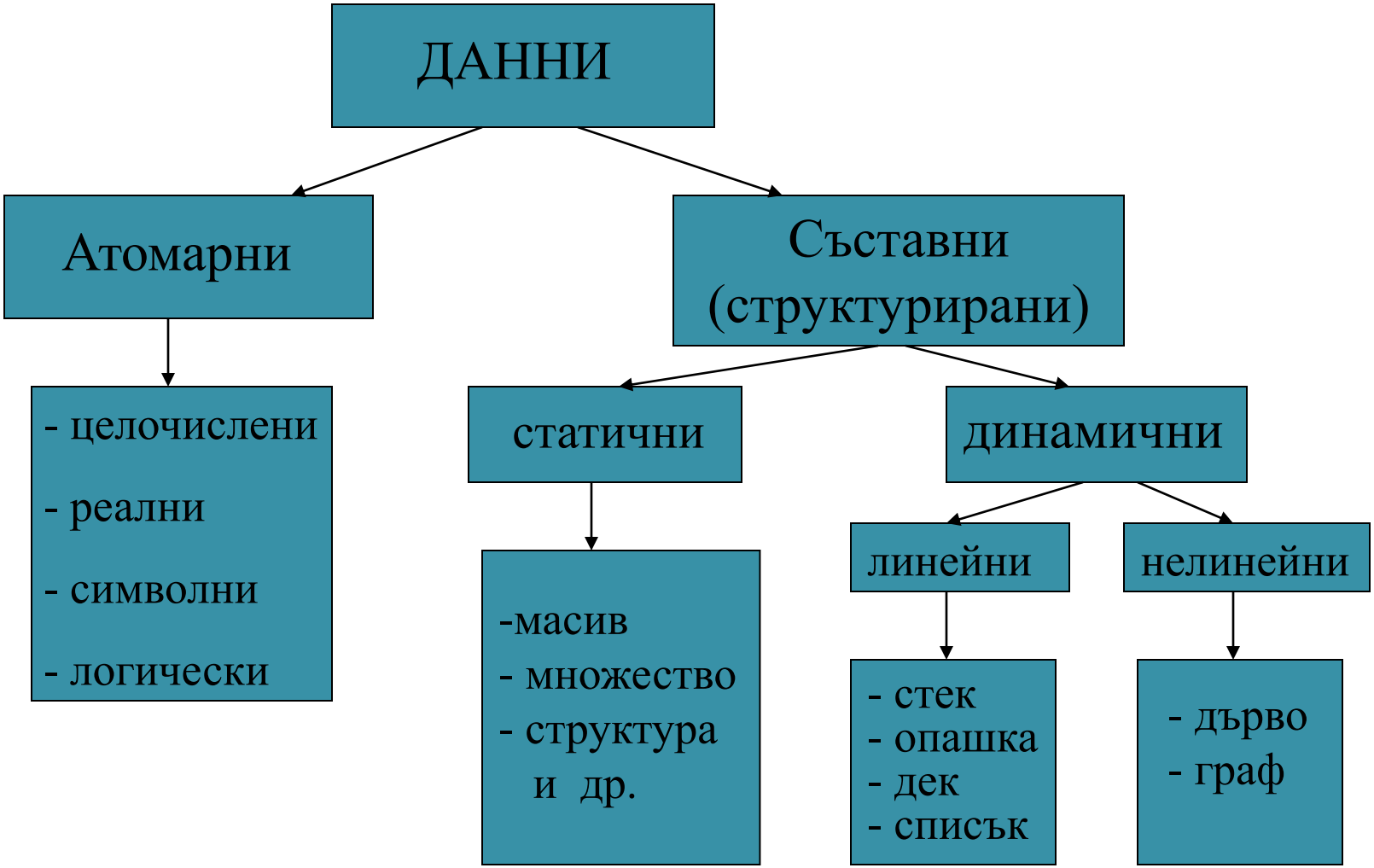
СТРУКТУРИ ОТ ДАННИ

"АЛГОРИТМИ + СТРУКТУРИ ОТ ДАННИ = ПРОГРАМИ"

Н. УИРТ

Класификация на данни



Множество

Множествата са колекции, в които няма повтарящи се елементи.

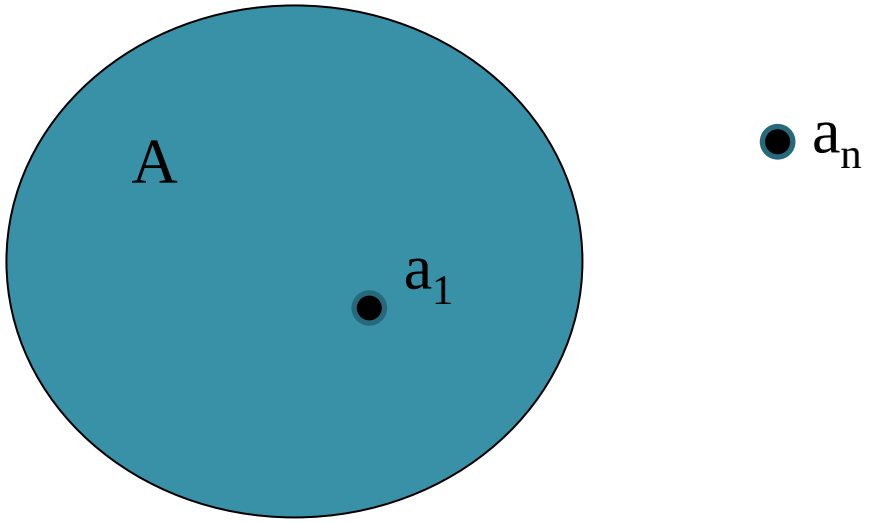
Освен, че не допуска повтарящи се обекти, друго важно нещо, което отличава множеството от масивите е, че неговите елементи си нямат номер. Елементите на множеството не могат да бъдат достъпвани по някакъв друг ключ, както е при речниците. Самите елементи играят ролята на ключ.

