiecare pas se vor șterge toate aparițiile din cuvânt ale unei vocale.

Exemplu:

dacă se citește cuvântul **programare** se va afișa:

**progrmr** (s-au șters cele două apariții ale vocalei **a**)

**prgrmr** (s-a șters unica apariție ale vocalei **e**)

**prgrmr** (s-a șters unica apariție ale vocalei **o**)

```
#include <iostream>
#include<string.h>
using namespace std;
char s[21], v[]="aeiou",*p;
int i = 0;
int main()
{
    cin.get(s,21);
    for(i = 0; v[i] != 0; i++) // se parcurge multimea vocalelor
    {
        p=strchr(s, v[i]); // vocale v[i] apare in text
        if(p != 0)
        {
            while(p != 0) // se șterg toate aparițiile
            {
                strcpy(p, p + 1);
                p = strchr(s, v[i]);
            }
            cout<<s<<endl; // se afișează șirul obținut
        }
    }
    return 0;
}
```

Execuția programului folosind compilatorul online <https://www.jdoodle.com/online-compiler-c++/>:






```
1 #include <iostream>
2 #include <string.h>
3 using namespace std;
4 char s[21], v[]="aeiou",*p;
5 int i = 0;
6 int main()
7 {
8     cin.get(s,21);
9     for(i = 0; v[i] != 0; i++) //se parcurge multimea vocalelor
10    {
11        p=strchr(s, v[i]); //vocale v[i] apare in text
12        if(p != 0)
13        {
14            while(p != 0) //se sterg toate aparitiile
15            {
16                strcpy(p, p + 1);
17                p = strchr(s, v[i]);
18            }
19            cout<<endl; //se afiseaza sirul obtinut
20        }
21    }
22    return 0;
23 }
24
25
26
```

Execute Mode, Version, Inputs & Arguments

GCC 11.1.0  Interactive Stdin Inputs

CommandLine Arguments

programare

**Execute**   

Result

CPU Time: 0.00 sec(s), Memory: 3396 kilobyte(s)

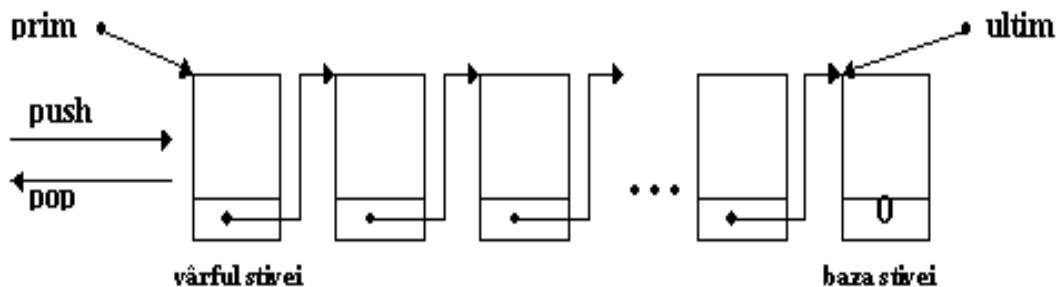
```
programare
programare
programare
```

**Probleme propuse spre rezolvare**

1. Sa se ruleze toate exemplele prezentate mai sus.

## 2. Stive si cozi implementate dinamic

O **stivă** se definește ca o listă liniară simplu înlănțuită în care toate intrările și ieșirile se fac pe la un singur capăt al ei. Stiva este o structură de tip LIFO (Last In First Out), adică ultimul nod introdus este primul scos. Rezultă că, înregistrarea de pe nivelul k reține înregistrarea de pe nivelul k-1. În cazul stivei se reține doar elementul din vârful stivei.



Fiind o structură particulară a unei liste simplu înlănțuite, operațiile principale asupra unei stive sunt:

- **push** = adăugare - pune un element pe stivă; funcția se realizează prin inserarea unui nod înaintea primului;
- **pop** = eliminare - scoate elementul din vârful stivei; funcția se realizează prin ștergerea primului nod;
- **clear** - ștergerea stivei;

1. Sa se creeze o lista cu numere intregi folosind crearea prin adaugarea elementelor la inceputul listei (stiva). Se cere:

- a) Sa se afiseze continutul listei
- b) Sa se determine suma si maximul elementelor listei.

