

Технология ООП

Санкт-Петербургский государственный политехнический университет

11 октября 2011

Достоинства списка:

- “быстрые” операции вставки/удаления;
- не требуют перераспределения памяти.

Недостатки списка:

- только последовательный доступ;
- время доступа к разным элементам разное;
- дополнительные затраты памяти на вспомогательный класс с данными.

Пример шаблона двухсвязного списка

```
template<typename T> class Node {
    Node<T>* pNext, *pPrev;
    T data;
    template<typename T> friend class List;
};

template<typename T> class List {
    Node<T> Head;
    Node<T> Tail;
    size_t m_size;
};
```

Реализация класса итератора для списка

```
class iterator {  
    Node* p;  
    public:  
    iterator() {  
    iterator(Node* pT) {  
    T& operator*() {  
    iterator& operator++(){  
    iterator operator++(int){  
    bool operator==(iterator& r) {  
    bool operator!=(iterator& r) {  
};
```

Пример использования итератора

```
MyList<int> l;  
l.push_back(1);  
l.push_front(2);  
l.push_back(3);  
l.push_front(4);  
MyList<int>::iterator it = l.begin();  
for(; it != l.end(); ++it) {  
    cout << *it << " ";  
}
```

Методы списка для получения итераторов

```
iterator begin() {  
iterator end()  {  
}
```

Шаблонный конструктор списка

```
template<typename IT> MyList(IT first , IT last)
{
    ....
}
```

Шаблонный конструктор списка

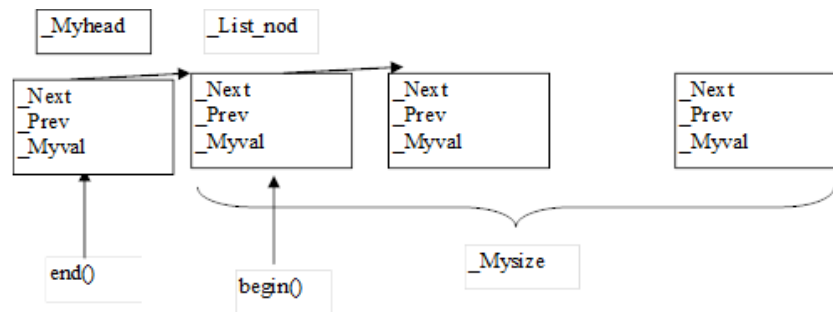
```
template<typename IT> MyList(IT first , IT last)
{
    Head.pNext = &Tail;
    Tail.pPrev = &Head;
    m_size=0;
    while(first != last)
    {
        push_back(*first);
        ++first;
    }
}
```

Пример создания списка по любой другой последовательности

```
int ar[10] = {5, 2, 3, 7};  
MyList<int> l1 (...); //  
vector<int> v(10, 1);  
MyList<int> l2 (...); //
```

```
template<class _Ty,  
class _Ax = allocator<_Ty> >  
class list : public _List_val<_Ty, _Ax>  
{// bidirectional linked list  
  
};
```

Внутреннее устройство list



```
list( );  
explicit list( size_type _Count );  
list( size_type _Count, const Type& _Val );  
list( const list& _Right );  
template<class InputIterator> list( InputIterator  
    _First, InputIterator _Last );  
list( list&& _Right );
```

Нет произвольного доступа. Поэтому не реализованы:

- `operator[]`
- `at()`

Не требуется резервирование. Поэтому нет:

- `capacity()`
- `reserve()`

Операция splice для list

Перемещение элементов из одного списка в другой за счет изменения указателей:

```
void splice(iterator it, list& x);  
void splice(iterator it, list& x, iterator first);  
void splice(iterator it, list& x, iterator first,  
            iterator last);
```

Пример использования splice

```
list<int> l; //1,2,3,4
list<int> l1; //11,22,33,44
//1
l.splice(l.begin(),l1); //11 22 33 44 1 2 3 4
//2
l.splice(l.begin(),l1,l1.begin()); //11 1 2 3 4
//3
list<int>::iterator it1=l1.begin();
++it1;
++it1;
l.splice(l.begin(),l1,it1,l1.end()); //33 44 1 2 3
4
```

Операция sort для list

Сортировка реализована только методом класса.

```
void sort(); //по возрастанию (operator<)  
template<class Pred> void sort(Pred pr); //!pr(*Pj,  
    *Pi)
```

```
list<int> l; //6, -2, 3, -4, 1  
l.sort(); //
```

//Как отсортировать список по модулю?

Сортировка списка по модулю

Сортировка реализована только методом класса.

```
bool Mod(int left , int right)
{
    return abs(left)<abs(right);
}
int main()
{
    list<int> l;//6, -2, 3, -4, 1
    l.sort(Mod);
}
```

Операция merge для list

Объединение **отсортированных** списков:

```
void merge( list<Type, Allocator>& _Right );  
template<class Traits> void merge( list<Type,  
    Allocator>& _Right, Traits _Comp );
```

```
merge() list<int> l; //1,2,3,55  
list<int> l1; //11,22,33,44  
l1.merge(l); //  
//Что произошло с l?
```

Операции для работы с началом последовательности

```
void push_front(const T& x);  
void pop_front();
```

Операция remove для list

```
void remove( const Type& _Val );  
template<class Predicate> void remove_if(Predicate  
    _Pred )
```

```
list <int> l; //3, 5, -2, 5  
l.remove(5); // ???
```

```
//Удалить все отрицательные
```

Использование `remove_if` с предикатом

```
bool Neg(int x)
{
    return x < 0;
}
int main()
{
    list <int> l; //3,5,-2,5
    l.remove_if(Neg);
}
```

Оператор unique для list

```
void unique( );  
template<class BinaryPredicate> void unique(  
    BinaryPredicate _Pred );
```