Фактът, че елементът a_i, i = 1,2,…, n, принадлежи на множеството A се обозначава с $a_i \in A$, а че не принадлежи - $a_i \notin A$.

Броят на елементите на множество се нарича мощност на множеството (n).

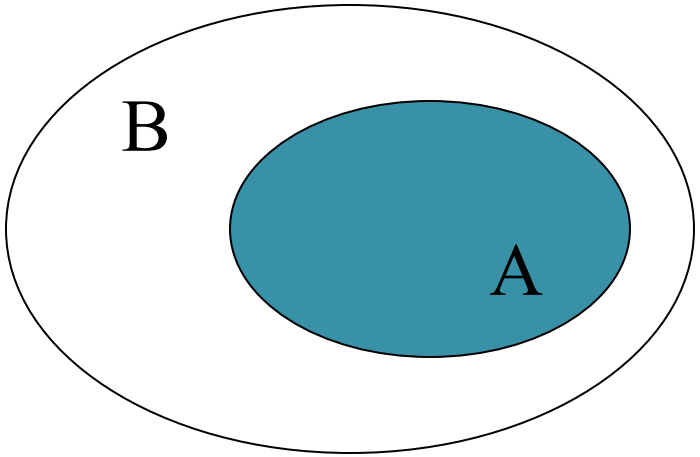
При n=0 множеството се нарича празно.

Диаграма на Вен:



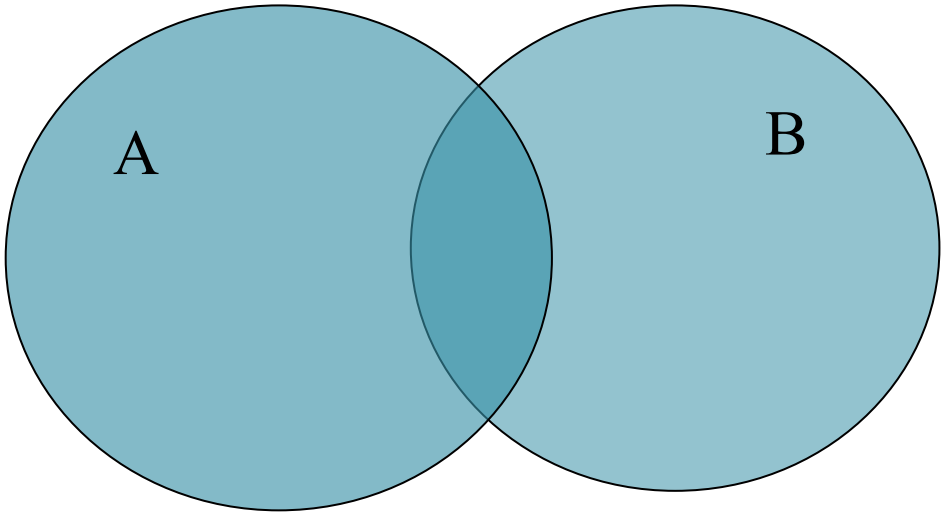
Ако всички елементи на дадено множество **A** са елементи на друго множество **B**, то **A** се нарича ***подмножество*** на **B**:

$A \subseteq B$

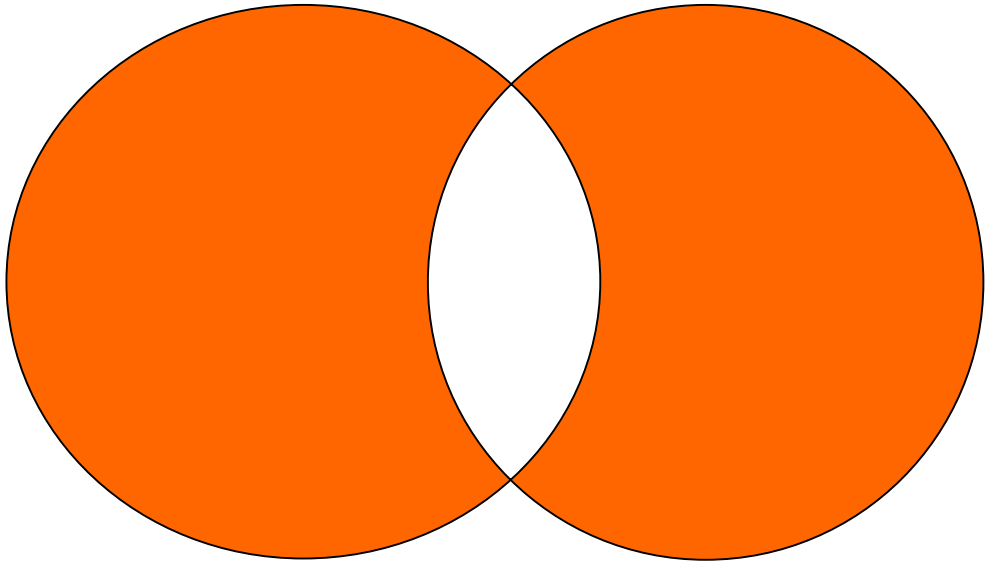


Множеството **C** се нарича ***обединение*** на множествата **A** и **B**, ако се състои от всички елементи **a** такива, че **a** $\in$ **A** и **a** $\in$ **B**.

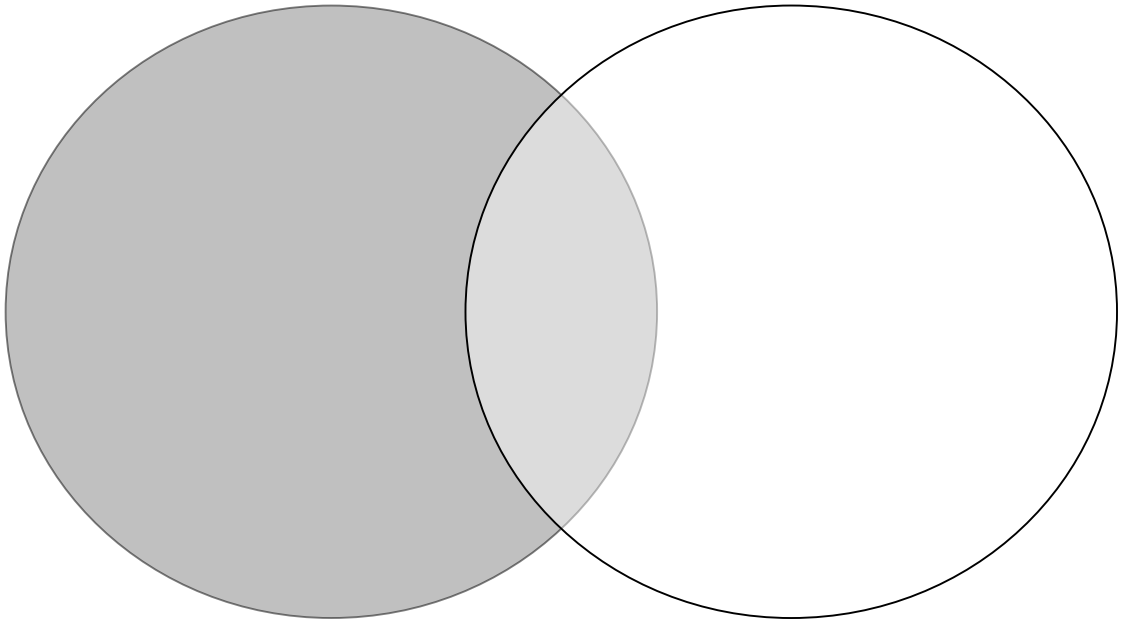
Записва се **C = A** $\cup$ **B**



Сечение **C = A ∩ B** на две множества **A** и **B** се нарича множество **C**, състоящо се от всички елементи, които принадлежат едновременно на **A** и **B**.



Разлика **C=A \ B** на множествата **A** и **B** се нарича множество **C**, състоящо се от всички елементи, които принадлежат на **A** и не принадлежат на **B**.



Прост пример за използване на множество:
Класически алгоритъм ***сито на Ератостен*** (III век преди н. е.) за намиране на всички прости числа в интервал от 0 до ***n***:

```
//primes from 1 to n
{
    F=[ ];
    for (i=2; i<n; i++)
       F=F+[i];
    for (i=2; i<n; i++)
          if (i in F)
                for (j=i; j*i<n; j++)
                      F=F-[i*j];
    for (i=2; i<n; i++)
          if (i in F)
                print i;
}
```

Масиви

Прост пример за използване на масив: Реализация на алгоритъма ***сито на Ератостен*** за намиране на всички прости числа в интервал от о до ***n***:

```
#include <iostream>
using namespace std;
const int n=100;
int main()
{
	int i, j, a[n];
	for (i=2; i<n; i++) a[i]=1;
	for (i=2; i<n; i++)
		if (a[i])
			for (j=i; i*j<n; j++)  a[i*j]=0;
	cout<<"1 ";
	for (i=2; i<n; i++)
		if (a[i]) cout<<i<<' ';
	return 0;
}
```

Динамично заделяне на памет за масив:

```
#include <iostream>
using namespace std;
void main()
{
        int n, i, j;
        cout<<"Input n:";
        cin>>n;
        int *a=new int [n];
        if (a==NULL)
        {cout<<"Недостиг на памет!";}
        else
          {     for (i=2; i<n; i++)    *(a+i)=1;
                for (i=2; i<n; i++)
                        if (*(a+i))
                                for (j=i; i*j<n; j++)     *(a+i*j)=0;
                cout<<"1 ";
                for (i=2; i<n; i++)
                if (*(a+i))    cout<<i<<' ';
          }
        delete  []a;
}
```

Недостатъци на статичните структури от данни:

1. Почти всички структури имат фиксиран брой елементи;
2. Структурите трудно се обновяват (добавяне и премахване на елементи);
3. Всички статични структури използват фиксиран обем оперативна памет, която не винаги се използва ефективно.

