Министерство науки и высшего образования Российской Федерации
Калужский филиал
федерального государственного бюджетного
образовательного учреждения высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»
КАФЕДРА ИУК4 «Программное обеспечение ЭВМ, информационные технологии»

ДОМАШНЯЯ РАБОТА №2

«Стандартная библиотека шаблонов STL»

ДИСЦИПЛИНА: «Высокоуровневое программирование»

Выполнил: студент гр. ИУК4-21Б (Суриков Н.С)
(подпись) (Ф.И.О.)

Проверил: (Пчелинцева Н. И.)
(подпись) (Ф.И.О.)

Дата сдачи (защиты):

Результаты сдачи (защиты):

- Балльная оценка:
- Оценка:

Калуга, 2024

Цель: освоение технологии обобщенного программирования с использованием библиотеки стандартных шаблонов (STL) языка C++.

Задачи:

1. Познакомиться со структурой STL.
2. Изучить основные контейнеры, итераторы, алгоритмы библиотеки STL.
3. Освоить порядок работы со стандартными контейнерами, итераторами и алгоритмами библиотеки STL.

Условие задачи:

Вариант 20

Написать и отладить три программы. Первая программа демонстрирует использование контейнерных классов для хранения встроенных типов данных.

Вторая программа демонстрирует использование контейнерных классов для хранения пользовательских типов данных.

Третья программа демонстрирует использование алгоритмов STL.

В программе № 1 выполнить следующее:

1. Создать объект первого контейнера в соответствии с вариантом задания и заполнить его данными, тип которых определяется вариантом задания.

2. Просмотреть контейнер.

3. Изменить контейнер, удалив из него одни элементы и заменив другие.

4. Просмотреть контейнер, используя для доступа к его элементам итераторы.

5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер.

6. Изменить первый контейнер, удалив из него n элементов после заданного и добавив затем в него все элементы из второго контейнера.

7. Просмотреть первый и второй контейнеры.

В программе № 2 выполнить то же самое, но для данных пользовательского типа (созданный класс из ЛР №1).

В программе № 3 выполнить следующее:

1. Создать контейнер, содержащий объекты пользовательского типа. Тип контейнера выбирается в соответствии с вариантом задания.

2. Отсортировать его по убыванию элементов.

3. Просмотреть контейнер.

4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий заданному условию.

5. Переместить элементы, удовлетворяющие заданному условию в другой (предварительно пустой) контейнер. Тип второго контейнера определяется вариантом задания.

6. Просмотреть второй контейнер.

7. Отсортировать первый и второй контейнеры по возрастанию элементов.

8. Просмотреть их.

9. Получить третий контейнер путем слияния первых двух.

10. Просмотреть третий контейнер.

11. Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит третий контейнер.

12. Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному условию.

Листинг программы:

Task1:

```
 1  #include <algorithm>
 2  #include <iostream>
 3  #include <iterator>
 4  #include <set>
 5
 6  int main()
 7  {
 8      // 1. Создать объект multiset контейнера и заполнить его данными, тип
которых int.
 9      std::multiset<int> mySet = {1, 2, 3, 4, 5, 2, 3, 1};
10
11      // 2. Просмотреть контейнер.
12      std::copy(mySet.begin(), mySet.end(),
std::ostream_iterator<char>(std::cout, " "));
13      std::cout << std::endl;
14
15      // 3. Изменить контейнер, удалив из него одни элементы и заменив другие.
16      mySet.erase(mySet.find(2));
17      mySet.insert(6);
18
```

```
19      // 4. Просмотреть контейнер, используя для доступа к его элементам
итераторы.
20      std::cout << "Содержимое контейнера (с использованием итераторов): ";
21      for (std::multiset<int>::iterator it = mySet.begin(); it != mySet.end(); +
+it)
22      {
23          std::cout << *it << " ";
24      }
25      std::cout << std::endl;
26
27      // 5. Создать второй контейнер этого же класса и заполнить его данными
того же типа, что и первый контейнер.
28      std::multiset<int> mySet2 = {4, 5, 6, 7, 8};
29
30      // 6. Изменить первый контейнер, удалив из него n элементов после
заданного и добавив затем в него все элементы из второго контейнера.
31      int n = 2;
32      std::multiset<int>::iterator it = mySet.find(4);
33      mySet.erase(++it, mySet.end());
34      mySet.insert(mySet2.begin(), mySet2.end());
35
36      // 7. Просмотреть первый и второй контейнеры.
37      std::cout << "Содержимое первого контейнера: ";
38      std::copy(mySet.begin(), mySet.end(),
std::ostream_iterator<char>(std::cout, " "));
39      std::cout << std::endl;
40
41      std::cout << "Содержимое второго контейнера: ";
42      std::copy(mySet.begin(), mySet.end(),
std::ostream_iterator<char>(std::cout, " "));
43      std::cout << std::endl;
44
45      return 0;
46  }
```

