ЛАБОРАТОРНАЯ РАБОТА №2
STL АЛГОРИТМЫ

1. Цель работы

Применение обобщенных алгоритмов вместе с контейнерами библиотеки STL.

2. Теоретические сведения

2.1. Основные концепции STL

STL – Standard Template Library, в дополнение к контейнерным классам включает набор обобщенных алгоритмов. **Обобщенные алгоритмы** реализуют большое количество процедур, применимых к контейнерам: поиск, сортировку, слияние и т. п. Однако они не являются методами контейнерных классов. Алгоритмы представлены в STL в форме глобальных шаблонных функций. Благодаря этому достигается универсальность: эти функции можно применять не только к объектам различных контейнерных классов, но также и к массивам. Независимость от типов контейнеров достигается за счет косвенной связи функции с контейнером: в функцию передается не сам контейнер, а пара адресов first, last, задающая диапазон обрабатываемых элементов.

2.2. Алгоритмы

Включение заголовка <algorithm> определяет набор функций специально разработанных для применения совместно с контейнерами. Диапазон любой коллекции объектов доступен с помощью итераторов либо указателей, например в массиве или в STL контейнере. Следует отметить, что алгоритмы работают с элементами через итераторы, ничего не зная о контейнере, его структуре и размере. Таким образом, алгоритм может работать с очень разными контейнерами, содержащими значения разнообразных типов. Алгоритмы, которые возвращают итератор, как правило, для сообщения о неудаче используют конец входной последовательности. Алгоритмы не выполняют проверки диапазона на их входе и выходе. Когда алгоритм возвращает итератор, это будет итератор того же типа, что и был на входе. Алгоритмы в STL реализуют большинство распространенных универсальных операций с контейнерами, такие как просмотр, сортировка, поиск, вставка и удаление элементов.

Подробнее о функциях: http://www.cplusplus.com/reference/algorithm/

Ниже приведены наиболее часто используемые функций-алгоритмов STL.

- **Операции которые не изменяют элементы контейнера.**

for_earch()	Вызывает заданную функцию для каждого элемента последовательности
find()	Находит первое вхождение значения в последовательность
find_if()	находит первое соответствие предикату в последовательности
count()	подсчитывает количество вхождений значения в последовательность
count_if()	подсчитывает количество выполнений предиката в последовательности
search()	находит первое вхождение последовательности как подпоследовательности
search_n()	находит n-е вхождение значения в последовательность

- **Операции меняют содержимое контейнера.**

copy()	копирует последовательность, начиная с первого элемента
swap()	меняет местами два элемента
replace()	заменяет элементы с указанным значением
replace_if()	заменяет элементы при выполнении предиката
replace_copy()	копирует последовательность, заменяя элементы с указанным значением
replace_copy_if()	копируетпоследовательность, заменяя элементы при выполнении предиката
fill()	заменяет все элементы данным значением
remove()	удаляет элементы с данным значением
remove_if()	удаляет элементы при выполнении предиката
remove_copy()	копирует последовательность, удаляя элементы с указанным значением
remove_copy_if()	копирует последовательность, удаляя элементы при выполнении предиката
reverse()	меняет порядок следования элементов на обратный
random_shuffle()	перемещает элементы согласно случайному равномерному распределению (“тасует” последовательность)
transform()	выполняет заданную операцию над каждым элементом последовательности
unique()	удаляет равные соседнии элементы
unique_copy()	копирует последовательность, удаляя равные соседние элементы

- **Сортировка.**

sort()	сортирует последовательность с хорошей средней эф-фективностью
partial_sort()	сортирует часть последовательности
stable_sort()	сортирует последовательность, сохраняя порядок следования равных элементов
lower_bound()	находит первое вхождение значения в отсортированной последовательности
upper_bound()	находит первый элемент, больший чем заданное значение
binary_search()	определяет, есть ли данный элемент в отсортированной последовательности
merge()	сливает две отсортированные последовательности