Динамичните структури от данни не се описват, а се създават (и унищожават) по време на изпълнение на програмата.

-Абстрактни типове (структури) данни

При работа с типове (структури) данни съществуват два подхода:

- **стандартен** (конвенционален) , при който един тип данни се определя чрез начина на представянето му в паметта на компютъра;

- **абстрактен**, при който един тип данни се дефинира чрез операции, които могат да се извършват с обекти от този тип, без да се отчита, как те се представят в паметта

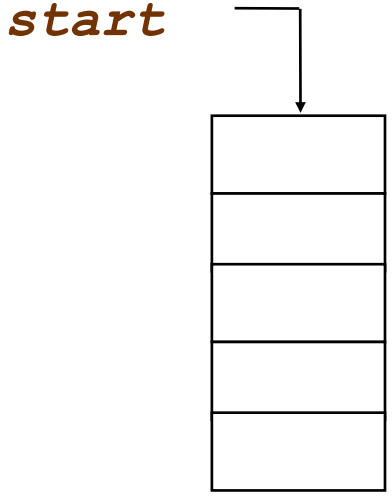
При този подход всеки тип (структура) се представя като тройка множества:

{D, F, A}

D – области;
F - функции, които съществуват и се променят в D;
A - аксиоми, определящи свойствата на F.

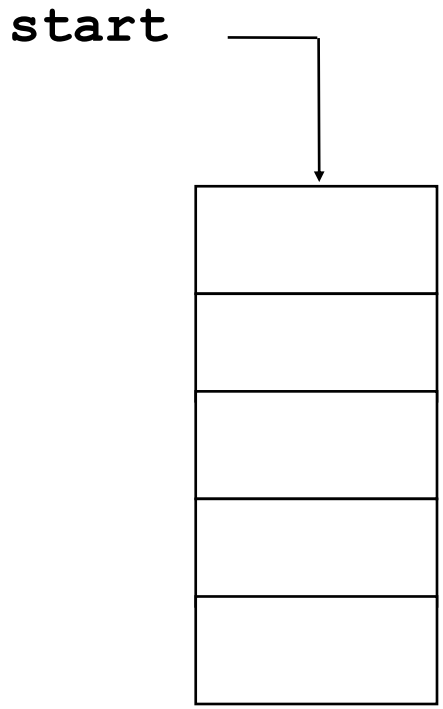
Стек (stack)

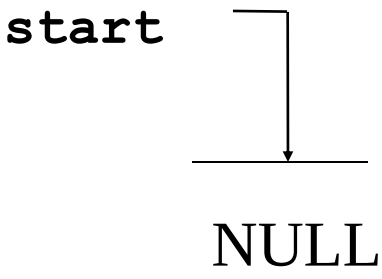
Стекът е подредена последователност от еднотипни елементи с определена дисциплина за тяхно добавяне и извличане. Дисциплината се нарича **LIFO** (**L**ast **I**n **F**irst **O**ut - последен влязъл- първи излязъл). Включването и изключването на елементи се извършва от единия край на структурата, свързан с указател ***start***, наричан "връх" на стека.



Основни операции

- Инициализиране на стека (`init`)
- Добавяне на елемент в стека (`push`)
- Извличане на елемент от стека (`pop`)





Примерна дефиниция на структурата стек (динамична реализация):

```
struct elem
    { int key; elem *next;}
          *start=NULL;
```

```
void push(int n) //добавяне на
                 //елемент в стека
{
  elem *p=start;
  start=new elem;
            //създаване на елемент
  start->key=n;
  start->next=p; // установяване
                 // на нов връх
}
```

```
int pop(int &n)          // извличане на
                         // елемент от стека
{
  if (start)              // проверка за
                          // непразен стек
    {
      n=start->key;
      elem *p=start;
      start=start->next;
      delete p;          //  изтриване на
                         //  достъпния елемент
      return 1;
    }
       else
       return 0;         //  стекът е празен
}
```

Динамична реализация на стек: въвеждане на поредица от цели положителни числа и нейното извеждане върху екрана в обратен ред. За край на поредицата от клавиатурата се въвежда символ, различен от цифра. Поредицата се реализира като стек.

```
#include <iostream>
using namespace std;
void push(int n);                    //prototype
int pop(int &n);                     //prototype
struct elem
   { int key; elem *next;} *start=NULL;
void main()
{
  int num;
  cout<<"Въведете число:\n";
  while (cin>>num)
     {push(num);}
  cout<<"\nСтек:  ";
  while (pop(num))
     {cout<<num<<"  ";}      // cin.clear(); cin.sync();
}
```