Task2:

```
1  #include "./Car/Car.h"
2  #include <algorithm>
3  #include <iostream>
4  #include <set>
5  #include <iterator>
6  #include <string>
7  #include <vector>
8
9  using namespace SNS;
10
11 int main()
12 {
13     // 1. Создать объект multiset контейнера и заполнить его данными, тип которых
пользовательский.
14     std::multiset<Car, std::greater<Car>> carSet = {
```

```
15          SNS::Car("Tesla", 60000, 2020, "Электрический автомобиль", "Autopilot, Regenerative Braking"),
16          SNS::Car("BMW", 50000, 2019, "Премиум-седан", "Turbo, Leather Seats"),
17          SNS::Car("Toyota", 30000, 2018, "Надежный семейный автомобиль", "Fuel Efficient, Spacious"),
18          SNS::Car("Honda", 25000, 2017, "Компактный городской автомобиль", "Reliable, Fuel Efficient")};
19
20      // 2. Просмотреть контейнер.
21      std::cout << "Содержимое контейнера: " << std::endl;
22      for (const auto &car : carSet)
23      {
24          std::cout << car << std::endl;
25      }
26      std::cout << std::endl;
27
28      // 3. Изменить контейнер, удалив из него одни элементы и заменив другие.
29      auto it = carSet.find(Car("BMW", 50000, 2019, "Премиум-седан", "Turbo, Leather Seats"));
30      if (it != carSet.end())
31      {
32          carSet.erase(it);
33      }
34      carSet.insert(Car("Audi", 45000, 2020, "Премиум-седан", "Quattro, LED Lights"));
35
36      // 4. Просмотреть контейнер, используя для доступа к его элементам итераторы.
37      std::cout << "Содержимое контейнера (с использованием итераторов): " << std::endl;
38      for (auto it = carSet.begin(); it != carSet.end(); ++it)
39      {
40          std::cout << *it << std::endl;
41      }
42      std::cout << std::endl;
43
44      // 5. Создать второй контейнер этого же класса и заполнить его данными того же типа, что и первый контейнер.
45      std::multiset<Car, std::greater<Car>> carSet2 = {
46          Car("Hyundai", 20000, 2016, "Надежный компактный автомобиль", "Fuel Efficient, Affordable"),
47          Car("Subaru", 35000, 2018, "Полноприводный автомобиль", "AWD, Boxer Engine"),
48          Car("Nissan", 28000, 2017, "Городской кроссовер", "Spacious, Stylish")};
49
50      // 6. Изменить первый контейнер, удалив из него n элементов после заданного и добавив затем в него все элементы из второго контейнера.
51      int n = 1;
52      it = carSet.find(Car("Toyota", 30000, 2018, "Надежный семейный автомобиль", "Fuel Efficient, Spacious"));
53      if (it != carSet.end())
```

```
    {
        carSet.erase(++it, carSet.end());
    }
    carSet.insert(carSet2.begin(), carSet2.end());

    // 7. Просмотреть первый и второй контейнеры.
    std::cout << "Содержимое первого контейнера: " << std::endl;
    for (const auto &car : carSet)
    {
        std::cout << car << std::endl;
    }
    std::cout << std::endl;

    std::cout << "Содержимое второго контейнера: " << std::endl;
    for (const auto &car : carSet)
    {
        std::cout << car << std::endl;
    }
    std::cout << std::endl;

    return 0;
}
```

Task3:

```
#include <iostream>
#include <set>
#include <vector>
#include <algorithm>
#include <functional>

class Book {
private:
   std::string title;
    std::string author;
    int year;
    double price;

public:
    Book(const std::string& title, const std::string& author, int year, double price)
        : title(title), author(author), year(year), price(price) {}

    std::string getTitle() const { return title; }
    std::string getAuthor() const { return author; }
    int getYear() const { return year; }
    double getPrice() const { return price; }
```

```
22
23      bool operator>(const Book& other) const { return price > other.price; }
24      bool operator<(const Book& other) const { return price < other.price; }
25
26      friend std::ostream& operator<<(std::ostream& os, const Book& book);
27  };
28
29  std::ostream& operator<<(std::ostream& os, const Book& book) {
30      os << "Title: " << book.title << ", Author: " << book.author << ", Year: " << book.year << ",
Price: " << book.price;
31      return os;
32  }
33
34  int main() {
35      // 1. Создать multiset, содержащий объекты пользовательского типа.
36      std::multiset<Book, std::greater<Book>> bookSet = {
37          Book("The Great Gatsby", "F. Scott Fitzgerald", 1925, 12.99),
38          Book("To Kill a Mockingbird", "Harper Lee", 1960, 9.99),
39          Book("1984", "George Orwell", 1949, 7.99),
40          Book("Harry Potter and the Sorcerer's Stone", "J.K. Rowling", 1997, 14.99)
41      };
42
43      // 2. Отсортировать его по убыванию элементов.
44      // Сортировка по убыванию происходит благодаря использованию std::greater<Book> в
качестве сравнивающего предиката.
45
46      // 3. Просмотреть контейнер.
47      std::cout << "Содержимое контейнера (отсортировано по убыванию):" << std::endl;
48      for (const auto& book : bookSet) {
49          std::cout << book << std::endl;
50      }
51      std::cout << std::endl;
52
53      // 4. Используя подходящий алгоритм, найти в контейнере элемент, удовлетворяющий
заданному условию.
54      auto it = std::find_if(bookSet.begin(), bookSet.end(), [](const Book& book) {
55          return book.getYear() >= 1950;
56      });
57      if (it != bookSet.end()) {
58          std::cout << "Найдена книга, соответствующая условию: " << *it << std::endl;
59      } else {
60          std::cout << "Книга, соответствующая условию, не найдена." << std::endl;
61      }
62      std::cout << std::endl;
63
64      // 5. Переместить элементы, удовлетворяющие заданному условию в другой
(предварительно пустой) контейнер. Тип второго контейнера vector.
```

```
65      std::vector<Book> booksToMove;
66      std::copy_if(bookSet.begin(), bookSet.end(), std::back_inserter(booksToMove), [](const Book&
book) {
67          return book.getPrice() > 10.0;
68      });
69
70      // 6. Просмотреть второй контейнер.
71      std::cout << "Содержимое второго контейнера (booksToMove):" << std::endl;
72      for (const auto& book : booksToMove) {
73          std::cout << book << std::endl;
74      }
75      std::cout << std::endl;
76
77      // 7. Отсортировать первый и второй контейнеры по возрастанию элементов.
78      std::sort(booksToMove.begin(), booksToMove.end());
79
80      // 8. Просмотреть их.
81      std::cout << "Содержимое первого контейнера (bookSet), отсортированного по
возрастанию:" << std::endl;
82      for (const auto& book : bookSet) {
83          std::cout << book << std::endl;
84      }
85      std::cout << std::endl;
86
87      std::cout << "Содержимое второго контейнера (booksToMove), отсортированного по
возрастанию:" << std::endl;
88      for (const auto& book : booksToMove) {
89          std::cout << book << std::endl;
90      }
91      std::cout << std::endl;
92
93      // 9. Получить третий контейнер путем слияния первых двух.
94      std::multiset<Book> mergedSet;
95      mergedSet.insert(bookSet.begin(), bookSet.end());
96      mergedSet.insert(booksToMove.begin(), booksToMove.end());
97
98      // 10. Просмотреть третий контейнер.
99      std::cout << "Содержимое третьего контейнера (mergedSet):" << std::endl;
100      for (const auto& book : mergedSet) {
101          std::cout << book << std::endl;
102      }
103      std::cout << std::endl;
104
105      // 11. Подсчитать, сколько элементов, удовлетворяющих заданному условию, содержит
третий контейнер.
106      int count = std::count_if(mergedSet.begin(), mergedSet.end(), [](const Book& book) {
107          return book.getYear() >= 1950;
```

```
108     });
109     std::cout << "Количество элементов, удовлетворяющих условию (год >= 1950): " << count
<< std::endl;
110
111     // 12. Определить, есть ли в третьем контейнере элемент, удовлетворяющий заданному
условию.
112     auto found = std::find_if(mergedSet.begin(), mergedSet.end(), [](const Book& book) {
113         return book.getAuthor() == "J.K. Rowling";
114     });
115     if (found != mergedSet.end()) {
116         std::cout << "Элемент, удовлетворяющий условию (автор 'J.K. Rowling'), найден." <<
std::endl;
117     } else {
118         std::cout << "Элемент, удовлетворяющий условию (автор 'J.K. Rowling'), не найден." <<
std::endl;
119     }
120
121     return 0;
122 }
123
```

Демонстрация работы:

Содержимое контейнера (с использованием итераторов): 1 1 2 3 3 4 5 6
Содержимое первого контейнера: 1 1 2 3 3 4 4 5 6 7 8
Содержимое второго контейнера: 4 5 6 7 8

Содержимое контейнера:
Brand: Tesla
Price: 60000
Year: 2020
Description: Электрический автомобиль
Features: Autopilot, Regenerative Braking

Brand: BMW
Price: 50000
Year: 2019
Description: Премиум-седан
Features: Turbo, Leather Seats

Brand: Toyota
Price: 30000

Year: 2018
Description: Надежный семейный автомобиль
Features: Fuel Efficient, Spacious

Brand: Honda
Price: 25000
Year: 2017
Description: Компактный городской автомобиль
Features: Reliable, Fuel Efficient

Содержимое контейнера (с использованием итераторов):
Brand: Tesla
Price: 60000
Year: 2020
Description: Электрический автомобиль
Features: Autopilot, Regenerative Braking

Brand: Audi
Price: 45000
Year: 2020
Description: Премиум-седан
Features: Quattro, LED Lights

Brand: Toyota
Price: 30000
Year: 2018
Description: Надежный семейный автомобиль
Features: Fuel Efficient, Spacious

Brand: Honda
Price: 25000
Year: 2017
Description: Компактный городской автомобиль
Features: Reliable, Fuel Efficient

Содержимое первого контейнера:
Brand: Tesla

Price: 60000
Year: 2020
Description: Электрический автомобиль
Features: Autopilot, Regenerative Braking

Brand: Audi
Price: 45000
Year: 2020
Description: Премиум-седан
Features: Quattro, LED Lights

Brand: Toyota
Price: 30000
Year: 2018
Description: Надежный семейный автомобиль
Features: Fuel Efficient, Spacious

Brand: Subaru
Price: 35000
Year: 2018
Description: Полноприводный автомобиль
Features: AWD, Boxer Engine

Brand: Nissan
Price: 28000
Year: 2017
Description: Городской кроссовер
Features: Spacious, Stylish

Brand: Hyundai
Price: 20000
Year: 2016
Description: Надежный компактный автомобиль
Features: Fuel Efficient, Affordable

Содержимое второго контейнера:
Brand: Tesla
Price: 60000

Year: 2020
Description: Электрический автомобиль
Features: Autopilot, Regenerative Braking

Brand: Audi
Price: 45000
Year: 2020
Description: Премиум-седан
Features: Quattro, LED Lights

Brand: Toyota
Price: 30000
Year: 2018
Description: Надежный семейный автомобиль
Features: Fuel Efficient, Spacious

Brand: Subaru
Price: 35000
Year: 2018
Description: Полноприводный автомобиль
Features: AWD, Boxer Engine

Brand: Nissan
Price: 28000
Year: 2017
Description: Городской кроссовер
Features: Spacious, Stylish

Brand: Hyundai
Price: 20000
Year: 2016
Description: Надежный компактный автомобиль
Features: Fuel Efficient, Affordable

Вывод: в ходе выполнения лабораторной работы были получены практические навыки использования библиотеки шаблонов STL.