- **Работа с множествами.**

includes()	проверка на вхождение
set_union()	объединение множеств
set_intersection()	пересечение множеств
set_difference()	разность множеств

- **Минимумы и максимумы.**

min()	меньшее из двух
max()	большее из двух
min_element()	наименьшее значение в последовательности
max_element()	наибольшее значение в последовательности

- **Перестановки.**

next_permutation()	следующая перестановка в лексикографическом порядке
pred_permutation()	предыдущая перестановка в лексикографическом порядке

2.3. Функциональные объекты

Функциональные объекты (ФО или предикат) - это объекты, используют синтаксис такой же как обычный вызов функции. Это возможно при определении в классе метода **operator()**. Напимер так:

```
struct myclass {
  int operator()(int a) {return a;}
} myobject;
int x = myobject (0);           // function-like syntax with object myobject
```

ФО обычно передают как аргументы функции, например предикат сравнения это функция передаваемая стандартному алгоритму. ФО важны для эффективного использования библиотеки. Разработчик может использовать и указатель на функцию, и обьекты классов в которых определен operator(). В свою очередь это позволяет алгоритмическим шаблонам работать как с указателями на функции, так и с функциональными объектами.

Ниже приведен пример использования алгоритма for_each() с ФО.

```
   http://www.cplusplus.com/reference/algorithm/for_each/
   // for_each example
#include <iostream>     // std::cout
#include <algorithm>    // std::for_each
#include <vector>       // std::vector

void myfunction (int i) {  // function:
  std::cout << ' ' << i;
}

// определение функционального объекта:
struct myclass {           // function object type:
  void operator() (int i) {std::cout << ' ' << i;}
} myobject;

int main () {
  std::vector<int> myvector;
  myvector.push_back(10);
  myvector.push_back(20);
  myvector.push_back(30);

  std::cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myfunction);
  std::cout << '\n';

  // or:
  std::cout << "myvector contains:";
  for_each (myvector.begin(), myvector.end(), myobject);
  std::cout << '\n';

  return 0;
}
```

Например, если мы хотим поэлементно сложить два вектора a и b, содержащие double, и поместить результат в a, мы можем сделать это так:

```
   transform(a.begin(), a.end(), b.begin(), a.begin(), plus‹double›());
```

Если мы хотим отрицать каждый элемент a, мы можем сделать это так:

```
   transform(a.begin(), a.end(), a.begin(), negate‹double›());
```

Соответствующие функции вставят сложение и отрицание.

Чтобы позволить адаптерам и другим компонентам манипулировать функциональными объектами, которые используют один или два параметра, требуется, чтобы они соответственно обеспечили определение типов (typedefs) argument_type и result_type для функциональных объектов, которые используют один параметр, и first_argument_type, second_argument_type и result_type для функциональных объектов, которые используют два параметра.

2.4. Базовые классы (Base)

Для унификации работы с ФО определяют понятие функции с одним и двумя аргументами, и унифицируют типы результат и аргументов с помощью операции typedef:

```
template <class Arg, class Result> struct
    unary_function {
          typedef Arg argument_type;
          typedef Result result_type;
    };

template <class Arg1, class Arg2, class Result> struct
    binary_function {
          typedef Arg1 first_argument_type;
          typedef Arg2 second_argument_type;
          typedef Result result_type;
    };
```

2.5. Арифметические операции (Arithmetic operations)

Библиотека обеспечивает базовые классы функциональных объектов для всех арифметических операторов языка.

```
template <class T> struct
plus:
    binary_function<T, T, T> {
          T operator()(const T& x, const T& y) const {return x + y;}
    };
template <class T> struct
minus:
    binary_function<T, T, T> {
          T operator()(const T& x, const T& y) const {return x - y;}
    };
template <class T> struct
times:
    binary_function<T, T, T> {
          T operator()(const T& x, const T& y) const (return x * y;}
    };
template <class T> struct
divides:
    binary_function<T, T, T> {
          T operator()(const T& x, const T& y) const {return x / y;}
    };
template <class T> struct
modulus:
    binary_function<T, T, T> {
          T operator()(const T& x, const T& y) const {return x % y;}
    };
template <class T> struct
negate:
    unary_function<T, T> {
          T operator()(const T& x) const {return -x;}
    };
```