Пример. Програмно да се реализира динамичен стек **S** от цели положителни числа, който да се преобразува в два други стека – **P** и **Q**, съответно с четните и нечетните стойности на **S**. Съдържанието на двата стека да се изведе на екрана.

```
#include <iostream.h>
#include <conio.h>
struct elem
  { int key; elem *next;}*start,*pst, *qst;
elem *push(elem *st, int n);    //prototype
elem *pop(elem *st, int &n);   //prototype
void init(elem *st);                 //prototype
int empty(elem *st);               //prototype
```

```
void main()
{
  init(start);
   int num;
   cout<<"Въведете стойност:\n";
   while (cin>>num)
     start=push(start, num);
   init(pst);
   init(qst);
   while (!empty(start))
    {
     start=pop(start, num);
     if (num%2)                  //проверка за четност
     qst=push(qst,num);     //добавяне на елемент в Q
           else
     pst=push(pst,num);     //добавяне на елемент в P
    }
```

```
cout<<"\Новите стекове са:  ";
 cout<<"\nP:  ";
 while (!empty(pst))
   {
    pst=pop(pst,num);
    cout<<num<<' ';
   }
 cout<<"\nQ:  ";
 while (!empty(qst))
   {
    qst=pop(qst,num);
    cout<<num<<' ';
   }
 cout<<"\n";
 system("pause");
}
```

```
void init (elem *st)            //инициализация на стек
{
  st=NULL;
}

int empty(elem *st)             //проверка за празен стек
{
  if (st==NULL)
    return 1;
      else
    return 0;
}
```

```
elem *push(elem *st, int n)    //добавяне на елемент
{
  elem *p;
  p=new elem;
  p->key=n;
  p->next=st;
  st=p;
  return p;
}
```

```
elem *pop(elem *st, int &n)     //извличане на елемент
{
    elem *p;
    n=st->key;
    p=st;
    st=st->next;
    delete p;
    return st;
}
```

Статична реализация на стек: от клавиатурата се въвеждат цели стойности. Въвеждането се прекратява при натискане на клавиш, различен от цифра, или при надхвърлянето на зададения размер на масива.

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

void push(int n);          //prototype
int pop(int n);            //prototype

const int s_size=10;       //размер на стека
int st[s_size];            //масив-стек
int i=0;                   //индекс
```

```
void main()
{
  int num;
  cout<<"Въведете число:\n";
  while((i<s_size)&&(cin>>num))
     {push(num);}
  cout<<"\nСтек: ";
  while (i>0)
     {cout<<pop(num)<<"  ";}
  }
void push(int n)    //включване на елемент
{
 st[i++]=n;
}
int pop(int n)      //изключване на елемент
{
     n=st[--i];
     return n;
}
```

Пример:

```
//Пример за СТЕК
//Изчисляване на прост аритметичен израз, включващ операции + и *
//върху цели числа <10:
//Например: (5*(((2+3)*(1*2))+7))

#include <iostream>
using namespace std;
#include <string.h>
#include <stdlib.h>
void push(char n);   //prototype
char pop();
struct elem
      { char key; elem *next;} *start, *p;
char in[30],  post[30];
char c,  cc;
int k;
void init();   //prototype
int empty(elem *st);
void in_post(char in[30], int l);
void calc(char post[30]);
```

```
void main()
{
  cout<<"\nInput expression: ";
  cin>>in;
  k=strlen(in);
  in_post(in, k);
  cout<<"Postfix form: "<<post;
  calc(post);
}
void init()
{
  start=NULL;
}
int empty(elem *st)
{
  if (st==NULL)
    return 1;
      else
    return 0;
}
```

```
void push(char n)
{
     p=start;
     start=new elem;
     start->key=n;
     start->next=p;
}
char pop()
{
      char t;
      if (start)
        {
           t=start->key;
            p=start;
            start=start->next;
            delete p;
            return t;
        }
   else
    return 0;
}
```

```
void in_post(char in[30], int l)
{
  init();
  int j=0;
  for (int i=0;  i<l;  i++)
   {
        if (in[i]==')')
             {post[j]=pop(); j++;}
        if ((in[i]=='+')||(in[i]=='*'))
              push(in[i]);
        if ((in[i]>='0')&&(in[i]<='9'))
             {post[j]=in[i]; j++;}
   }
}
```

