МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Объектно ориентированное программирование»
Тема: Контейнеры

Студент гр. 7304 _____ Давыдов А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2019

Задача

1) Вектор
    - Необходимо реализовать конструкторы и деструктор для контейнера вектор. Поведение реализованных функций должно быть таким же, как у класса std::vector
    - Необходимо реализовать операторы присваивания и функцию assign для контейнера вектор. Поведение реализованных функций должно быть таким же, как у класса std::vector.
    - Необходимо реализовать функции resize и erase для контейнера вектор. Поведение реализованных функций должно быть таким же, как у класса std::vector
    - Необходимо реализовать функции insert и push_back для контейнера вектор. Поведение реализованных функций должно быть таким же, как у класса std::vector
2) Список
    - Необходимо реализовать список со следующими функциями:
        1. вставка элементов в голову и в хвост,
        2. получение элемента из головы и из хвоста,
        3. удаление из головы, хвоста и очистка
        4. проверка размера.
    - Необходимо добавить к сделанной на прошлом шаге реализации списка следующие функции:
        1. деструктор
        2. конструктор копирования,
        3. конструктор перемещения,
        4. оператор присваивания.
    - Реализовать итератор для списка. Для краткости реализации можно ограничиться однонаправленным изменяемым (неконстантным) итератором. Необходимо реализовать операторы: =, ==, !=, ++ (постфиксный и префиксный), *, ->.
    - На данном шаге с использованием итераторов необходимо реализовать:

1. вставку элементов (Вставляет value перед элементом, на который указывает pos. Возвращает итератор, указывающий на вставленный value),
2. удаление элементов (Удаляет элемент в позиции pos. Возвращает итератор, следующий за последним удаленным элементом).

Экспериментальные результаты.

1) Были реализованы конструкторы и деструктор класса вектор

```
explicit vector(size_t count = 0) : size_(count)
{
if(size_ == 0)
pointer = nullptr;
else
pointer = new Type[count];
m_first = pointer;
m_last = pointer + count;
}

template <typename InputIterator>
vector(InputIterator first, InputIterator last) : size_(last - first)
{
if(size_ == 0)
pointer = nullptr;
else
pointer = new Type[size_];
for(InputIterator it = first ; it!= last; ++it)
{
pointer[it - first] = *it;
}
m_first = pointer;
m_last = pointer + size_;
}

vector(std::initializer_list<Type> init) : size_(init.size())
{
if(size_ == 0)
pointer = nullptr;
else
{
pointer = new Type[size_];
int i = 0;
for(auto &el : init)
{
pointer[i] = el;
++i;
}
}
m_first = pointer;
m_last = pointer + size_;
}

//copy constructor
vector(const vector& other) : size_(other.size_)
{
if(size_ == 0)
pointer = nullptr;
```

```
    else
    {
    pointer = new Type[size_];
    for(int i = 0; i < size_; ++i)
    pointer[i] = other.pointer[i];
    }
    m_first = pointer;
    m_last = pointer + size_;
    }

    vector(vector&&   other)   :   size_(other.size_),   m_first(other.m_first),
    m_last(other.m_last), pointer(other.pointer)
    {
    other.pointer = nullptr;
    other.m_last = nullptr;
    other.m_first = nullptr;
    }

    ~vector()
    {
    delete[] pointer;
    }
```

2) Были реализованы опреаторы присваивания, функция assign для класса вектор

```
//assignment operators
vector& operator=(const vector& other)
{
if(this!= &other)
{
size_ = other.size_;
delete[] pointer;
if(size_ == 0)
pointer = nullptr;
else
{
pointer = new Type[size_];
for(int i = 0; i < size_; ++i)
pointer[i] = other.pointer[i];
}
}
m_first = pointer;
m_last = pointer + size_;

return *this;
}

vector& operator=(vector&& other)
{
if(this!= &other)
{
size_ = other.size_;
delete[] pointer;
pointer = other.pointer;
m_first = other.m_first;
m_last = other.m_last;
other.m_last = nullptr;
other.m_first = nullptr;
other.pointer = nullptr;
}

return *this;
}
```

```
// assign method
template <typename InputIterator>
void assign(InputIterator first, InputIterator last)
{
delete[] pointer;
size_ = last - first;
pointer = new Type[size_];
for(InputIterator it = first; it!= last; ++it)
pointer[it - first] = *it;
m_first = pointer;
m_last = pointer + size_;
      }
```