2.6. Сравнения (Comparisons)

Библиотека обеспечивает базовые классы функциональных объектов для всех операторов сравнения языка

```
template ‹class T› struct
equal_to:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x == y;}
    };

template ‹class T› struct
not_equal_to:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x!= y;}
    };

template ‹class T› struct
greater:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x › y;}
    };

template ‹class T› struct
less:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x ‹ y;}
    };

template ‹class T› struct
greater_equal:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x ›= y;}
    };

template ‹class T› struct
less_equal:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x ‹= y;}
};
```

2.7. Логические операции (Logical operations)

```
template ‹class T› struct
logical_and:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x&& y;}
    };

template ‹class T› struct
logical_or:
    binary_function‹T, T, bool› {
          bool operator()(const T& x, const T& y) const {return x || y;}
    };

template ‹class T› struct
logical_not:
    unary_function‹T, bool› {
          bool operator()(const T& x) const {return!x;}
};
```

3. Постановка задачи

Задача 1.

1. Создать последовательный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции
3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill())
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(), remove_if(), replace_copy(), replace_copy_if()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере заданный элемент (использовать алгоритмы find(), find_if(), count(), count_if())
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each())
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 2.

1. Создать не сортированный ассоциативный контейнер.
2. Заполнить его элементами стандартного типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции
3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill())
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(), remove_if(), replace_copy(), replace_copy_if()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере заданный элемент (использовать аалгоритмы find(), find_if(), count(), count_if())
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each())
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

Задача 3

1. Создать ассоциативный контейнер.
2. 2. Заполнить его элементами стандартного типа (тип указан в варианте). Для пользовательского типа перегрузить необходимые операции
3. Заменить элементы в соответствии с заданием (использовать алгоритмы replace_if(), replace_copy(), replace_copy_if(), fill())
4. Удалить элементы в соответствии с заданием (использовать алгоритмы remove(), remove_if(), replace_copy(), replace_copy_if()).
5. Отсортировать контейнер по убыванию и по возрастанию ключевого поля (использовать алгоритм sort()).
6. Найти в контейнере заданный элемент (использовать аалгоритмы find(), find_if(), count(), count_if())
7. Выполнить задание варианта для полученного контейнера (использовать алгоритм for_each())
8. Для выполнения всех заданий использовать стандартные алгоритмы библиотеки STL.

4.Варианты

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
1	1	*вектор*	*АТД — time*	Пункт 3 Задача X(X = 1...3) Заменить максимальный элемент на заданное значение
	2	*Не сортированный ассоциативный контейнер - словарь с дубликатами.*	*АТД — time*	Пункт 4 Задача X(X = 1..3) Найти минимальный элемент и удалить его из контейнера
	3	*Ассоциативный контейнер - множество.*	*АТД — time*	Пункт 5 Задача X(X = 1...3)К каждому элементу добавить среднее арифметическое контейнера.
2	1	*список*	*АТД — time*	Пункт 3 Задача X(X = 1...3) Найти минимальный элемент и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер — словарь.*	*АТД — time*	Пункт 4 Задача X(X = 1...3) Найти элемент с заданным ключом и удалить его из контейнера.
	3	*Ассоциативный контейнер – множество с дубликатами.*	*АТД — time*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить сумму минимального и максимального элементов контейнера.
3	1	*двунаправленная очередь*	*АТД — time*	Пункт 3 Задача X(X = 1...3) Найти элемент с заданным ключом и добавить его на заданную позицию контейнера
	2	*Не сортированный ассоциативный контейнер – множество с дупликатами.*	*АТД — time*	Пункт 4 Задача X(X = 1..3) Найти элемент с заданным ключом и удалить его из контейнера
	3	*Ассоциативный контейнер – словарь*	*АТД — time*	Пункт 5 Задача X(X = 1...3) Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера.
4	1	*двунаправленная очередь*	*АТД — time*	Пункт 3 Задача X(X = 1...3) Найти максимальный элемент и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер - множество.*	*АТД — time*	Пункт 4 Задача X(X = 1...3) Найти элемент с заданным ключом и удалить его из контейнера.
	3	*Ассоциативный контейнер – словарь с дубликатами.*	*АТД — time*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить среднее арифметическое элементов контейнера.
5	1	*список*	*АТД — time*	Пункт 3 Задача X(X = 1...3) Найти минимальный элемент и добавить его на заданную позицию контейнера.
	2	*Не сортированный ассоциативный контейнер – словарь с дубликатами.*	*АТД — time*	Пункт 4 Задача X(X = 1..3) Найти элементы большие среднего арифметического и удалить их из контейнера.
	3	*Ассоциативный контейнер — множество.*	*АТД — time*	Пункт 5 Задача X(X = 1...3) Каждый элемент домножить на максимальный элемент контейнера.
6	1	*вектор*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти максимальный элемент и добавить его в начало контейнера.
	2	*Не сортированный ассоциативный контейнер – словарь.*	*АТД — money*	Пункт 4 Задача X(X = 1...3) Найти минимальный элемент и удалить его из контейнера.
	3	*Ассоциативный контейнер – множество с дубликатами.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить среднее арифметическое контейнера.

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
7	1	*список*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти минимальный элемент и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер – множество с дубликатами.*	*АТД — money*	Пункт 4 Задача X(X = 1..3) Найти элемент с заданным ключом и удалить его из контейнера.
	3	*Ассоциативный контейнер — словарь.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить сумму минимального и максимального элементов контейнера.
8	1	*список*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти элемент с заданным ключом и добавить его на заданную позицию контейнера.
	2	*Не сортированный ассоциативный контейнер – множество.*	*АТД — money*	Пункт 4 Задача X(X = 1...3) Найти элемент с заданным ключом и удалить его из контейнера.
	3	*Ассоциативный контейнер – словарь с дубликатами.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) Найти разницу между максимальным и минимальным элементами контейнера и вычесть ее из каждого элемента контейнера.
9	1	*двунаправленная очередь*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти максимальный элемент и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер – словарь с дубликатами.*	*АТД — money*	Пункт 4 Задача X(X = 1..3) Найти элемент с заданным ключом и удалить его из контейнера.
	3	*Ассоциативный контейнер — множество.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить среднее арифметическое элементов контейнера.
10	1	*вектор*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти минимальный элемент и добавить его на заданную позицию контейнера.
	2	*Не сортированный ассоциативный контейнер – словарь.*	*АТД — money*	Пункт 4 Задача X(X = 1...3) Найти элементы большие среднего арифметического и удалить их из контейнера.
	3	*Ассоциативный контейнер – множество с дубликатами.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) Каждый элемент домножить на максимальный элемент контейнера.
11	1	*вектор*	*АТД — money*	Пункт 3 Задача X(X = 1...3) Найти среднее арифметическое и добавить его в начало контейнера.
	2	*Не сортированный ассоциативный контейнер – множество с дубликатами.*	*АТД — money*	Пункт 4 Задача X(X = 1..3) Найти элемент с заданным ключом и удалить их из контейнера.
	3	*Ассоциативный контейнер — словарь.*	*АТД — money*	Пункт 5 Задача X(X = 1...3) Из каждого элемента вычесть минимальный элемент контейнера.
12	1	*список*	*АТД — point*	Пункт 3 Задача X(X = 1...3) Найти среднее арифметическое и добавить его на заданную позицию контейнера.
	2	*Не сортированный ассоциативный контейнер – множество.*	*АТД — point*	Пункт 4 Задача X(X = 1...3) Найти элементы с ключами из заданного диапазона и удалить их из контейнера.
	3	*Ассоциативный контейнер – словарь с дубликатами.*	*АТД — point*	Пункт 5 Задача X(X = 1...3) Из каждого элемента вычесть среднее арифметическое контейнера.
13	1	*двунаправленная очередь*	*АТД — point*	Пункт 3 Задача X(X = 1...3) Найти максимальный элемент и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер – словарь с дубликатами.*	*АТД — point*	Пункт 4 Задача X(X = 1..3) Найти элементы с ключами из заданного диапазона и удалить их из контейнера.
	3	*Ассоциативный контейнер — множество.*	*АТД — point*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить среднее арифметическое контейнера.

Вариант	номер задачи	тип контейнера	тип элементов	Что нужно сделать к каждой задаче варианта
14	1	*вектор*	*АТД — point*	Пункт 3 Задача X(X = 1...3) Найти минимальный элемент и добавить его на заданную позицию контейнере.
	2	*Не сортированный ассоциативный контейнер – словарь.*	*АТД — point*	Пункт 4 Задача X(X = 1..3) Найти меньше среднего арифметического и удалить их из контейнера.
	3	*Ассоциативный контейнер — множество с дубликатами.*	*АТД — point*	Пункт 5 Задача X(X = 1...3) Каждый элемент разделить на максимальный элемент контейнера.
15	1	*список*	*АТД — point*	Пункт 3 Задача X(X = 1...3) Найти среднее арифметическое и добавить его в конец контейнера.
	2	*Не сортированный ассоциативный контейнер – множество с дубликатами.*	*АТД — point*	Пункт 4 Задача X(X = 1...3) Найти элементы сключами из заданного диапазона и удалить их из контейнера.
	3	*Ассоциативный контейнер – словарь.*	*АТД — point*	Пункт 5 Задача X(X = 1...3) К каждому элементу добавить сумму минимального и максимального элементов контейнера.

5. Контрольные вопросы

1. Из каких частей состоит библиотека STL?
2. Какие типы контейнеров существуют в STL?
3. Что нужно сделать для использования контейнера STL в своей программе?
4. Что представляет собой итератор?
5. Какие операции можно выполнять над итераторами?
6. Каким образом можно организовать цикл для перебора контейнера с использованием итератора?
7. Какие типы итераторов существуют?
8. Перечислить операции и методы общие для всех контейнеров.
9. Какие операции являются эффективными для контейнера vector? Почему?
10. Какие операции являются эффективными для контейнера list? Почему?
11. Какие операции являются эффективными для контейнера deque? Почему?
12. Перечислить методы, которые поддерживает последовательный контейнер vector.
13. Перечислить методы, которые поддерживает последовательный контейнер list.
14. Перечислить методы, которые поддерживает последовательный контейнер deque.
15. Задан контейнер vector. Как удалить из него элементы со 2 по 5?
16. Задан контейнер vector. Как удалить из него последний элемент?
17. Задан контейнер list. Как удалить из него элементы со 2 по 5?
18. Задан контейнер list. Как удалить из него последний элемент?
19. Задан контейнер deque. Как удалить из него элементы со 2 по 5?
20. Задан контейнер deque. Как удалить из него последний элемент?
21. Написать функцию для печати последовательного контейнера с использованием итератора.
22. Что представляют собой адаптеры контейнеров?
23. Чем отличаются друг от друга объявления stack<int> s и stack<int, list<int> > s?
24. Перечислить методы, которые поддерживает контейнер stack.
25. Перечислить методы, которые поддерживает контейнер queue.
26. Чем отличаются друг от друга контейнеры queue и priority_queue?
27. Задан контейнер stack. Как удалить из него элемент с заданным номером?
28. Задан контейнер queue. Как удалить из него элемент с заданным номером?
29. Написать функцию для печати контейнера stack с использованием итератора.
30. Написать функцию для печати контейнера queue с использованием итератора.