```
void calc(char post[30])
{
  char p[40]; int j=0;
  int k=strlen(post);
  for (int i=0; i<k; i++)
    {p[j]=post[i]; p[++j]=' ';j++;}  // разделяне на цифри и знаци с интервал
  p[j]='\0';
  cout<<"\nNew expression:"<<p;
  k=strlen(p);
  init();
  for (i=0; i<k; i++)
    {
     if (p[i]=='+')
            push(pop()+pop());
     if (p[i]=='*')
            push(pop()*pop());
     if ((p[i]>='0')&&(p[i]<='9'))
              push(0);
     while ((p[i]>='0')&&(p[i]<='9'))
            push(10*pop()+(p[i++]-'0'));
    }
  cout<<"\nResult: "<<unsigned(pop())<<'\n';
}
```

Input expression: ((6+4)*7)

Postfix form: 64+7*

New expression: 6 4 + 7 *

Result: 70

или

Input expression: (5*(((2+3)*(1*2)+7))

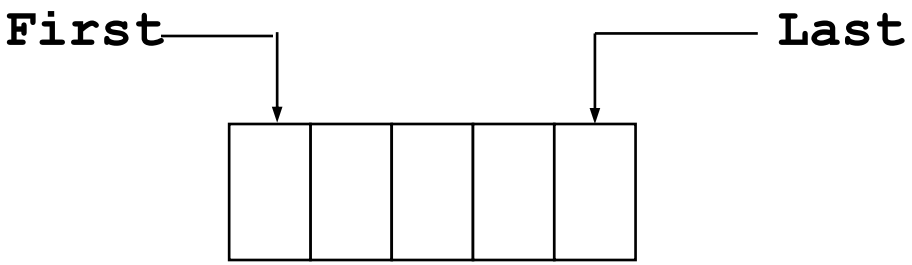
Postfix form: 523+ 12**7+*

New expression: 5 2 3 + 1 2 * * 7 + *

Result: 85

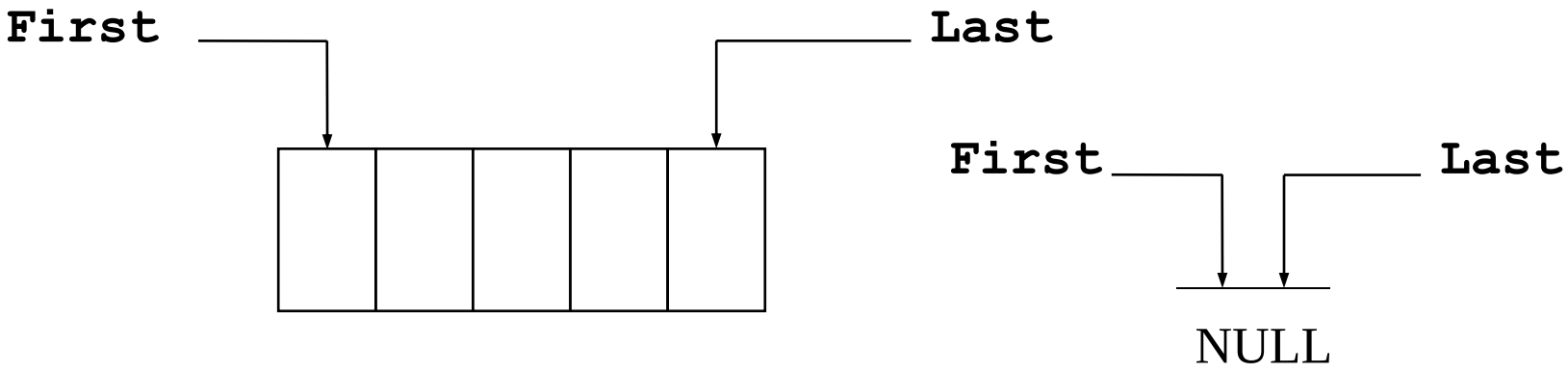
Опашка (Queue)

Опашката е подредена последователност от еднотипни елементи с определена дисциплина за тяхното добавяне и извличане. Дисциплината се нарича **FIFO** (**F**irst **I**n **F**irst **O**ut - първи влязъл - първи излязъл). Структурата опашка се задава с два указателя - **First** и **Last** за нейните начало и край. Включването се извършва от края на опашката (**Last**), а изключването на елементи - от нейното начало (**First**):



Основни операции със структурата

1. Инициализиране на опашката (`init`).
2. Добавяне на елемент в края на опашката (`push`).
3. Извличане на елемент от началото на опашката (`pop`).



Опашката подобно на стека може да има както динамична, така и статична реализация.

Примерна дефиниция на структурата опашка (динамична реализация):

```
struct elem
{ int key; elem *next;}
   *first, *last;
```

```
void push(int n)        // добавяне на елемент
{
  elem *p=last;
  last=new elem;
  last->key=n;
  last->next=NULL;
  if(p!=NULL) p->next=last;
  if (first==NULL)      // добавяне на първ
                        // елемент

