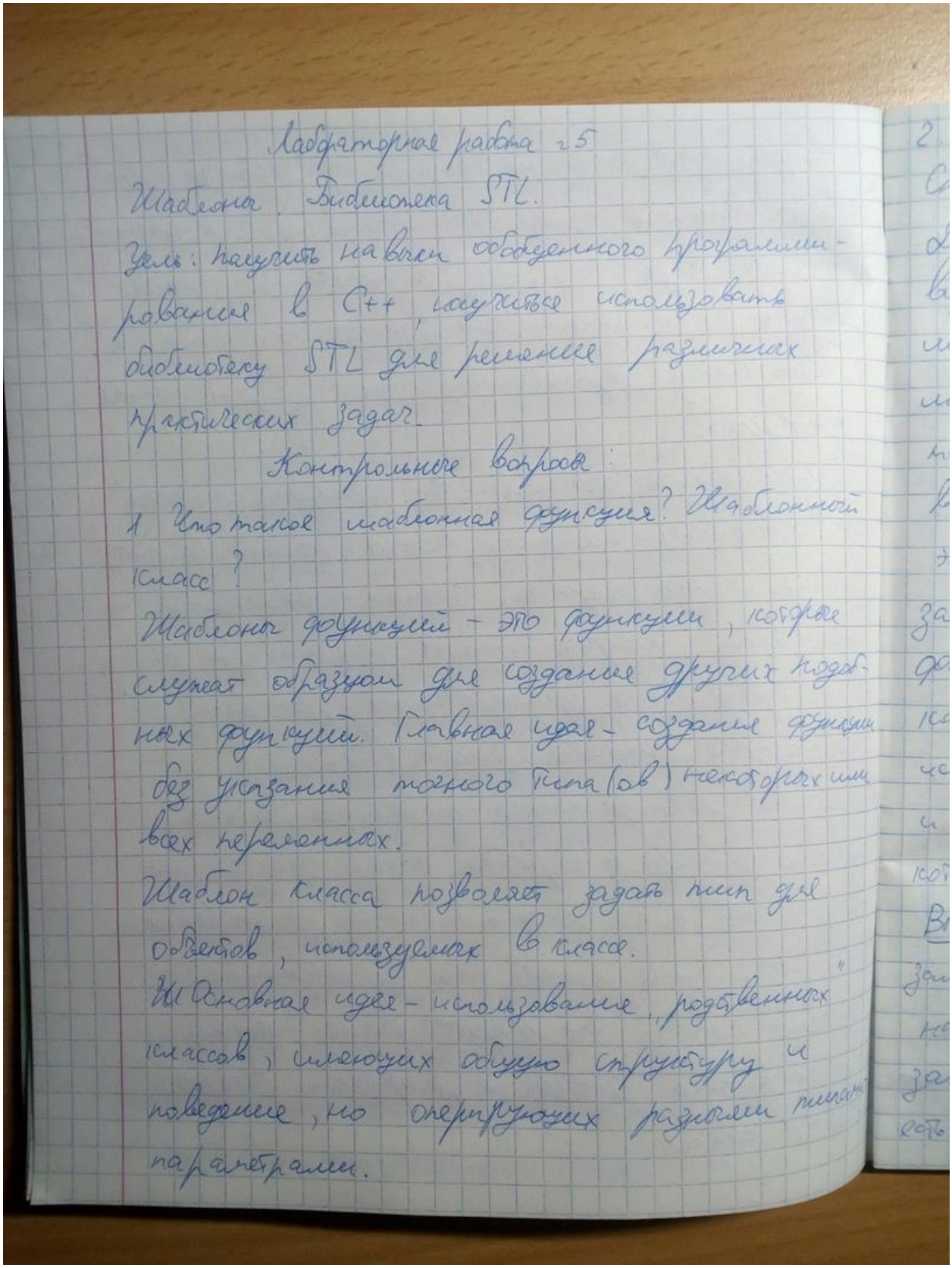
Федоров Максим (ИВТ-3). Отчет. Лабораторная работа №5.

Контрольные вопросы:



2. Как компилятор работает с шаблонами?
Статические члены шаблонных классов.
Для использования шаблона компилятор должен видеть как определение шаблона, так и тип шаблона, применяемый для создания экземпляра шаблона. Когда заголовочный файл подключается в main.cpp, то определение шаблона класса копируется в это файл. В main.cpp видит, что нам нужна экземпляра шаблона класса, он создаст их, а затем скомпилирует весь этот код как часть файла main.cpp. Однако, когда дело дойдет до компиляции (cpp), компилятор забудет, что мы использовали экземпляр шаблона и в main.cpp и не создаст экземпляр шаблона функции --, которая необходим для выполнения программы.
Вывод: компилятор копирует шаблон класса, заменяя типы параметров шаблона класса на фактические (передаваемые) типа данных, а затем компилирует эту копию. Если у нас есть шаблон класса, но мы его не используем, то

компилятор не будет его даже компилировать.
В шаблоне класса могут быть объявлены статические члены класса. Каждый конкретизированный экземпляр шаблона класса имеет собственный набор таких членов.
Если тип статического члена класса зависит от параметра шаблона класса, то и сам член является шаблоном.
Т.к. статические члены класса только объявляются внутри класса, то шаблон статического члена класса должен быть определен вне шаблона класса. Любой статический член шаблона класса конкретизируется только в том случае, если он реально используется в программе.
3. Что означает явная специализация шаблона? Частичная специализация шаблона?
Явная специализация шаблона - это такое определение, в кот. за ключевым словом template следует пара угловых скобок <>, а за ними - определение специализированного шаблона:

template <> тип_возврата
функция <параметры шаблона>
(параметры_функции) {реализация}
Частичная специализация шаблона позволяет
выполнить специализацию шаблона класса
(но не функции), где некоторые (но не все)
параметры шаблона явно определены.
4. Что такое идиома SFINAE?
Идиома SFINAE - механизм языка C++,
связанный с шаблонами и перегрузкой функции.
Значение SFINAE: если не получается
рассчитать окончательные типы аргументов
перегруженной шаблонной функции, компи-
лятор не выбрасывает ошибку, а ищет
другую подходящую перегрузку.
5. Кратко опишите типы контейнеров STL?
Контейнеры STL делятся на три основных типа:
1) последовательные - это контейнерные классы,
элементы коих находятся в последовательности.
Их определяют особенностью хар-ой нрв.: то, что

вы можете добавить свой элемент в любое место контейнера.

6 контейнеров:

- std::vector - динамический массив, способный увеличиваться по мере необходимости для содержания всех своих элементов
- std::deque - двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов;
- std::array - массив;
- std::list - двусвязный список, каждый элемент кот. содержит 2 указателя: один указывает на след. элемент списка, а другой - на предыдущий элемент списка.
- std::format_list;
- std::basic_string

+ 2) ассоциативные - это контейнерные классы, кот. автоматически сортируют все свои элементы
- set - контейнер, в кот. хранятся только уникальные элементы, и повторения запрещены;

- multiset - это Set, но в котором допускаются повторяющиеся элементы;
- map (ассоциативный массив) - это Set, в кот. каждый элемент является "парой", ключ-значение".
- multimap (словарь) - map, кот. допускает дублирование ключей.

3) Адаптеры - это специальные предопределенные контейнерные классы, кот. адаптированы для выполнения конкретных заданий.
- Stack - контейнерный класс, элементы кот. работают по принципу LIFO; (Последний пришел, первым ушел)
- queue - контейнерный класс, элементы кот. работают по принципу FIFO (Первым пришел, первым ушел);
- priority_queue - это тип очереди, в кот. все элементы отсортированы.

6. Чем отличаются контейнера vector, deque и list? Как они реализованы в STL?

vector - это динамический массив, способный увеличиваться по мере необходимости для содержания всех своих элементов. Класс vector

обеспечивает произвольный доступ к своим элементам через оператор индексации [], а также поддерживает добавление и удаление элементов

```
std::vector<int> vect;
for(int count=0; count<5; ++count)
    vect.push_back(10-count); // доб. числа в конец
```

deque - двусторонняя очередь, реализованная в виде динамического массива, который может расти с обоих концов.

```
std::deque<int> deq;
for(int count=0; count<4; ++count)
{
    deq.push_back(count); // в конец
    deq.push_front(10-count); // в начало
}
```

list - двусвязный список, каждый элемент которого содержит 2 указателя: один указывает на следующий элемент списка, а другой - на предыдущий. Элемент списка list предоставляет доступ только к началу и концу списка - произвольный доступ запрещен

Преимуществом двусвязного списка является то, что добавление элементов происходит очень быстро, если вы, конечно, знаете, куда хотите добавлять. Обычно для перебора элементов двусвязного списка используются итераторы.

```
std::list<int> numbers = {1, 2, 3, 4, 5};
int first = numbers.front(); // 1
int last = numbers.back(); // 5
```

7. Кратко опишите типы итераторов STL и методологию работы с ними.