Exemplu:

Pentru  $n=5$  si elementele  $\{12, 10, -23, 100, -3\}$ , se obtine suma elementelor = 96 si maximul elementelor = 100.

```
#include <iostream>
```

```
using namespace std;
```

```
struct lista
```

```
{
```

```
    int info;
```

```
    lista *leg;
```

```
};
```

```
lista *p, *prim, *ultim;
int n;

void creare(lista *&prim, lista *&ultim)
{
    int i,inf;
    cin>>n;
    cin>>inf;
    prim=new lista;
    prim->info=inf;
    prim->leg=NULL;
    ultim=prim;
    for(i=2;i<=n;i++)
    {
        cin>>inf;
        p=new lista;
        p->info=inf;
        p->leg=prim;
        prim=p;
    }
}

void afisare(lista *prim)
{
    p=prim;
    while(p!=NULL)
    {
        cout<<p->info<<" ";
        p=p->leg;
    }
}

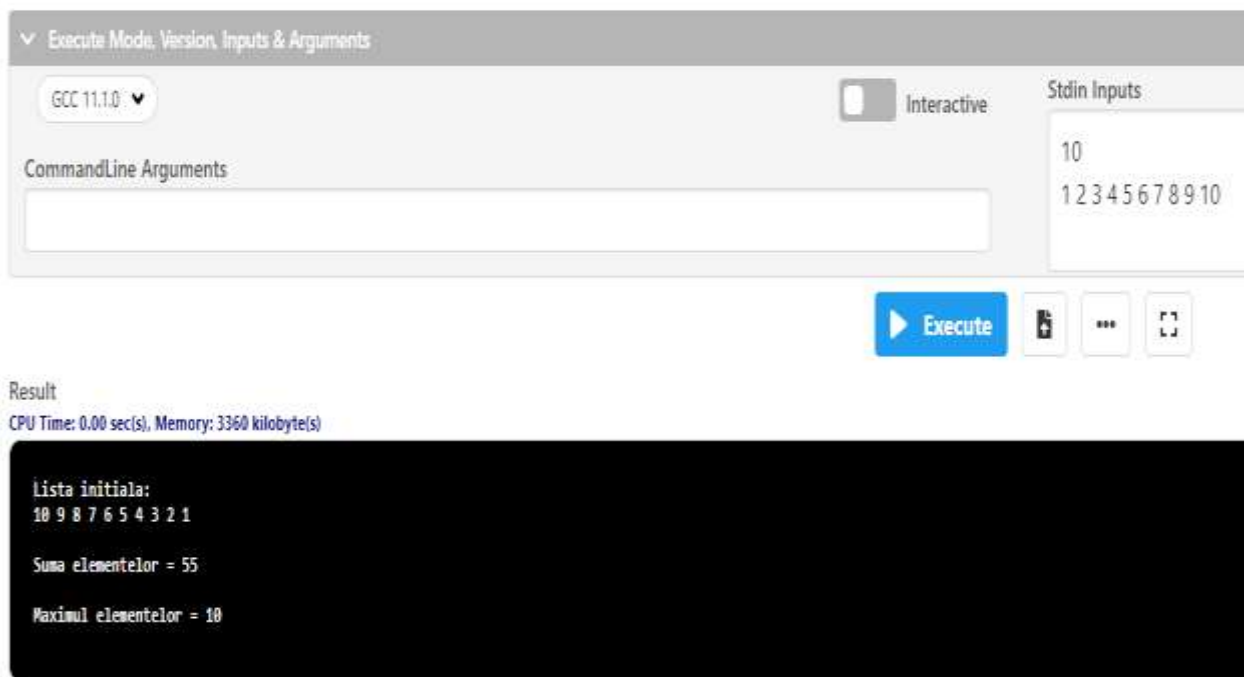
int suma(lista *prim)
{
    int suma=0;
    p=prim;
    while(p!=NULL)
    {
        suma+=p->info;
        p=p->leg;
    }
    return suma;
}

int maxim(lista *prim)
{
    int max=-10000;
    p=prim;
    while(p!=NULL)
```

```
{
    if( p->info > max ) max=p->info;
    p=p->leg;
}
return max;
}

int main(void)
{
    creare(prim, ultim);
    cout<<"Lista initiala:\n";
    afisare(prim);
    cout<<endl<<"\nSuma elementelor = "<<suma(prim);
    cout<<endl<<"\nMaximul elementelor = "<<maxim(prim);
    return 0;
}
```

Executia programului folosind compilatorul online <https://www.jdoodle.com/online-compiler-c++/>:



2. Să se creeze o lista cu numere întregi folosind crearea prin adăugare (coada) la sfarsitul listei.

Se cere:

- a) Să se afișeze conținutul listei
- b) Să se determine numărul de numere prime

```
#include <iostream>
using namespace std;
struct lista
{
    int info;
    lista *leg;
};

lista *p, *prim, *ultim;
int n;

void creare(lista *&prim, lista *&ultim)
{
    int i,inf;
    cin>>n;
    cin>>inf;
    prim=new lista;
    prim->info=inf;
    prim->leg=NULL;
    ultim=prim;
    for(i=2;i<=n;i++)
    {
        cin>>inf;
        p=new lista;
        p->info=inf;
        p->leg=NULL;
        ultim->leg=p;
        ultim=p;
    }
}

void afisare(lista *prim)
{
    p=prim;
    while(p!=NULL)
    {
        cout<<p->info<<" ";
        p=p->leg;
    }
}
```

```

int verif_prim(int x)
{
    int q=1;
    if(x == 1) return 0;
    for(int i=2;i<=x/2;i++)
        if(x%i==0)    q=0;
    return q;
}

int nr_prime(lista *prim)
{
    int nr=0;
    p=prim;
    while(p!=NULL)
    {
        if(verif_prim(p->info)==1) nr++;
        p=p->leg;
    }
    return nr;
}

int main(void)
{
    creare(prim,ultim);
    afisare(prim);
    cout<<endl<<"numarul de numere prime = "<<nr_prime(prim)<<endl;
    return 0;
}

```

Executia programului folosind compilatorul online <https://www.jdoodle.com/online-compiler-c++/>:

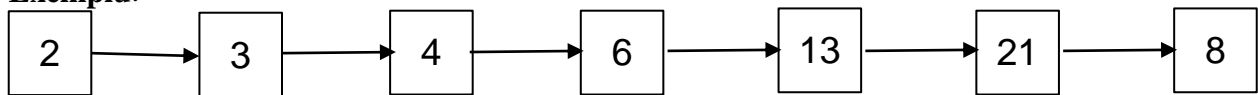
The screenshot shows an online C++ compiler interface. At the top, there is a dropdown menu for the compiler version, currently set to 'GCC 11.1.0'. To the right of this is a checkbox labeled 'Interactive' which is currently unchecked. Below the compiler version is a text input field for 'CommandLine Arguments'. To the right of this is a 'Stdin Inputs' section with a text area containing the number '10' and the sequence '1 2 3 4 5 6 7 8 9 10'. At the bottom right of the interface is a blue 'Execute' button with a play icon, and next to it are icons for file operations and a refresh icon. Below the execution area, the 'Result' section shows the output of the program: '1 2 3 4 5 6 7 8 9 10' followed by 'numarul de numere prime = 4'. Above the result, it also displays performance metrics: 'CPU Time: 0.00 sec(s), Memory: 3404 kilobyte(s)'.

**Probleme propuse spre rezolvare**

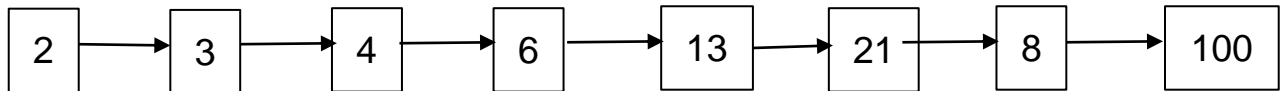
1. Să se ruleze programele prezentate mai sus, urmărind apelurile și valorile parametrilor de apel.

2. Sa se implementeze un program care sa exemplifice operațiile care se efectuează într-o coada, construind un program C++ cu meniuri specifice operațiile cu coada.

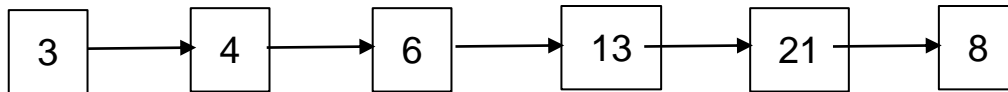
**Exemplu:**



Adăugarea valorii 100, la sfarsitul cozii:



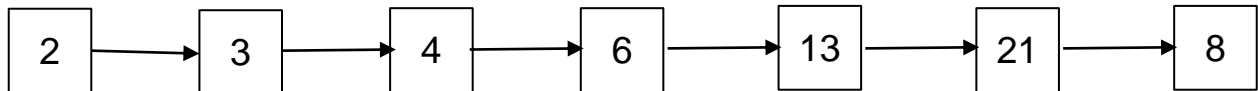
Eliminarea valorii 2, de la începutul cozii:



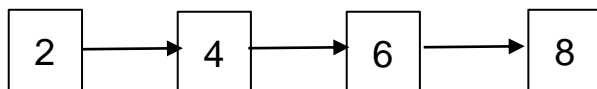
3. Se da o lista de tip stiva cu numere naturale. Să se creeze doua liste, una cu numerele pare din lista inițială, iar cea de a doua cu numerele impare. Să afișeze la final cele doua liste obținute.