     {first=last;}
}
```

```
int pop(int &n)          // извличане на елемент
{
  if (first)             // проверка за непразна
                         // опашка
    {
      n=first->key;
      elem *p=first;
      first=first->next;
      delete p;          //премахване на елемента
      return 1;
    }
      else
       return 0;         //опашката е празна
}
```

Динамична реализация на опашка, в която се съхраняват цели числа, задавани от потребителя:

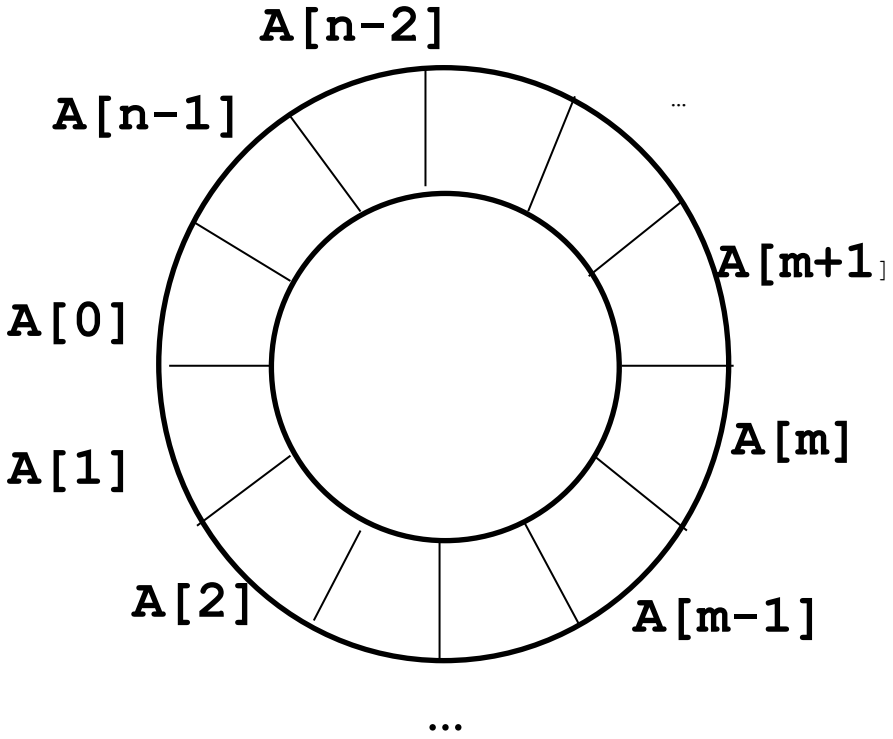
```
#include <iostream>
using namespace std;
void push(int n);   //prototype
int pop(int &n);  //prototype
struct elem    //дефиниране и инициализация на опашка
   { int key; elem *next;} *first=NULL, *last=NULL;

void main()
{
  int num;
  cout<<"\n Въведете число:";       //  '/' за край на
                                        въвеждане
  while (cin>>num)
     push(num);
  cout<<"\nБуферирани данни:  ";
  while (pop(num))
     {cout<<num<<"  ";}
  cout<<"\n";
}
```

Статична реализация на опашка чрез масив:

```
#include <iostream>
using namespace std;
void push(int n);                       //prototype
int pop(int n);                         //prototype
const int q_size=10;                    //размер на опашката
int que[q_size];                        //масив-опашка
int first=0, last=0;   //индекси на първия и посл.ел-ти
void main()
{  int num;
   cout<<"Въведете число:\n";
   while((last<q_size)&&(cin>>num)) //за край-
       {push(num);}                      //произволен символ
   cout<<"\nБуферирани данни: ";
   while (first<last)
      {cout<<pop(num)<<"  ";}
  cout<<"\n";
}
void push(int n)                        //добавяне на елемент
       {que[last++]=n;}
int pop(int n)       //извличане на достъпния елемент
      {n=que[first++]; return n;}
```

Статичната реализация на опашка чрез използване на кръгов масив (масив, който имитира затворен кръг). Подобен подход позволява многократно добавяне и извличане на елементи от опашката. Буферът се препълва само при брой елементи, по-голям от n - размера на масива.



```
#include <iostream>
using namespace std;
void push(int n);           //prototype
int pop(int n);             //prototype
void init();                //prototype
const int q_size=15;        //Размер на буфера
int q[q_size], first, last, counter;
void main()
{
  init();
  int num, rep;
  do
  {cout<<"1 - Добавяне на стойност\n";
   cout<<"2 - Извеждане на стойност\n";
   cout<<"3 - Край на работа\n";
   cout<<"Изберете:";
   cin>>rep;
   switch (rep)
     { case(1):
       {
         if(counter<q_size)
           {cout<<"Въведете число:";cin>>num;
            push(num);counter++; break;}
             else cout<<"Буферът е препълнен!\n";break;
       }
```

```
case(2):
          { if (counter)
            {cout<<"num="<<pop(num)<<'\n';counter--;}
               else cout<<"Буферът е празен!\n";}
    }
 }
 while (rep!=3);
   cout<<"\n";
}
void push(int n)
{
 q[last]=n; last=(last+1)%q_size;
}
int pop(int n)
{
     n=q[first]; first=(first+1)%q_size;
     return n;
}
void init()
{
 first=last=0;
}
```

Видове опашки

- FIFO - опашка (класическа)
- приоритетна опашка
- отборна опашка
- случайна опашка

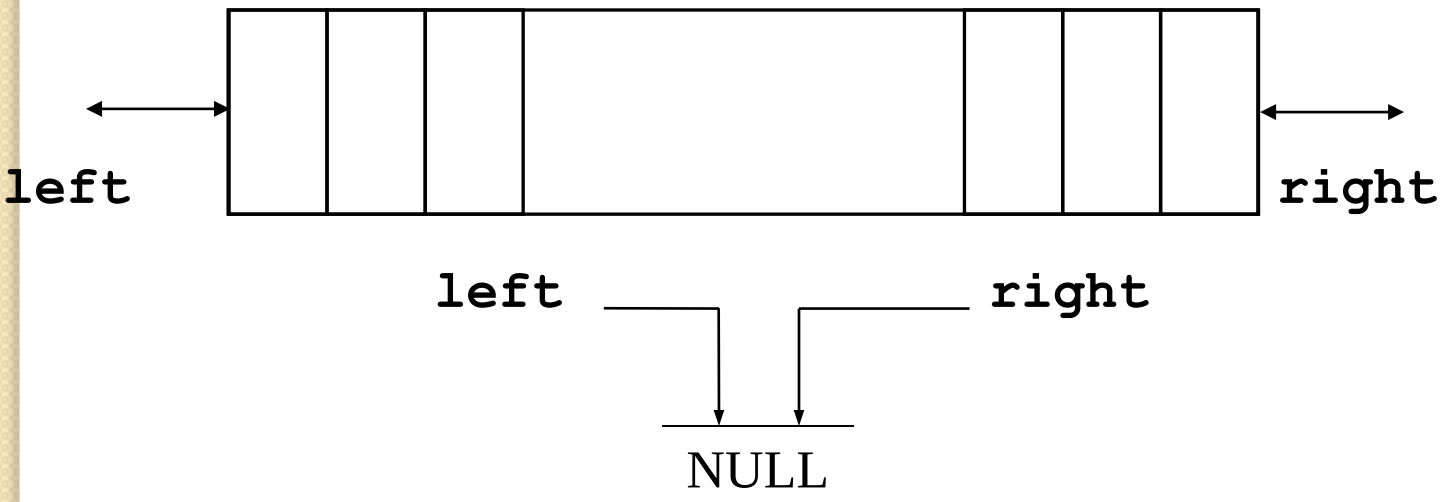
ДЕК (Deque)

Декът е подредена последователност от еднотипни елементи с възможност за добавяне на нови и извличане на съществуващи от двата края на структура. Възможен е частен случай - ограничен дек, при който в единия му край се допуска добавяне и извличане, а в другия - само една от операциите:

Стекът и опашката със своите дисциплини за достъп **FIFO** и **LIFO** могат да се разглеждат като частен случай на дек.

Основните операции със структурата са:

- Инициализиране на дек (**`init`**).
- Добавяне на елемент в единия от двата края на дека (**`push`**).
- Извличане на елемент от единия от двата края на дека (**`pop`**).



Примерна дефиниция и инициализация на структурата дек:

```
struct elem
   { int key; elem *next;} *left=NULL,
*right=NULL;
```

Добавяне на нов елемент отляво:

```
void push_l(int n)//добавяне отляво
{
  elem *p=left;
  left=new elem;
  left->key=n; left->next=p;
  if (right==NULL) {right=left;}
}
```

Добавяне на нов елемент отдясно:

```
void push_r(int n)  //добавяне отдясно
{
  elem *p;
  p=right;
  right=new elem;
  right->key=n;
  right->next=NULL;
  if (left==NULL)
      left=right;
    else
      p->next=right;
}
```

Премахване на елемент отляво:

```
int pop_l(int &n)      //премахване отляво
{ elem *p;
  if (left)
    {
      n=left->key;
      p=left;
      left=left->next;
      if (left==NULL)
        right=NULL;
      delete p;
      return 1;
    }
      else
       return 0;
}
```

Премахване на елемент отдясно:

```
int pop_r(int &n)          //премахване отдясно
{ elem *p;
  if (right)
    { n=right->key;
      if (left==right)
       { delete right;
         left=right=NULL;
       }
       else
        { p=left;
         while (p->next!=right)
                    p++;
         p->next=NULL;
         delete right;
         right=p;
        }
       return 1;
    }     else  return 0;
}
```

Примерна програма за работа с дек, съдържащ цели числа. На потребителя се предоставя възможност да избира операция за изпълнение.

```
#include <iostream>
using namespace std;
void push_l(int n);    //prototype
void push_r(int n);    //prototype
int pop_l(int &n);     //prototype
int pop_r(int &n);     //prototype

struct elem
 { int key; elem *next;} *left=NULL,
                         *right=NULL;
```

```
void main()
{
  int ch;
  do
 {
  int num;
  cout<<"          Меню:\n";
  cout<<"1 - Добавяне отляво\n";
  cout<<"2 - Добавяне отдясно\n";
  cout<<"3 - Извличане отляво\n";
  cout<<"4 - Извличане отдясно\n";
  cout<<"5 - Край на работа\n";
  cout<<"Избор: ";
  cin>>ch;
```

```
switch (ch)
  {
   case (1):
   case (2):
    cout<<"\Въведете число: ";
    cin>>num;
    if (ch==1)
      push_l(num);
           else
      push_r(num);
    break;
   case (3):
     {
      if (pop_l(num))
       {
        cout<<num<<"\n";
       }
         else
      { cout<<"Празна структура!\n";}
       break;
     }
```

```
case (4):
      { if(pop_r(num))
       {
        cout<<num<<"\n";
       }
         else
       {cout<<"Празна структура!"; }
      }
   }
  }
 while (ch!=5);
}
```