Лабораторная работа № 11

Цель работы

1. Создание консольного приложения, состоящего из нескольких файлов в системе программирования Visual Studio.
2. Использование последовательных контейнеров библиотеки STL в ОО программе.

Постановка задачи

Задача 1.

1. Создать последовательный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте).
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

- Контейнер - двунапрвленная очередь
- Тип элементов - int

Задача 2.

1. Создать последовательный контейнер.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

- Тип элементов Money (см. лабораторную работу №3)

Задача 3

1. Создать параметризированный класс, используя в качестве контейнера последовательный контейнер.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

- Параметризированный класс – Вектор (см. лабораторную работу №7)

Задача 4

1. Создать адаптер контейнера.
2. Заполнить его элементами пользовательского типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде глобальных функций.

- Адаптер контейнера - стек.

Задача 5

1. Создать параметризированный класс, используя в качестве контейнера адаптер контейнера.
2. Заполнить его элементами.
3. Добавить элементы в соответствии с заданием
4. Удалить элементы в соответствии с заданием.
5. Выполнить задание варианта для полученного контейнера.
6. Выполнение всех заданий оформить в виде методов параметризированного класса.

- Параметризированный класс – Вектор
- Адаптер контейнера - стек.

Задание 3 Найти максимальный элемент и добавить его в конец контейнера

Задание 4 Найти элемент с заданным ключом и удалить его из контейнера

Задание 5 К каждому элементу добавить среднее арифметическое элементов контейнера

Описание класса

Для выполнения работы были использованы следующие классы и структуры данных:

- Класс Money (из предыдущих лабораторных работ):
  - Хранит денежную сумму в рублях и копейках
  - Имеет перегруженные операторы ввода/вывода
  - Реализует основные арифметические операции
- Шаблонный класс Vector:
  - Реализует динамический массив
  - Поддерживает основные операции работы с последовательным контейнером
  - Имеет методы для добавления/удаления элементов
- Адаптеры контейнеров:
  - stack - реализация стека (LIFO)
  - queue - реализация очереди (FIFO)
  - deque - двусторонняя очередь

Определение компонентных функций

Для класса Money были реализованы следующие ключевые методы:

```
// Конструкторы
Money();
Money(long rub, int cheers);
Money(const Money& m);

// Операторы
Money& operator=(const Money& m);
Money operator+(const Money& m) const;
bool operator==(const Money& m) const;

// Ввод/вывод
friend istream& operator>>(istream& in, Money& m);
friend ostream& operator<<(ostream& out, const Money& m);
```

Для шаблонного класса Vector:

```
// Основные методы
void push_back(const T& value);
void pop_back();
void insert(iterator pos, const T& value);
void erase(iterator pos);
T& operator[](size_t index);
```

Определение глобальных функций

Были реализованы следующие глобальные функции для работы с контейнерами:

```
// Для deque<int>
void fillDeque(deque<int>& d);
void printDeque(const deque<int>& d);
void addMaxToEnd(deque<int>& d);
void removeByValue(deque<int>& d, int value);

// Для stack<Money>
void fillStack(stack<Money>& s);
void printStack(stack<Money> s); // Передается по значению для сохранения
исходного стека
void addAverageToStack(stack<Money>& s);
```

Функция main()

```
int main() {
    // Задача 1 - работа с deque<int>
    deque<int> dq;
    fillDeque(dq);
    addMaxToEnd(dq);
```

```
    removeByValue(dq, 5);

    // Задача 2 - работа с list<Money>
    list<Money> moneyList;
    fillList(moneyList);
    processList(moneyList);

    // Задача 3 - параметризированный Vector
    Vector<int> vec;
    // ... операции с вектором

    // Задача 4 - адаптер stack
    stack<Money> moneyStack;
    fillStack(moneyStack);
    addAverageToStack(moneyStack);

    return 0;
}
```

Объяснение результатов работы программы

Программа успешно демонстрирует:

1. Работу с последовательными контейнерами STL:
    - Заполнение, модификация и вывод содержимого
    - Использование стандартных алгоритмов (find, max_element и др.)
2. Особенности работы с пользовательским типом Money:
    - Корректное хранение в контейнерах
    - Правильную работу перегруженных операторов
3. Применение адаптеров контейнеров:
    - Ограниченный интерфейс stack
    - Особенности работы с элементами через top() и pop()
4. Реализацию шаблонного класса Vector:
    - Поддержку основных операций последовательного контейнера
    - Работу с разными типами данных

Ответы на контрольные вопросы

1. Из каких частей состоит библиотека STL?

Ответ:

- Контейнеров (vector, list, map и др.)
- Итераторов
- Алгоритмов (sort, find и др.)
- Функциональных объектов (функторы)
- Адаптеров (stack, queue)
- Аллокаторов (управление памятью)

2. Какие типы контейнеров существуют в STL?

Ответ:

- Последовательные (vector, list, deque)
- Ассоциативные (set, map, multiset, multimap)
- Неупорядоченные ассоциативные (C++11: unordered_set, unordered_map)
- Адаптеры (stack, queue, priority_queue)

3. Что нужно сделать для использования контейнера STL в своей программе?

