list

БАЗОВЫЕ КОНТЕЙНЕРЫ

Сравнение списка и вектора

Достоинства вектора:

- ???

Недостатки вектора:

- ???

Достоинства списка:

- ???

Недостатки списка:

- ???

Сравнение списка и вектора

Достоинства списка:

- Операции вставки/удаления
- Не требуют перераспределения памяти
- Итератор остается действительным при вставке/удалении

Недостатки списка:

- Только последовательный доступ
- Время доступа к разным элементам разное
- Дополнительные затраты памяти на «обертку» для каждого данного
- Кэширование практически исключается

Шаблон двухсвязного списка

```
template<typename T> class Node{
        Node* 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;
        …
};
```

push_front(), pop_front()

???

Итератор для списка

```
public:
class iterator{
//Данные
        ???
public:
//Методы
        ???
};
```

Итератор для списка

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

Методы списка для получения итераторов

iterator begin()

{???}

iterator end()

{???}

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<<" ";
}
```

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

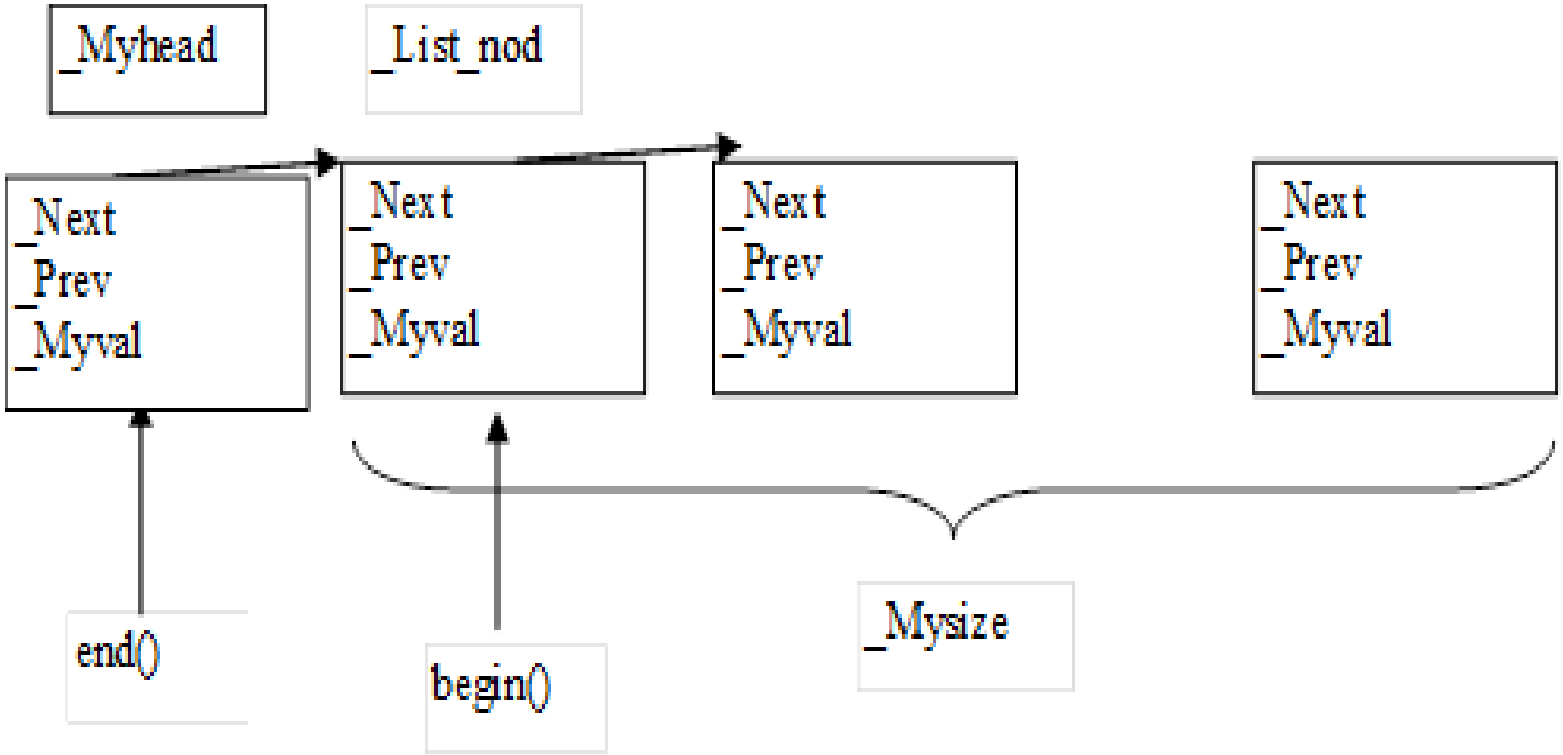
```
int ar[10] = {5,2,3,…};
MyList<int> l1(??? );

MyVector<int> v(10,1);
MyList<int> l2(??? );
```

STL

list

Полубенцева М.И.

Внутреннее устройство list



Header <list>
namespace std

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

Header <list>
namespace std

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

Конструкторы

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

Нет произвольного доступа

Поэтому **не** реализованы:

- operator[]
- at

Не требуется резервирование

Поэтому **нет**:

- capacity()
- reserve()

Специфические для list операции

Перемещение элементов из одного списка в другой без копирования node-ов (**только 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);

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
```

Специфические для list операции

сортировка реализована только методом класса!