3) Были реализованы функции resize и erase для класса вектор

```
void resize(size_t count)
{
if(size_ == count)
return;
else if(size_ > count)
{
Type *buff = new Type[count];

for(int i = 0; i < count; ++i)
buff[i] = pointer[i];

size_ = count;
delete[] pointer;
pointer = buff;
}
else if(size_ < count)
{
Type *buff = new Type[count];

for(int i = 0; i < size_; ++i)
buff[i] = pointer[i];
size_ = count;
delete[] pointer;
pointer = buff;
}
m_first = pointer;
m_last = pointer + size_;
}

//erase methods
iterator erase(const_iterator pos)
{
int dif = pos - m_first;
if(pos == m_last)
return m_last;
Type * buff = new Type[size_ - 1];
int i = 0;
for(const_iterator it = m_first; it!= m_last; ++it)
if(it == pos)
;//pass
else
{
buff[i] = *it;
++i;
}

delete[] pointer;
```

```
    --size_;
    pointer = buff;
    m_first = pointer;
    m_last = pointer + size_;
    return pointer + dif;
    }

    iterator erase(const_iterator first, const_iterator last)
    {
    int dif = first - m_first;
    Type * buff = new Type[size_ - (last - first)];
    int i = 0;
    for(const_iterator it = m_first; it!= m_last; ++it)
    if(it >= first && it < last)
    ;//pass
    else
    {
    buff[i] = *it;
    ++i;
    }

    delete[] pointer;
    size_ -= (last - first);
    pointer = buff;
    m_first = pointer;
    m_last = pointer + size_;
    return pointer + dif;
    }
```

4) Были реализованы функции insert и push_back для класса вектор

```
iterator insert(const_iterator pos, const Type& value)
{
int dif = pos - m_first;
Type *buff = new Type[size_ + 1];

for(iterator it = m_first; it!= pos; ++it)
buff[it - m_first] = pointer[it - m_first];
buff[pos - m_first] = value;

for(const_iterator it = pos; it!= m_last; ++it)
buff[it - m_first + 1] = pointer[it - m_first];

size_ += 1;
delete[] pointer;
pointer = buff;
m_first = pointer;
m_last = pointer + size_;
return pointer + dif;
}

template <typename InputIterator>
iterator insert(const_iterator pos, InputIterator first, InputIterator last)
{
int dif = pos - m_first;
Type *buff = new Type[size_ + (last - first)];

for(iterator it = m_first; it!= pos; ++it)
buff[it - m_first] = pointer[it - m_first];

const_iterator it = pos;
for(InputIterator it1 = first; it1!= last; ++it, ++it1)
```

```
buff[it - m_first] = *it1;

//use it after work of cycle above
for(const_iterator it1 = pos; it1!= m_last; ++it, ++it1)
buff[it - m_first] = pointer[it1 - m_first];

size_ += (last - first);
delete[] pointer;
pointer = buff;
m_first = pointer;
m_last = pointer + size_;
return pointer + dif;
}

//push_back methods
void push_back(const value_type& value)
{
resize(size_ + 1);
pointer[size_ - 1] = value;
}
```

5) Были реализован список с возможностью вставки элементов в голову и хвост, получения элемента из головы, удаления из головы, хвоста и очистки, проверки размера.

```
void push_back(const value_type& value)
{
if(m_head == nullptr)
{
m_head = new node<Type>(value, nullptr, nullptr);
m_head->prev = nullptr;
m_head->next = nullptr;
m_tail = m_head;
}
else if(m_tail == m_head)
{
m_tail = new node<Type>(value, nullptr, m_head);
m_head->next = m_tail;
m_tail->prev = m_head;
}
else
{
m_tail->next = new node<Type>(value, nullptr, m_tail);
m_tail->next->prev = m_tail;
m_tail = m_tail->next;
}
}

void push_front(const value_type& value)
{
if(m_head == nullptr)
{
m_head = new node<Type>(value, nullptr, nullptr);
m_head->prev = nullptr;
m_head->next = nullptr;
m_tail = m_head;
}
else
{
node<Type> *old_head = m_head;
m_head = new node<Type>(value, old_head, nullptr);
```

```
m_head->next = old_head;
old_head->prev = m_head;
m_head->prev = nullptr;
}
}

reference front()
{
return m_head->value;
}

const_reference front() const
{
return m_head->value;
}

reference back()
{
return m_tail->value;
}

const_reference back() const
{
m_tail->value;
}

void pop_front()
{
if(m_head == nullptr)
return;
else if(m_tail == m_head)
{
delete m_head;
m_head = nullptr;
m_tail = nullptr;
}
else
{
node<Type> *next_head = m_head->next;
delete m_head;
m_head = next_head;
m_head->prev = nullptr;
}
}

void pop_back()
{
if(m_head == nullptr)
return;
else if(m_tail == m_head)
{
delete m_head;
m_head = nullptr;
m_tail = nullptr;
}
else
{
node<Type> *last = m_tail;
m_tail = m_tail->prev;
delete last;
m_tail->next = nullptr;
}
}
```

```
void clear()
{
if(m_head == nullptr)
return;
else
{
node<Type> *old_tail = m_tail;

while(m_tail!= nullptr)
{
m_tail = m_tail->prev;
delete old_tail;
old_tail = m_tail;
}
m_head = nullptr;
}
}

bool empty() const
{
if(m_head == nullptr && m_tail == nullptr)
return true;
else
return false;
}

size_t size() const
{
int size_ = 0;
node<Type> *cur = m_tail;

while(cur!= nullptr)
{
cur = cur->prev;
++size_;
}
return size_;
      }
```