Ответ:

1. Подключить соответствующий заголовочный файл (#include и т.д.)
2. Использовать пространство имен std (using namespace std;)
3. Объявить контейнер с нужным типом элементов (vector v;)

4. Что представляет собой итератор?

Ответ: Итератор - это объект, подобный указателю, который предоставляет доступ к элементам контейнера и позволяет перемещаться между ними.

5. Какие операции можно выполнять над итераторами?

Ответ:

- Инкремент (++it)
- Декремент (--it)
- Разыменование (*it)
- Сравнение (==, !=)
- Для random-access: арифметика (it + n), сравнение (<, >)

6. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?

Ответ:

```
for(vector<int>::iterator it = v.begin(); it != v.end(); ++it) {
    cout << *it << endl;
}
```

7. Какие типы итераторов существуют?

Ответ:

- Input (только чтение, однонаправленный)
- Output (только запись, однонаправленный)
- Forward (чтение/запись, однонаправленный)
- Bidirectional (чтение/запись, двунаправленный)
- Random access (чтение/запись, произвольный доступ)

8. Перечислить операции и методы общие для всех контейнеров.

Ответ:

- begin(), end()
- size(), empty()
- clear()
- swap()
- Операторы сравнения (==, !=, < и др.)

9. Какие операции являются эффективными для контейнера vector? Почему?

Ответ:

- Доступ по индеку (operator[])
- Вставка/удаление в конце (push_back, pop_back)

Причина: непрерывное хранение элементов в памяти

10. Какие операции являются эффективными для контейнера list? Почему?

Ответ:

- Вставка/удаление в любом месте (insert, erase)
- Слияние списков (merge, splice)

Причина: связная структура данных

11. Какие операции являются эффективными для контейнера deque? Почему?

Ответ:

- Вставка/удаление в начале и конце (push_front, pop_front, push_back, pop_back)
- Доступ по индексу

Причина: гибридная структура (блоки с указателями)

12. Перечислить методы, которые поддерживает последовательный контейнер vector.

Ответ:

- at(), operator[]
- push_back(), pop_back()
- reserve(), capacity()
- resize()
- insert(), erase()

13. Перечислить методы, которые поддерживает последовательный контейнер list.

Ответ:

- push_front(), pop_front()

- merge(), splice()
- sort(), unique()
- remove(), remove_if()

14. Перечислить методы, которые поддерживает последовательный контейнер deque.

Ответ:

- push_front(), pop_front()
- push_back(), pop_back()
- insert(), erase()
- resize()

15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?

Ответ:

```
v.erase(v.begin() + 1, v.begin() + 5);
```

16. Задан контейнер vector. Как удалить из него последний элемент?

Ответ:

```
v.pop_back();
```

17. Задан контейнер list. Как удалить из него элементы со 2 по 5?

Ответ:

```
auto first = l.begin(); advance(first, 1);
auto last = l.begin(); advance(last, 5);
l.erase(first, last);
```

18. Задан контейнер list. Как удалить из него последний элемент?

Ответ:

```
l.pop_back();
```

19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?

Ответ:

```
d.erase(d.begin() + 1, d.begin() + 5);
```

20. Задан контейнер deque. Как удалить из него последний элемент?

Ответ:

```
d.pop_back();
```

21. Написать функцию для печати последовательного контейнера с использованием итератора.

Ответ:

```
template<typename T>
void print_container(const T& container) {
    for(auto it = container.begin(); it != container.end(); ++it) {
        cout << *it << " ";
    }
    cout << endl;
}
```

22. Что представляют собой адаптеры контейнеров?

Ответ: Это обертки над базовыми контейнерами, предоставляющие ограниченный интерфейс (stack, queue, priority_queue)

23. Чем отличаются друг от друга объявления stack s и stack<int, list > s?

Ответ:

- stack s - использует deque по умолчанию
- stack<int, list> s - использует list в качестве базового контейнера

24. Перечислить методы, которые поддерживает контейнер stack.

Ответ:

- push(), pop()
- top()
- empty(), size()

25. Перечислить методы, которые поддерживает контейнер queue.

Ответ:

- push(), pop()

- front(), back()
- empty(), size()

26. Чем отличаются друг от друга контейнеры queue и priority_queue?

Ответ:

- queue - FIFO (первый вошел - первый вышел)
- priority_queue - элементы упорядочены по приоритету

27. Задан контейнер stack. Как удалить из него элемент с заданным номером?

Ответ: Нельзя напрямую, нужно:

1. Извлечь все элементы во временный stack до нужного
2. Удалить элемент
3. Вернуть остальные элементы обратно

28. Задан контейнер queue. Как удалить из него элемент с заданным номером?

Ответ: Аналогично stack - напрямую нельзя, только через временную очередь

29. Написать функцию для печати контейнера stack с использованием итератора.

Ответ:

```
void print_stack(stack<int> s) {
    while(!s.empty()) {
        cout << s.top() << " ";
        s.pop();
    }
}
```

30. Написать функцию для печати контейнера queue с использованием итератора.

Ответ:

```
void print_queue(queue<int> q) {
    while(!q.empty()) {
        cout << q.front() << " ";
        q.pop();
    }
}
```