- void **sort**(); //по возрастанию (operator<)

- template<class Pred> void **sort**(Pred pr);
  //!pr(*Pj, *Pi)

sort()

1.

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

2. Как отсортировать по модулю?

sort()

```
bool Mod(int left, int right)
{ return abs(left)<abs(right);}

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

Специфические для list операции

объединение отсортированных списков:

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

merge()

```
list<int> L1; //1,2,3,55
list<int> L2; //11,22,33,44
L1.merge(L2); // L1 ???
//L2 ???
```

Операции с началом последовательности

В дополнение к push_back(), pop_back():

- void **push_front**(const T& x);
- void **pop_front**();

Специфические для list операции

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

remove(), remove_if()

1.

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

2. Удалить все отрицательные

remove_if()

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

}
```

Специфические для list операции

- void **unique**( );
  abbbcabbb -> abcab
- template<class BinaryPredicate> void **unique**(BinaryPredicate _Pred);

C++11 Односвязный список

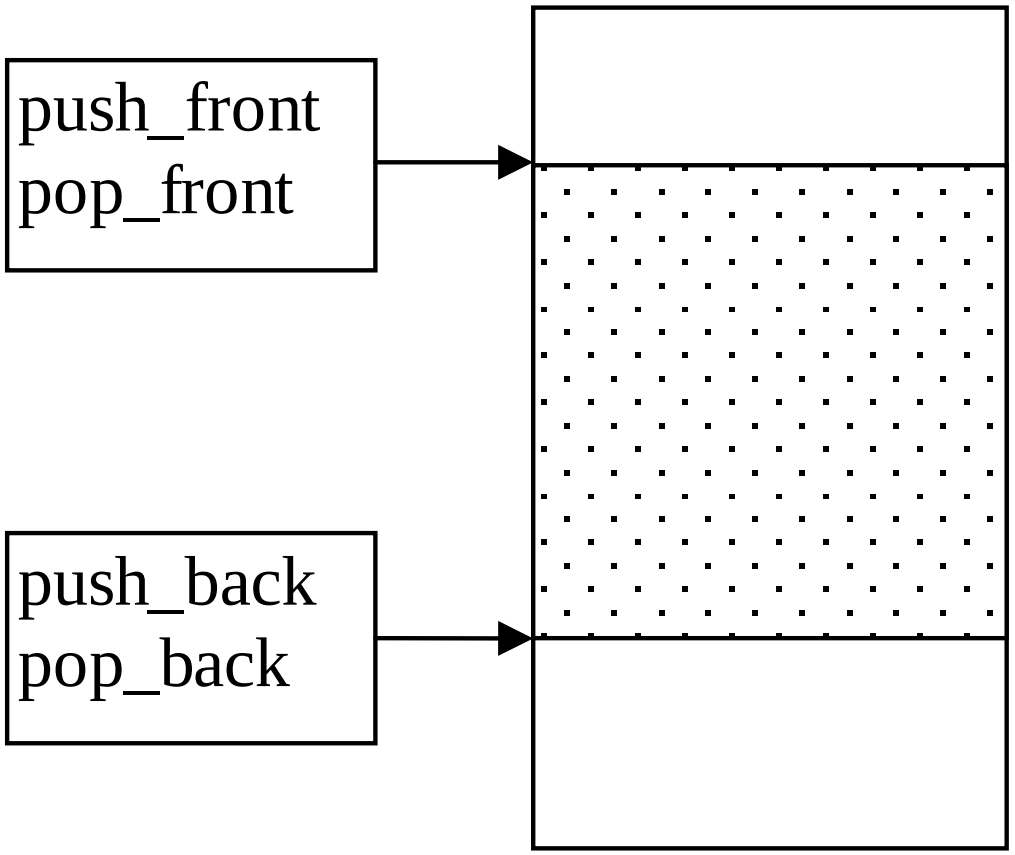
```
#include <forward_list>

namespace std {
template <typename T,
typename Allocator = allocator<T> >
class forward_list;
}
```

По сравнению с двухсвязным списком – занимает меньше памяти

STL

deque

Назначение deque:



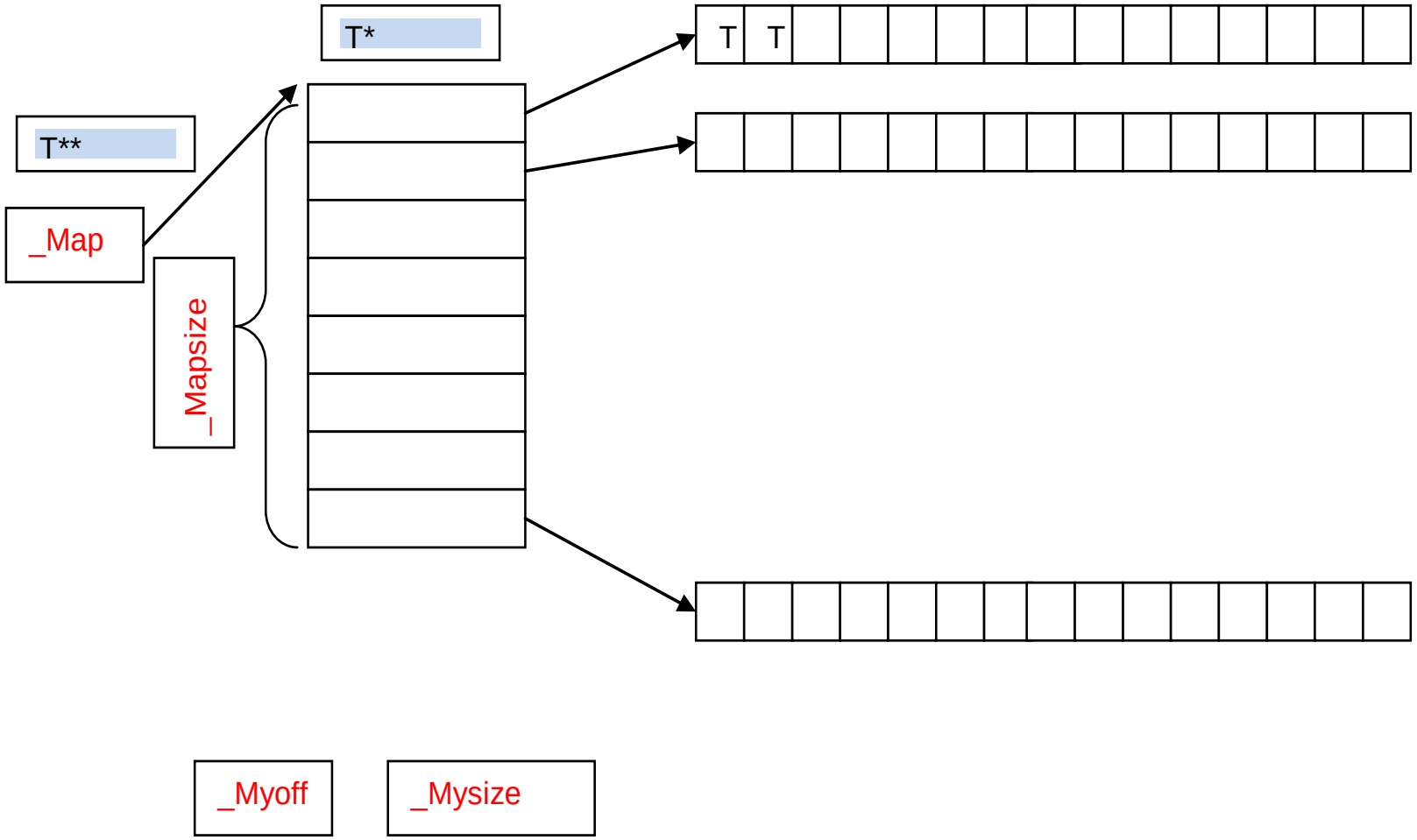
Требования к deque:

- Обеспечивать **произвольный** доступ к элементам (как у vector => at(), operator[]…)
- Эффективная работа с **началом последовательности** (как у list => push_front(), pop_front()…)

Header <deque>
namespace std

```
template<class _Ty,
                    class _Ax = allocator<_Ty> >
class deque
                : public _Deque_val<_Ty, _Ax>
{// circular queue of pointers to blocks
```

Внутреннее устройство deque



Пример использования deque

```
deque<char> d;//_Map=0, _Mapsize=0, _Myoff=0, _Mysize=0

d.push_front('A');//_Map, _Mapsize=8, _Myoff=127, _Mysize=1
d.push_front('B');//_Map, _Mapsize=8, _Myoff=126, _Mysize=2
d.push_front('C');//_Map, _Mapsize=8, _Myoff=125, _Mysize=3

d.push_back('a');// _Myoff=125, _Mysize=4
d.push_back('b');// _Mysize=5
d.push_back('c');// _Mysize=6

deque<char>::iterator it=d.begin();
while(it!=d.end()) {cout<<*it<<" "; ++it;}
```

Итератор

```
class iterator{
        deque* _MyCont;
        size_type _Off;
public:
iterator& operator++(){++_Off; return *this;}
T& operator*(){
        size_type n = _Off/16;
        n = n% (_MyCont->_Mapsize);
        size_type m = _Off%16;
        return _MyCont->_Map[n][m];
}
};
```

Класс deque

```
iterator begin(){ return iterator(_MyOff, this);}

iterator end()
{ return iterator(_MyOff + _Mysize, this);}

T& operator[](int i){ return *(begin()+i);}
```