**Exemplu:**

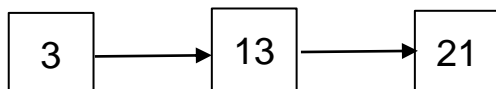
Lista, inițial, conține următoarele valori:



Lista cu elemente pare:



Lista cu elemente impare:

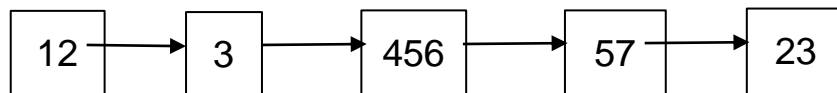


4. Să se creeze o listă simplu înlănțuită (de tip coada) cu numerele naturale preluate de la tastatura. Numerele sunt dispuse pe mai multe linii ale fișierului de intrare, cele de pe aceeași linie fiind despărțite de câte un spațiu. Se cere să se adauge după fiecare număr din listă, răsturnatul său. Să se afișeze conținutul listei înainte și după modificare, fiecare afișare pe câte un rând.

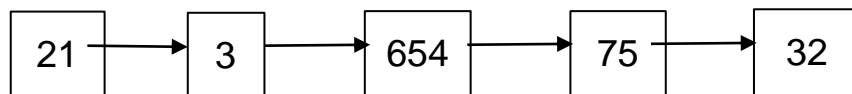
**Exemplu:**

Se introduc următoarele valori: 12 3 456 57 23

Se obține următoarea listă simplu înlănțuită (de tip coada):



Lista modificata:



5. Să se creeze o listă simplu înlănțuită (de tip stiva) cu numerele întregi preluate de la tastatura. Numerele sunt dispuse pe mai multe linii ale zonei de intrare, cele de pe aceeași linie fiind despărțite de câte un spațiu. Se cere să se adauge după fiecare număr din listă cifra de control a numărului respectiv. Să se afișeze conținutul listei înainte și după modificare, fiecare afișare pe câte un rând.

**Exemplu:**

Se introduc următoarele valori:

153

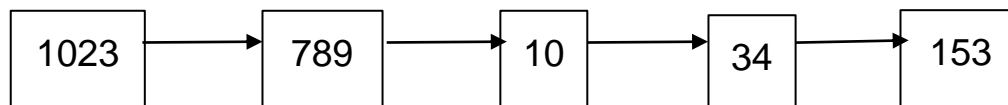
34

10

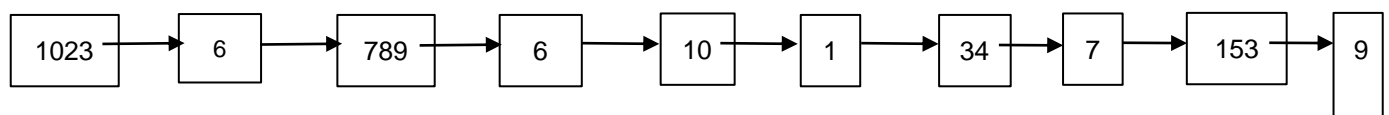
789

1023

Se obține următoarea listă simplu înlănțuită (de tip stiva):



Lista modificata:





6. In doua fisiere text F1.IN si F2.IN se gasesc 2 multimi A si respectiv B cu elemente numere naturale.

a) Sa se creeze cate o lista liniara simplu inlantuita pentru memorarea elementelor din fiecare dintre cele 2 multimi.

b) Sa se creeze o lista liniara simplu inlantuita care se memoreze reuniunea celor 2 multimi.

c) Sa se creeze o lista liniara simplu inlantuita care se memoreze intersecția celor 2 multimi.

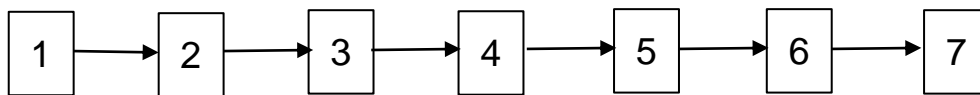
**Exemplu:**

In fisierul **F1.IN** avem elementele din multimea **A**: 1 2 3 4 5 6 7

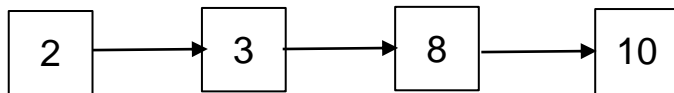
In fisierul **F2.IN** avem elementele din multimea **B**: 2 3 8 10

a)

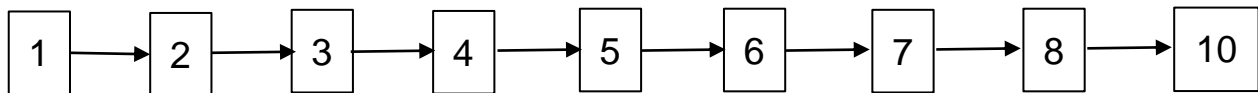
Prima lista cu elementele din multimea A:



A doua lista cu elementele din multimea B:



b) Lista cu elementele reuniunii:



c) Lista cu elementele intersecției:

