Obyektyönlü proqramlaşdırma

C++

ШАГ
компьютерная
АКАДЕМИЯ

Dərs №8

C++ dili ilə obyektyönlü proqramlaşdırma

Mündəricat

Təkəlaqəli siyahı

Bugün biz sizinlə ölçüsüz massifi xatırladan dinamik verilənlər strukturu ilə tanış olacağıq. Bu struktur **siyahı** adlanır. Siyahıların bir neçə forması vardır. İlk olaraq biz **təkəlaqəli və ya biristiqanətli siyahıya** baxacağıq.

Təkəlaqəli siyahı — hər biri iki hissədən ibarət olub siyahının elementini əks etdirən bir neçə obyektlər dəstidir. Elementin ilk hissəsi onun saxladığı qiymətdir, ikinci hissəsi isə siyahının növbəti elementinin göstəricisidir.

Növbəti element haqqında informasiyanın xarakteri siyahının konkret olaraq harada yerləşməsindən asılıdır. Məsələn, əgər bu əməli yaddaşdırsa, onda informasiya növbəti elementi əks etdirəcək, əgər fayldırsa – növbəti elementin fayldakı mövqeyini. Biz sizinlə əməli yaddaşda saxlanılan siyahının reallaşdırılmasına baxacağıq, lakin istəyinizə görə siz siyahının faylda saxlanılması üçün xüsusi kod yaza bilərsiniz. Beləliklə, başlayaq: Siyahının hər bir elementini iki tərkib hissəsindən (komponentdən) ibarət olan struktur vasitəsi ilə verəcəyik:

1. Saxlamaq üçün nəzərdə tutulan əsas informasiyanı ehtiva edən bir və ya bir neçə sahə.
2. Siyahının növbəti elementinin göstəricisini saxlamaq üçün sahə.

Bu cür strukturun ayrı-ayrı elementlərini biz sonradan onları növbəti elementlərin göstəricilərini saxlayan sahələr vasitəsi ilə əlaqələndirərək düyünlər adlandıracağıq.

```
//Siyahının düyünü
struct node
{
    //Düyünün informasiya hissəsi
    int value;

    //Növbəti elementin göstəricisi
    node *next;
};
```

Strukturun yaradılmasından sonra, biz onun obyektlərini hansısa bir sinfə inkapsulyasiya etməliyik, bu da bizə siyahını məqsədli konstruksiya vasitəsi ilə idarə etməyə imkan verəcək. Sinif iki göstərici (siyahının "başı" və ya əvvəli və siyahının "quyruğu" və ya sonu), Siyahı ilə işləmək üçün funksiyalar dəsti ehtiva edəcək.

```
    //baş
    node *phead;

    //quyruq
    node *ptail;
```

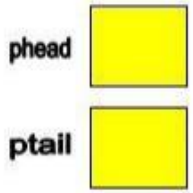
Ümumi olaraq, alınmış siyahını növbəti şəkildə təsvir etmək olar:

Beləliklə, biz siyahının yaradılmasının əsas mərhələlərinə baxdıq, indi isə onun birbaşa formalaşdırılmasına keçirik.

Siyahının formalaşdırılması

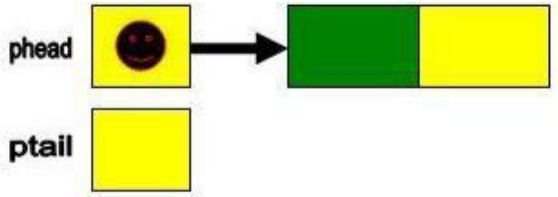
1. Statik yaddaşda göstəricilər üçün yer ayıraq.

```
node *phead;
node *ptail;
```



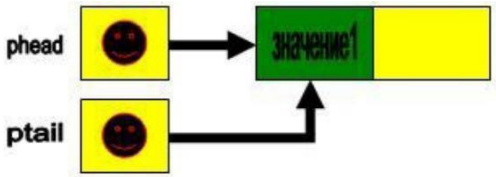
2. Dinamik obyekt üçün yer ayırırıq.

```
phead=new node;
```



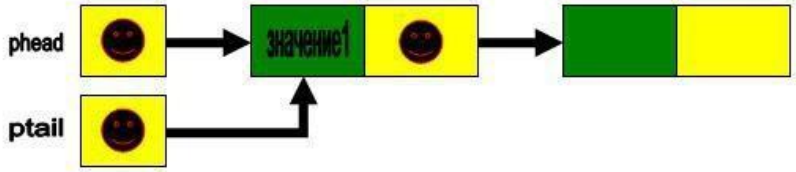
3. ptail dəyişəninin qiymətini mənimsədək və informasiya sahəsinə elementin qiymətini yerləşdiriək.

```
ptail = phead;
ptail->value = "qiymət1";
```



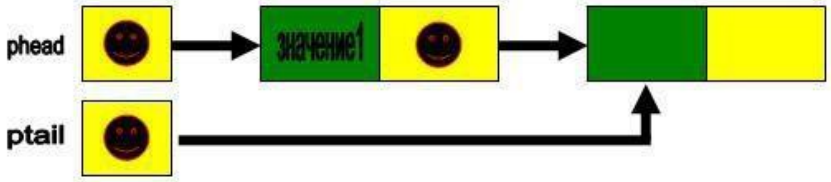
4. Düyün sahəsinə daha bir – yeni dinamik obyektin ünvanını yerləşdirək.

```
ptail->next = new node;
```



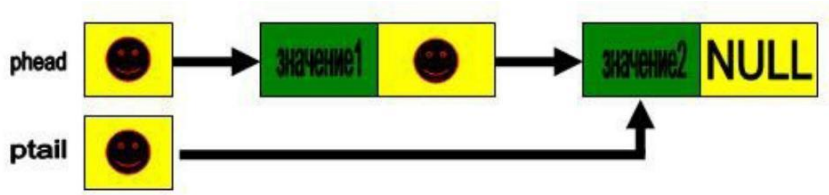
5. ptail dəyişəni sonuncu əlavə edilmiş elementin ünvanını saxlamalıdır, belə ki, o sona əlavə edilmişdir.

```
ptail = ptail->next;
```



6. Əgər siyahının qurulmasını tamamlamaq tələb olunursa, onda sonuncu elementin göstərici sahəsinə NULL yazmaq lazımdır.

```
ptail->next = NULL;
ptail->value = "qiymət2";
```

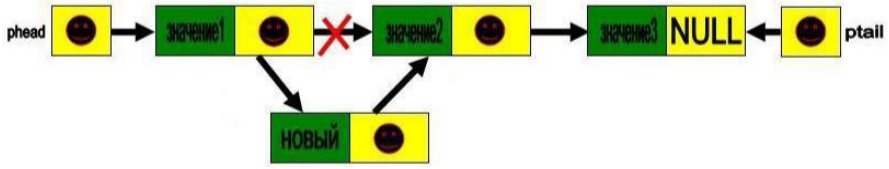



Nəticədə iki düyün ehtiva edən xətti təkəlaqəli siyahı qurulmuş olur.

Siyahının müəyyən veilmiş yerinə düyün əlavə edilməsi

Qeyd: Burada və bundan sonra kod fraqmentləri verməyəcəyik, çünki onlar çox böyükdürlər. Sonra isə biz sizinlə təkəlaqəli siyahının tam reallaşdırılmasına aid misala baxacağıq. İndi isə, şəkillər vasitəsi ilə siyahı üzərində əməliyyatları nəzərdən keçirək.

1. Yeni düyün üçün yaddaş ayırmaq.
2. Yeni düyünə qiymət yazmaq.
3. Yeni düyündən sonra yerləşəcək düyünün ünvanını növbəti ünvan göstəricisinə yazmaq.
4. Yeni düyündən əvvəl yerləşəcək düyündə olan ünvanı yeni düyünün ünvanı ilə əvəzləmək.



Siyahıdan düyünün silinməsi

1. Silinəcək düyündən sonrakı düyünün ünvanını silinəcək düyündən əvvəlki düyünün növbəti ünvan göstəricisinə yazmaq.
2. Silinmək üçün nəzərdə tutulan düyünü silmək.

Nəhayət, təcrübəyə keçə bilərik – dərsin növbəti bölməsində misala baxaq.

Təkəlaqəli siyahının reallaşdırılması

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

struct Element
{
    //Verilənlər
    char data;
    //Siyahının növbəti elemntinin ünvanı
    Element * Next;
};

//Təkəlaqəqli siyahı
class List
{
    //Siyahının başlanğıc elementinin ünvanı
    Element * Head;
    //Siyahının başlanğıc elementinin ünvanı
    Element * Tail;
    //Siyahının elementlərinin sayı
    int Count;

public:
    //Konstruktor
    List();
    //Destruktor
    ~List();

    //Siyahıya elementin əlavə edilməsi
    //(yeni element sonucu element olur)
    void Add(char data);

    //Siyahının elementinin silinməsi
    //(Başlanğıc element silinir)
    void Del();
    //Bütün siyahının silinməsi
    void DelAll();

    //Siyahının məzmununun ekrana verilməsi
    //(Çap başlanğıc elementdən başlayır)
    void Print();

    //Siyahıda olan elementlərin
    //sayının əldə edilməsi
    int GetCount();
};

List::List()
{
    //Başlanğıcda sıyahı boşdur
    Head = Tail = NULL;
    Count = 0;
}

List::~List()
{
    //Silmə funksiyasının çağırılması
    DelAll();
}
```

```
int List::GetCount()
{
    //Elementlərin sayını qaytarırıq
    return Count;
}

void List::Add(char data)
{
    //Yeni elementin yaradılması
    Element * temp = new Element;

    //verilənlərlə doldurulması
    temp->data = data;
    //Növbəti element yoxdur
    temp->Next = NULL;
    //yeni element əgər o ilk əlavə edilən element
    //deyilsə, onda o sonuncu element olur
    if(Head!=NULL){
        Tail->Next=temp;
        Tail = temp;
    }
    //yeni element ilk əlavə edilən
    //elementdirsə, onda o yeganə element olur
    else{
        Head=Tail=temp;
    }
}

void List::Del()
{
    //Başlanğıc  elementin ünvanını saxlayırıq
    Element * temp = Head;
    //növbəti elementin başlığını mənimsədirik
    Head = Head->Next;
    //əvvəlki başlıq elementini silirik
    delete temp;
}
```



```
void List::DelAll()
{
    //Nə qədər ki, element var
    while(Head != 0)
        //Elementləri tək-tək silirik
        Del();
}

void List::Print()
{
   //Başlanğıc elementin ünvanını saxlayırıq
    Element * temp = Head;

    //Nə qədər ki, element var
    while(temp != 0)
    {
        //Verilənləri daxil edirik
        cout << temp->data << " ";
        //Növbəti elementə keçirik
        temp = temp->Next;
    }

    cout << "\n\n";
}

//Test nümunəsi
void main()
{
    //Sinfin obyektini yaradırıq
    List List lst;

    //Test sətri
    char s[] = "Hello, World !!!\n";
    //Sətrin ekrana verilməsi
    cout << s << "\n\n";
    //Sətrin uzunluğunu təyin edirik
    int len = strlen(s);
```

```
    //Sətri siyahıya əlavə edirik
    for(int i = 0; i < len; i++)
        lst.Add(s[i]);
    //Siyahının məzmununu ekrana veririk
    lst.Print();
    //Siyahının üç elementini silirik
    lst.Del();
    lst.Del();
    lst.Del();
    //Siyahının məzmununu ekrana veririk
    lst.Print();
}
```


İkiəlaqəli siyahı

Siz artıq təkəlaqəli siyahının olduğunu bilirsiniz, buna görə də sizə ikiəlaqəli siyahının iş prinsipini anlamaq asan olacaq. Bu strukturun əvvəlkindən fərqi ondan ibarətdir ki, ikiəlaqəli (və ya ikiistiqamətli) siyahıda hər bir düyün iki deyil, üç hissədən ibarət olur. Üçüncü komponentdə əvvəlki elementin göstəricisi saxlanılır.

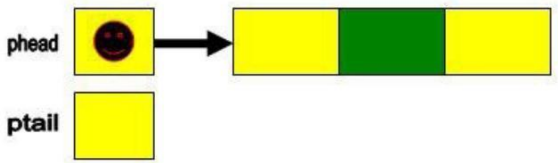
```
//siyahının düyünü
struct node
{
    //siyahının düyününün informasiya elementi
    int value;

    //Siyahının əvvəlki düyününün göstəricisi
    node *prev;

    //Siyahının sonrakı düyününün göstəricisi
    node *next;
};
```

Bu cür düzülüş siyahıda həm düz, həm də əks istiqamətdə hərəkət etməyə imkan verir.

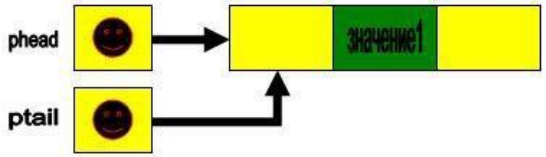
İkiəlaqəli siyahı üzərində əsas əməliyyatları şəkillərlə gözdən keçirək.

İkiəlaqəli siyahının formalaşması

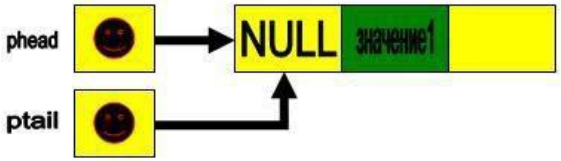
1. Statik yaddaşda göstəricilər üçün yer ayıraq və dinamik obyekt üçün yer saxlayaq.



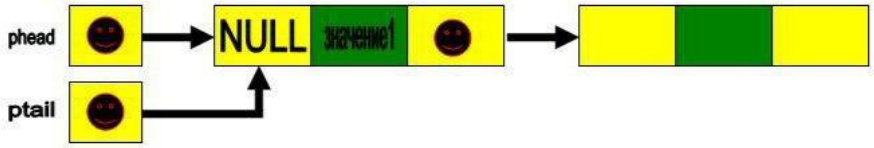
2. ptail dəyişəninə qiymət mənimsədək və informasiya sahəsinə elementin qiymətini yerləşdirək.



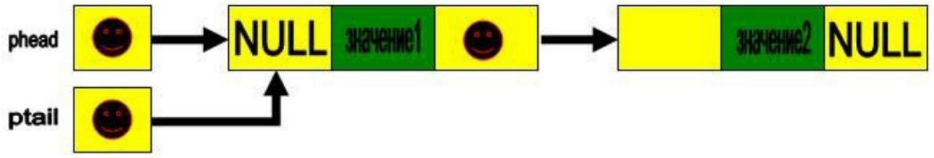
3. Əvvəlki elementin göstəricisinə NULL mənimsədək (yəni, element ilkdir – ondan əvvəl element yoxdur.



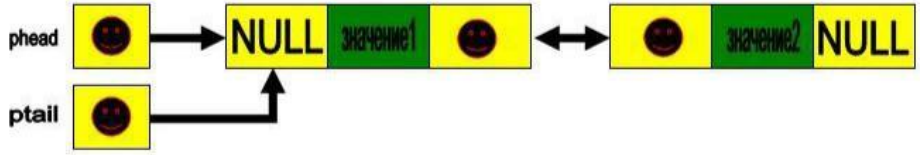
4. Düyünün ünvan sahəsinə daha bir – yeni dinamik obyektin ünvanını yerləşdirək.



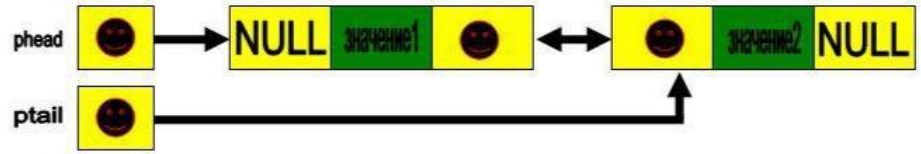
5. Əlavə edilmiş yeni obyektə qiymət yazırıq, növbəti düyünün ünvan göstəricisinə NULL yazırıq, belə ki, obyekt siyahının sonuna əlavə edilir.



6. Əvvəlki elementin göstəricisinə əvvəlki obyektin ünvanı yazılır.



7. ptail dəyişəni sonuncu əlavə edilmiş elementin ünvanını saxlamalıdır, belə ki, o sona əlavə edilmişdir.

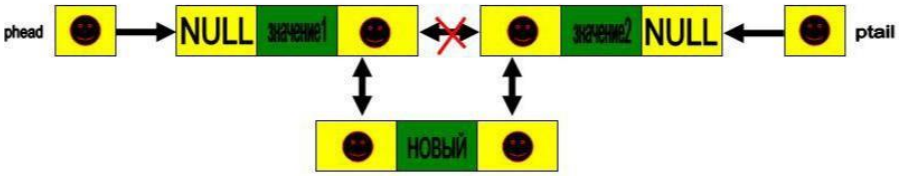


İki elementli ikiəlaqəli siyahı hazırdır.

İkiəlaqəli siyahıya elementin əlavə edilməsi

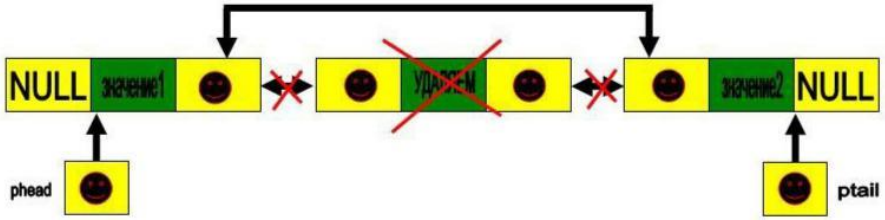
1. Yeni düyün üçün yaddaşın ayrılması.
2. Yeni düyünə qiymət yazmaq.
3. Növbəti düyünün göstəricisinə yeni düyündən əvvəl yerləşəcək düyünün ünvanını yazmaq.

4. Növbəti düyünün göstəricisinə yeni düyündən sonra yerləşəcək düyünün ünvanını yazmaq.
5. Əvvəlki düyündə növbəti düyünün ünvan göstəricisinin qiymətini yeni düyünün ünvanı ilə əvəzləyirik.
6. Növbəti düyündə əvvəlki düyünün ünvan göstəricisinin qiymətini yeni düyünün ünvanı ilə əvəzləyirik.



İkiəlaqəli siyahıdan elementin silinməsi

1. Silinən düyündən sonrakı düyünün ünvanını silinən düyündən əvvəlki düyünün növbəti elementin ünvan göstəricisinə yazmaq.
2. Silinən düyünün əvvəlindəki düyünün ünvanını silinən düyündən sonrakı düyünün əvvəlki ünvan göstəricisinə yazmaq.
3. Silinmək üçün nəzərdə tutulmuş düyünü silmək.



Bu da son – indi də dərsin növbəti bölməsinə keçək və misala baxaq.

İkiəlaqəli siyahının reallaşdırılması

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

struct Elem
{
    int data; //verilənlər
    Elem * next, * prev;
};

class List
{
    //Başlıq, quyruq
    Elem * Head, * Tail;
    //Elementlərin sayı
    int Count;

public:
    //Konstruktor
    List();
    //Köçürmə konstruktoru
    List(const List&);
    //Destruktor
    ~List();

    //Sayı əldə etmək
    int GetCount();
    //Siyahının elementini əldə etmək
    Elem* GetElem(int);

    //Bütün siyahını silmək
    void DelAll();
    //Elementin silinməsi, əgər parametr verilməyibsə,
    //onda funksiya onu soruşur

    void Del(int pos = 0);
```

```
    //Elementin əlavə edilməsi, əgər parametr
    //verilməyibsə, onda funksiya onu soruşur
    void Insert(int pos = 0);

    //Siayhının sonuna əlavə etmək
    void AddTail(int n);

    //Siayhının əvvəlinə əlavə etmək
    void AddHead(int n);

    //Siyahının çapı
    void Print();
    //Müəyyən elementin çapı
    void Print(int pos);

    List& operator = (const List&);
    //iki siyahının birləşdirilməsi
    List operator + (const List&);

    //elementlərə görə müqayisə
    bool operator == (const List&);
    bool operator != (const List&);
    bool operator <= (const List&);
    bool operator >= (const List&);
    bool operator < (const List&);
    bool operator > (const List&);

    //Siyahının çevrilməsi
    List operator - ();
};

List::List()
{
    //Başlanğıcda siyahı boşdur
    Head = Tail = NULL;
    Count = 0;
}
```



```
List::List(const List & L)
{
    Head = Tail = NULL;
    Count = 0;

    //Siyahının başlığından köçürürük
    Elem * temp = L.Head;
    //Siyahının sonu deyilsə
    while(temp != 0)
    {
        //verilənləri ikiyə bölürük
        AddTail(temp->data);
        temp = temp->next;
    }
}

List::~List()
{
    //Bütün elementləri silirik
    DelAll();
}

void List::AddHead(int n)
{
    //yeni element
    Elem * temp = new Elem;

    //Əvvəlki yoxdur
    temp->prev = 0;
    //Verilənləri doldururuq
    temp->data = n;
    //Növbəti - əvvəlki başlıq
    temp->next = Head;

    //Əgər elementlər varsa?
    if(Head != 0)
        Head->prev = temp;
```

```
    //Əgər element ilkdirsə, onda o eyni
    //zamanda həm başlıq, həm də quyruqdur
    if(Count == 0)
        Head = Tail = temp;
    else
        //əks halda yeni element - başlıq
        Head = temp;

    Count++;
}

void List::AddTail(int n)
{
    //Yeni element yaradırıq
    Elem * temp = new Elem;
    //Növbəti element yoxdur
    temp->next = 0;
    //Verilənləri doldururuq
    temp->data = n;
    //Əvvəlki - köhnə quyruq
    temp->prev = Tail;

    //Əgər elementlər varsa?
    if(Tail != 0)
        Tail->next = temp;

    //Əgər element ilkdirsə, onda həm
    //başlıq, həm də quyruqdur
    if(Count == 0)
        Head = Tail = temp;
    else
        //əks halda yeni element - quyruq
    Tail = temp;

    Count++;
}

void List::Insert(int pos)
{
```



```
    //əgər parametr yoxdursa və ya sıfra
    //bərabərdirsə, onda onu soruşuruq
    if(pos == 0)
    {
        cout << "Input position: ";
        cin >> pos;
    }

    //1-dən Count-a qədər mövqe?
    if(pos < 1 || pos > Count + 1)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
        return;
    }

    //Siyahının sonuna əlavə edilərsə
    if(pos == Count + 1)
    {
        //Verilənləri əlavə edirik
        int data;
        cout << "Input new number: ";
        cin >> data;
        //Siayahının sonuna əlavə edilməsi
        AddTail(data);
        return;
    }
    else if(pos == 1)
    {
        //Verilənləri əlavə edirik
        int data;
        cout << "Input new number: ";
        cin >> data;
        //Siyahının əvvəlinə əlavə etmə
        AddHead(data);
        return;
    }
```

```
    //Sayğac
    int i = 1;
    //Başlıqdan başlayaraq n - 1 element

    //sayırıq

    Elem * Ins = Head;

    while(i < pos)
    {
        //Əvvəlinə əlavə edəcəyimiz elementə çatırıq
        Ins = Ins->next;
        i++;
    }

    //Qabaqda olan elementə çatırıq

    Elem * PrevIns = Ins->prev;

    //Yeni element yaradırıq
    Elem * temp = new Elem;

    //Verilənləri əlavə edirik
    cout << "Input new number: ";
    cin >> temp->data;

    //Əlaqələrin sazlanması
    if(PrevIns != 0 && Count != 1)
        PrevIns->next = temp;

    temp->next = Ins;
    temp->prev = PrevIns;
    Ins->prev = temp;

    Count++;
}

void List::Del(int pos)
{
```



```
    //əgər parametr yoxdursa və ya sıfra
    //bərabərdirsə, onda onu soruşuruq
    if(pos == 0)
    {
        cout << "Input position: ";
        cin >> pos;
    }
    //1-dən Count-a qədər mövqedirsə?
    if(pos < 1 || pos > Count)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
        return;
    }

    //Sayğac
    int i = 1;

    Elem * Del = Head;

    while(i < pos)
    {
        //Silinəcək elementə qədər gedirik

        Del = Del->next;
        i++;
    }

    //Silinəcək elementən əvvəlki
    //elementə qədər gedirik
    Elem * PrevDel = Del->prev;
    //Silinəcək elementdən sonrakı elementə qədər

    //gedirik

    Elem * AfterDel = Del->next;

    //Əgər silinən başlıq deyilsə
    if(PrevDel != 0 && Count != 1)
```

```
        PrevDel->next = AfterDel;
    //Silinən quyruq deyilsə
    if(AfterDel != 0 && Count != 1)
        AfterDel->prev = PrevDel;

    //Qıraqdakılar silinirsə?
    if(pos == 1)
        Head = AfterDel;
    if(pos == Count)
        Tail = PrevDel;

    //Elementin silinməsi
    delete Del;

    Count--;
}

void List::Print(int pos)
{
    //1-dən Count-a qədər mövqedirsə?
    if(pos < 1 || pos > Count)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
        return;
    }

    Elem * temp;
    //Hansı tərəfdən daha tez hərəkət
    //edilməsini təyin edirik

    if(pos <= Count / 2)
    {
        //Başlıqdan başlayaraq silmə
        temp = Head;
        int i = 1;

        while(i < pos)
        {
```



```
            //Lazım olan elementə qədər getmək
            temp = temp->next;
            i++;
        }
    }
    else
    {
        //Quyruqdan başlayaraq hesablamaq
        temp = Tail;
        int i = 1;

        while(i <= Count - pos)
        {
            //Lazım olan elementə qədər getmək
            temp = temp->prev;
            i++;
        }
    }
    //Elementin çapı
    cout << pos << " element: ";
    cout << temp->data << endl;
}

void List::Print()
{
    //Əgər siyahıda elementlər varsa, onda onun
    //üzərindən keçirik və başlıqdan başlayaraq
    //elementləri çap edirik
    if(Count != 0)
    {
        Elem * temp = Head;
        cout << "( ";
        while(temp->next != 0)
        {
            cout << temp->data << ", ";
            temp = temp->next;
        }
        cout << temp->data << " )\n";
    }
}
```

```
void List::DelAll()
{
    //Nəqədər ki, element qalıb, başlıqdan başlayaraq
    //bir-bir silirik
    while(Count != 0)
        Del(1);

}

int List::GetCount()
{
    return Count;

}

Elem * List::GetElem(int pos)
{
    Elem *temp = Head;

    //1-dən Count-a qədər mövqedirsə?
    if(pos < 1 || pos > Count)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
        return 0;
    }


    int i = 1;
    //Bizə lazım olan elementi axtarırıq
    while(i < pos && temp != 0)
    {

        temp = temp->next;
        i++;

    }
        return 0;
    else
        return temp;
}
```



```
List & List::operator = (const List & L)
{
    //Elementin özünün özünə mənimsədilməsini
    //yoxlayırıq
    if(this == &L)
        return *this;

    //Köhnə siyahının silinməsi
    this->~List(); // DelAll();

    Elem * temp = L.Head;

    //Elementləri köçürürük
    while(temp != 0)
    {
        AddTail(temp->data);
        temp = temp->next;
    }

    return *this;
}

//İki siyahının toplanması (birləşdirilməsi)
List List::operator + (const List& L)
{
    //Müvəqqəti siyahıya birinci siyahının
    //elementlərini əlavə edirik

    List Result(*this);
    //List Result = *this;
    Elem * temp = L.Head;

    //Müvəqqəti siyahıya ikinci siyahının
    //elementlərini əlavə edirik

    {
        Result.AddTail(temp->data);
        temp = temp->next;
    }
```

```
    return Result;
}

bool List::operator == (const List& L)
{
    //Saya görə müqayisə
    if(Count != L.Count)
        return false;

    Elem *t1, *t2;

    t1 = Head;
    t2 = L.Head;

    //Məzmuna görə müqayisə
    while(t1 != 0)
    {
        //Eyni mövqedə olan verilənləri
        //müqayisə edirik
        if(t1->data != t2->data)
            return false;

        t1 = t1->next;
        t2 = t2->next;
    }

    return true;
}

bool List::operator != (const List& L)
{
    //əvvəlki müqayisə funksiyasını istifadə edirik
    return !(*this == L);
}

bool List::operator >= (const List& L)
{
    //Saya görə müqayisə
    if(Count > L.Count)
        return true;
```



```
    //Məzmuna görə müqayisə
    if(*this == L)
        return true;

    return false;
}

bool List::operator <= (const List& L)
{
    //Saya görə müqayisə
    if(Count < L.Count)
        return true;
    //Məzmuna görə müqayisə
    if(*this == L)
        return true;

    return false;
}

bool List::operator > (const List& L)
{
    if(Count > L.Count)
        return true;

    return false;
}

bool List::operator < (const List& L)
{
    if(Count < L.Count)
        return true;

    return false;
}

//çevirmə
List List::operator - ()
{
```

```
    List Result;

    Elem * temp = Head;
    //Siyahının elementlərini başlıqdan başlayaraq
    //köçürürük,elementlər başlığa elə əlavə edilir
    //ki, müvəqqəti Result siyahısı elementləri əks
    //ardıcıllıda saxlayacaq
    while(temp != 0)
    {
        Result.AddHead(temp->data);
        temp = temp->next;
    }

    return Result;
}

//Test misalı
void main()
{
    List L;

    const int n = 10;
    int a[n] = {0,1,2,3,4,5,6,7,8,9};
    //Elementlərin cüt indekslərdə olan
    //yerə, başlığa, tək indeksli olan
    //yerə - quyruğa əlavə edilməsi
    for(int i = 0; i < n; i++)
        if(i % 2 == 0)
            L.AddHead(a[i]);
        else
            L.AddTail(a[i]);

    //Siyahının çapı
    cout << "List L:\n";
    L.Print();

    cout << endl;

    //Elementin siyahıya əlavə edilməsi
    L.Insert();
```



```
    //Siyahının çapı
    cout << "List L:\n"; L.Print();

    //Siyahının 2-ci və 8-ci elementinin çapı
    L.Print(2);
    L.Print(8);

    List T;

    //Siyahını köçürürük
    T = L;
    //Nüsxənin çapı
    cout << "List T:\n";
    T.Print();

    //İki siyahını cəmləyirik (birinci çevrilmiş
    //vəziyyətdə)
    cout << "List Sum:\n";
    List Sum = -L + T;
    //Siyahının çapı
    Sum.Print();
}
```

Sinif şablonları

Deyildiyi kimi, yenə də salam. Biz sizinlə şablonları əvvələr də öyrənmişdik. Lakin bu funksiya şablonları idi. Heç vaxt əldə edilənlə kifayətlənməmək lazımdır. C++ dilində, necə deyərlər, ümumi sinif təyin etmək mümkündür. Bu o deməkdir ki, siz özündə istifadə edilən bütün funksiyaları əks etdirən sinif yarada bilərsiniz, lakin sinif üzvlərinin verilənlər tipi bu sinfin obyektlərinin yaradılması zamanı verilir. Başqa sözlə, sizə müxtəlif tipli verilənlərin qiymətlərini emal edən sinif yazmaq lazım gələrsə, şablonlardan yaxşı vasitə tapa bilməzsiniz.

Sinif üçün şablonun yaradılmasının ümumi sintaksisi növbəti şəkildədir:

```
template <class verilənlər_tipi> class sinfin_adı {
    //....sinfin təsviri.......
};
```

Qeyd: Vacibdir!!! Yaddan çıxarmamaq lazımdır ki, şaplonun tipinin təyini üçün class açar sözünün yerinə biz, əvvəl olduğu kimi, typename istifadə edə bilərik!!!

Şərh

- **Verilənlər_tipi** — vəziyyətdən asılı olaraq, real verilənlər tipi ilə əvəz ediləcək şablonun tip adıdır. Burada bir-birindən vergül ilə ayrılan bir neçə veriənlər tipi parametrləri təyin etmək olar.
- Sinfin təyin edilməsi daxilində şablonun adını istənilən yerdə istifadə etmək olar.

Şablon sinfin reallaşdırılması üçün növbəti sintaksis istifadə edilir:

Şərh

- **Verilənlər_tipi** — şablonun tipi yerinə qoyulacaq real verilənlər tipinin adıdır.

Qeyd: Vacibdir! Şablon sinfin üzv funksiyaları avtomatik olaraq şablondurlar. Bundan başqa, siz bu funksiyaların reallaşdırılmasını sinfin daxilində yazırsınız, onları template açar sözü vasitəsilə şablon kimi elan etmək lazımdır.

Nümunə

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

//parametrləşdirilmiş sinif
template <class T> class TestClass
{ private:
    //tempo sahəsinin elan edilməsi
    //o hansı tipdə olacaq,
    //bunu yalnız konkret sinif nüsxəsinin yaradılması
    //zamanı müəyyən etmək olar
    T tempo;
public:
    TestClass(){tempo=0;}
    //yoxlanılan funksiya
    T testFunc();
};

//TestClass sinfinin üzv-funksiyası
//Metod sinif daxilində reallaşdırılmadığı üçün,
//aşkar xatırlatmadan istifadə edirik
template <class T>
```

```
T TestClass<T>::testFunc() {
    //proqram T tipində tempo dəyişəninin yaddaşda
    //tutduğu baytların sayını ekrana verir
    cout<<"Type's size is:
    "<<sizeof(tempo)<<"\n\n"; return tempo;
}

void main()
{
    //sinfin konkret nüsxələrini yaradırıq
    TestClass //char
    TestClass<char> ClassChar;
    ClassChar.testFunc();
    //int
    TestClass<int> ClassInt;
    ClassInt.testFunc();
    //double
    TestClass<double> ClassDouble;
    ClassDouble.testFunc();
}
```

İkiəlaqəli şablon siyahının reallaşdırılması

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

template <typename T>
struct Elem
{
    // İstənilən verilənlər
    T data;
    Elem * next, * prev;
};

template <typename T>
class List
{
```



```
    // Quyruğun başlığı
    Elem<T> * Head, *
    Tail; int Count;

public:
    List();
    List(const List&);
    ~List();

    int GetCount();
    Elem<T>* GetElem(int);

    void DelAll();
    void Del(int);
    void Del();

    void AddTail();
    void AddTail(T);

    void AddHead(T);
    void AddHead();

    void Print();
    void Print(int pos);

    List& operator = (const List&);
    List operator + (const List&);

    bool operator == (const List&);
    bool operator != (const List&);
    bool operator <= (const List&);
    bool operator >= (const List&);
    bool operator < (const List&);
    bool operator > (const List&);

    List operator - ();
};
```

```
template <typename T>
List<T>::List()
{
    Head = Tail = 0;
    Count = 0;
}

template <typename T>
List<T>::List(const List & L)
{
    Head = Tail = 0;
    Count = 0;

    Elem<T> * temp = L.Head;
    while(temp != 0)
    {
        AddTail(temp->data);
        temp = temp->next;
    }
}

template <typename T>
List<T>::~List()
{
    DelAll();
}

template <typename T>
Elem<T>* List<T>::GetElem(int pos)
{
    Elem<T> *temp = Head;

    //1-dən Count-a qədər movqe?
    if(pos < 1 || pos > Count)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
```



```
        return;
    }

    int i = 1;
    while(i < pos && temp != 0)
    {
        temp = temp->next;
        i++;
    }

    if(temp == 0)
        return 0;
    else
        return temp;
}

template <typename T>
void List<T>::AddHead()
{
    Elem<T> * temp = new Elem<T>;

    temp->prev = 0;

    int n;
    cout << "Input new number: ";
    cin >> n;

    temp->data = n;
    temp->next = Head;

    if(Head != 0)
        Head->prev = temp;

    if(Count == 0)
        Head = Tail = temp;
    else
        Head = temp;
```

```
    Count++;
}

template <typename T>
void List<T>::AddHead(T n)
{
    Elem<T> * temp = new Elem<T>;

    temp->prev = 0;
    temp->data = n;
    temp->next = Head;

    if(Head != 0)
        Head->prev = temp;

    if(Count == 0)
        Head = Tail = temp;
    else
        Head = temp;
    Count++;
}

template <typename T>
void List<T>::AddTail()
{
    Elem<T> * temp = new Elem<T>;

    temp->next = 0;
    int n;
    cout << "Input new number: ";
    cin >> n;

    temp->data = n;
    temp->prev = Tail;

    if(Tail != 0)
        Tail->next = temp;

    if(Count == 0)
```



```
        Head = Tail = temp;
    else
        Tail = temp;

    Count++;
}

template <typename T>
void List<T>::AddTail(T n)
{
    Elem<T> * temp = new Elem<T>;

    temp->next = 0;
    temp->data = n;
    temp->prev = Tail;

    if(Tail != 0)
        Tail->next = temp;

    if(Count == 0)
        Head = Tail = temp;
    else
        Tail = temp;

    Count++;
}

template <typename T>
void List<T>::Del()
{
    int n;
    cout << "Input position: ";
    cin >> n;

    if(n < 1 || n > Count)
    {
        cout << "Incorrect position !!!\n";
        return;
    }
```

```
    int i = 1;
    Elem<T> * Del = Head;

    while(i <= n)
    {
        //Silinəcək elementə qədər gəlirik
        Del = Del->next;
        i++;
    }

    //Silinəcək elementdən əvvəlki elementə qədər
    //gəlirik
    Elem<T> * PrevDel = Del->prev;
    //Silinəcək elementdən sonrakı elementə qədər
    //gəlirik
    Elem<T> * AfterDel = Del->next;

    if(PrevDel != 0 && Count != 1)
        PrevDel->next = AfterDel;

    if(AfterDel != 0 && Count != 1)
        AfterDel->prev = PrevDel;

    if(n == 1)
        Head = AfterDel;
    if(n == Count)
        Tail = PrevDel;
    delete Del;

    Count--;
}

template <typename T>
void List<T>::Del(int n)
{
    if(n < 1 || n > Count)
    {
        cout << "Incorrect position !!!\n";
        return;
}
```



```
    int i = 1;
    Elem<T> * Del = Head;

    while(i < n)
    {
       //Silinəcək elementə qədər gəlirik
       Del = Del->next;
       i++;
    }


    //Silinəcək elementdən əvvəlki elementə qədər
    //gəlirik
    Elem<T> * PrevDel = Del->prev;
    //Silinəcək elementdən sonrakı elementə qədər
    //gəlirik
    Elem<T> * AfterDel = Del->next;

    if(PrevDel != 0 && Count != 1)
        PrevDel->next = AfterDel;

    if(AfterDel != 0 && Count != 1)
        AfterDel->prev = PrevDel;

    if(n == 1)
        Head = AfterDel;
    if(n == Count)
        Tail = PrevDel;

    delete Del;

    Count--;
}

template <typename T>
void List<T>::Print(int pos)
{
```

```
    //1-dən Count-a qədər mövqedirsə?
    if(pos < 1 || pos > Count)
    {
        //Düzgün olmayan mövqe
        cout << "Incorrect position !!!\n";
        return;
    }

    Elem<T> * temp;


    //Hansı tərəfdən daha tez hərəkət
    //edilməsini təyin edirik
    if(pos <= Count / 2)
    {
        //Başlıqdan başlayaraq hesablamaq
        temp = Head;
        int i = 1;

        while(i < pos)
        {
            //Lazım olan elementə qədər getmək
            temp = temp->next;
            i++;
        }
    }
    else
    {
        //Quyruqdan başlayaraq hesablamaq
        temp = Tail;
        int i = 1;

        while(i <= Count - pos)
        {
            //Lazım olan elementə qədər getmək
            temp = temp->prev;
            i++;
        }
    }
```



```
    //Elementin çapı
    cout << pos << " element: ";
    cout << temp->data << "\n";
}

template <typename T>
void List<T>::Print()
{
    if(Count != 0)
    {
        Elem<T> * temp = Head;
        while(temp != 0)
        {
            cout << temp->data << "\n";
            temp = temp->next;
        }
    }
}

template <typename T>
void List<T>::DelAll()
{
    while(Count != 0)
        Del(1);
}

template <typename T>
int List<T>::GetCount()
{
    return Count;
}

template <typename T>
List<T>& List<T>::operator = (const List<T> & L)
{
    if(this == &L)
        return *this;
```

```
    this->~List();
    Elem<T> * temp = L.Head;
    while(temp != 0)
    {
       AddTail(temp->data);
       temp = temp->next;
    }

    return *this;
}

template <typename T>
List<T> List<T>::operator + (const List<T>& L)
{
    List Result(*this);

    Elem<T> * temp = L.Head;

    while(temp != 0)
    {
       Result.AddTail(temp->data);
       temp = temp->next;
    }

    return Result;
}

template <typename T>
bool List<T>::operator == (const List<T>& L)
{
    if(Count != L.Count)
       return false;

    Elem<T> *t1, *t2;

    t1 = Head;
    t2 = L.Head;
```



```
    while(t1 != 0)
    {
        if(t1->data != t2->data)
            return false;

        t1 = t1->next;
        t2 = t2->next;
    }

    return true;
}

template <typename T>
bool List<T>::operator != (const List& L)
{
    if(Count != L.Count)
        return true;
    Elem<T> *t1, *t2;
    t1 = Head;
    t2 = L.Head;

    while(t1 != 0)
    {
        if(t1->data != t2->data)
            return true;

        t1 = t1->next;
        t2 = t2->next;
    }

    return false;
}

template <typename T>
bool List<T>::operator >= (const List& L)
{
    if(Count > L.Count)
        return true;
```

```
    if(*this == L)
        return true;

    return false;
}

template <typename T>
bool List<T>::operator <= (const List& L)
{
    if(Count < L.Count)
        return true;

    if(*this == L)
        return true;

    return false;
}

template <typename T>
bool List<T>::operator > (const List& L)
{
    if(Count > L.Count)
        return true;

    return false;
}

template <typename T>
bool List<T>::operator < (const List& L)
{
    if(Count < L.Count)
        return true;

    return false;
}
```



```
template <typename T>
List<T> List<T>::operator - ()
{
    List Result;
    Elem<T> * temp = Head;

    while(temp != 0)
    {
        Result.AddHead(temp->data);
        temp = temp->next;
    }

    return Result;
}

//Test nümunəsi
void main()
{
    List <int> L;

    const int n = 10;
    int a[n] = {0,1,2,3,4,5,6,7,8,9};

    //Cüt indekslərdə, başlıqda, tək indekslərdə -
    //quyruqda olan elementlərə əlavə edirik
     for(int i = 0; i < n; i++)
        if(i % 2 == 0)
            L.AddHead(a[i]);
        else
            L.AddTail(a[i]);

    //Siyahının çapı
    cout << "List L:\n";
    L.Print();

    cout << "\n\n";
```

```
    //Siyahının çapı
    cout << "List L:\n";
    L.Print();

    //2-ci və 8-ci elementin çapı
    L.Print(2);
    L.Print(8);

    List <int> T;

    //Siyahını köçürürük
    T = L;
    //Nüsxənin çapı
    cout << "List T:\n";
    T.Print();

    //İki siyahını cəmləyirik
    //(birinci çevrilmiş vəziyyətdə)
    cout << "List Sum:\n";
    List <int> Sum = -L + T;
    //Siyahının çapı
    Sum.Print();
}
```


Ev tapşırığı

Mövcud təkəlaqəli siyahı sinfinə növbəti əməliyyatları daxil etməli: elementin verilmiş mövqeyə əlavə edilməsi, verilmiş mövqedən elementin silinmsəi, verilmiş elementin axtarılması (müvvəffəq olduğu zaman funksiya tapılmış elementin mövqeyini qaytarır).

1. İkiəlaqəli siyahı əsasında “Növbə” şablon sinfini reallaşdırın.
2. Verilmiş tipdə obyektlər saxlayan massiv olan Array şablon sinif-konteyneri yaradın. Sinif ikiəlaqəli siyahı vasitəsi ilə reallaşdırılmalıdır. Sinif növbəti funksiyaları reallaşdırmalıdır:
   A) **GetSize** — massivin ölçüsünü əldə etməli (yaddaşın ayrıldığı elementlərin sayı);
   B) **SetSize(int size, int grow = 1)** — massivin ölçüsünün verilməsi (əgər size parametri əvvəlki ölçüdən böyükdürsə, onda əlavə yaddaş bloku ayırmaq lazımdır, əks halda artıq elementlər itirilir və yaddaş azad edilir); grow parametri əgər massivin elementlərinin sayı mövcud ölçünü aşırsa, nə qədər sayda element üçün yaddaşın ayrılacağını təyin edir. Məsələn, SetSize(5, 5); onu göstərir ki, 6-ci elementi əlavə edərkən massivin ölçüsü 10 olur, 11-ci elementi əlavə edərkən isə 15 olur və s.;

C) **GetUpperBound** — massivdə axırıncı mümkün indeksin əldə edilməsi. Məsələn, massivin ölçüsü 10 olarsa, siz ona 4 element əlavə edərsinizsə, funksiya 3 qaytaracaq.
D) **IsEmpty** — massiv boşdurmu?
E) **FreeExtra** — artıq yaddaşı (sonuncu mümkün indeksdən yuxarı olan) silmək;
F) **RemoveAll** — hamısını silmək;
G) **GetAt** — müəyyən elementi (indeksinə görə) əldə etmək;
H) **SetAt** — müəyyən element üçün yeni qiymətin verilməsi (elementin indeksi massivin cari ölçüsündən kiçik olmalıdır);
I) **operator []** — əvvəlki iki funksiyanı reallaşdırmaq üçün;
J) **Add** — massivə element əlavə etmək (ehtiyac olduğu zaman massiv SetSize funksiyasının grow qiyməti qədər artırılır);
K) **Append** — iki massivin "toplanması";
L) **operator =;**
M) **GetData** — verilənlər massivinin ünvanının əldə edilməsi;
N) **InsertAt** — verilmiş mövqeyə element(lər) yerləşdirməsi;
O) **RemoveAt** — verilmiş mövqedən element(lər)in silinməsi.