6) Были реализованы деструктор, конструктор копирования, конструктор перемещения и оператор присваивания класса список

```
~list()
{
node<Type> *cur = m_tail;
while(m_tail!= nullptr)
{
m_tail = m_tail->prev;
delete cur;
cur = m_tail;
}
m_head = nullptr;
}

//copy constructor
list(const list& other)
{
if(other.m_head == nullptr)
m_head = m_tail = nullptr;
else
{
m_head = new node<Type>(other.m_head->value, nullptr, nullptr);
```

```
m_tail = m_head;
if(other.m_head->next == nullptr)
return;
else
{
node<Type> *cur = other.m_head->next;
while(cur!= nullptr)
{
m_tail->next = new node<Type>(cur->value, nullptr, m_tail);
m_tail = m_tail->next;
cur = cur->next;
}
}
}
}

list(list&& other) : m_head(other.m_head), m_tail(other.m_tail)
{
other.m_head = nullptr;
other.m_tail = nullptr;
}

list& operator= (const list& other)
{
if(this!= &other)
{
clear();
if(other.m_head == nullptr)
m_head = m_tail = nullptr;
else
{
m_head = new node<Type>(other.m_head->value, nullptr, nullptr);
m_tail = m_head;
if(other.m_head->next == nullptr)
; //pass
else
{
node<Type> *cur = other.m_head->next;
while(cur!= nullptr)
{
m_tail->next = new node<Type>(cur->value, nullptr, m_tail);
m_tail = m_tail->next;
cur = cur->next;
}
}
}
}

return *this;
}
```

7) Был реализован класс итератор для списка

```
template <class Type>
class list_iterator
{
public:
typedef ptrdiff_t difference_type;
typedef Type value_type;
typedef Type* pointer;
typedef Type& reference;
typedef size_t size_type;
```

```
typedef std::forward_iterator_tag iterator_category;

list_iterator()
: m_node(NULL)
{}

list_iterator(const list_iterator& other)
: m_node(other.m_node)
{}

list_iterator& operator = (const list_iterator& other)
{
if(this!= &other)
m_node = other.m_node;

return *this;
}

bool operator == (const list_iterator& other) const
{
if(m_node == other.m_node)
return true;
else
return false;
}

bool operator != (const list_iterator& other) const
{
if(!(*this == other))
return true;
else
return false;
}

reference operator * ()
{
return m_node->value;
}

pointer operator -> ()
{
return &m_node->value;
}

list_iterator& operator ++ ()
{
m_node = m_node->next;
return *this;
}

list_iterator operator ++ (int)
{
m_node = m_node->next;
return *this;
}

private:
friend class list<Type>;

list_iterator(node<Type>* p)
: m_node(p)
{
}
```

```
node<Type>* m_node;
      };
```

8) Были реализованы функции insert и erase для класса список

```
iterator insert(iterator pos, const Type& value)
{
if(pos.m_node == m_head)
{
push_front(value);
pos.m_node = pos.m_node->prev;
return pos;
}
else if(pos.m_node == nullptr)
{
push_back(value);
pos.m_node = m_tail;
return pos;
}
else
{
node<Type> *new_node = new node<Type>(value, pos.m_node, pos.m_node->prev);
pos.m_node->prev->next = new_node;
pos.m_node->prev = new_node;
pos.m_node = pos.m_node->prev;
return pos;
}
}

iterator erase(iterator pos)
{
if(pos.m_node == m_head)
{
pop_front();
pos.m_node = m_head;
return pos;
}
else if(pos.m_node == nullptr)
{
return pos;
}
else if(pos.m_node->next == nullptr)
{
pop_back();
pos.m_node = m_tail;
return pos;
}
else
{
pos.m_node->prev->next = pos.m_node->next;
pos.m_node->next->prev = pos.m_node->prev;
delete pos.m_node;
pos.m_node = pos.m_node->next;
return pos;
}
}
```

Выводы.

Были реализованы упрощенные контейнеры вектор и список согласно заданию. Контейнеры имеют основные методы вставки элементов в начало и конец, удаление и вставка в произвольную позицию, поддерживают работу с итераторами.