Функции итераторов:

- оператор * - возвращает элемент, на кот. в данный момент указывает итератор
- оператор ++ перемещает итератор к след. элементу контейнера;
- операторы == и != используются для опред. того, указывают ли два итератора на один и тот же элемент или нет;
- оператор = присваивает итератору новую позицию.

Каждый контейнерный класс имеет 4 основных метода для работы с итераторами:

- метод begin() возв. итератор, представляющий начальный элемент контейнера;
- метод end() возв. итератор, указывающий на элемент, который находится после последнего элемента в контейнере.
- метод cbegin() возвр. константный итератор, ... начальный элемент ...)
- метод cend() возв. константный итератор ... end().

Два типа итераторов:

- container::iterator - итератор для чтения/записи
- container::const_iterator - итератор только для чтения

Итерация по вектору:

std::vector<int>::const_iterator it)

Итерация по списку:

std::list<int>::const_iterator it;

Итерация по set-у:

std::set<int>::const_iterator it;

n begin

Итерация по ассоциативному массиву:

```
std::map<int, std::string> myMap
myMap.insert(std::make_pair(3, "cat"));
std::map<int, std::string>::const_iterator it;
```

8. Что такое ассоциативный массив? Кратко опишите контейнеры map, multimap, set.

Ассоциативный массив - абстрактный тип данных, позволяющий хранить пары вида „(ключ, значение)" и поддерживающий операции добавления пары, а также поиска и удаления пары по ключу.

Set - это контейнер, в котором хранятся только уникальные элементы, и повторение запрещено. Элементы сортируются в соответствии с их значением.

map (ассоциативный массив) - это set, в кот. каждый элемент является парой „ключ - значение". „Ключ" используется для сортировки и индексации данных и должен быть уникальным. А „значение" - это фактические данные

multimap (словарь) – это map, кот. допускает дублирование ключей. Все ключи отсортированы в порядке возрастания, и в нем хранятся значение по ключу.

9. Кратко опишите адаптеры контейнеров stack, очередь, priority - очередь.

Stack - контейнерный класс, элементы кот. работают по принципу LIFO (Последним пришел, первым ушел), т. е элементы добавляются в конец контейнера и удаляются оттуда же. Обычно в стеках используется дека в качестве последовательного контейнера по умолчанию, но можно использовать vector или list.

Очередь - контейнерный класс, элементы кот. работают по принципу FIFO (Первым Пришел, Первым Ушел), т. е. элементы добавляются в конец контейнера, но удаляются из начала контейнера. По умолчанию в очереди используется дека в качестве последов. контейнера, но можно использовать list.

Priority - дикие (очередь с приоритетом) - это тип очереди, в кот. все элементы отсортированы. При добавлении элемента, он автоматически сортируется. Элемент с наивысшим приоритетом (самый большой приоритет) находится в самом начале очереди с приоритетом, также, как и удаление элементов выполняется с самого начала очереди с приоритетом.

10. Что такое функтор? Где и как функторы используются в библиотеке STL?

Функторы - это в C++ прежде всего класс с перегруженной операцией (); любые объекты, которые умеют вести себя как функции: это указатели на функции, но сами функции и ссылки на функции функторами не являются, т.к. они не объекты. Функторы полезны там, где функции должны вводить себя как объекты.

Пример использования функтора:

```
class Square {
public:
 int operator() (const int value) const {
    return value * value;
    }
};
int main() {
 Square sq;
 std::cout << sq(10) << '\n';
```

Внутри main использование объекта sq
уподоблено механизму вызова функции.

11. Что означает аббревиатура RAII?
Что такое Умный указатель?

RAII (Resource Asquisition Is Initialization) - программная идиома ООП, которая заключается в том, что с помощью конструктора и деструктора получения некоторого ресурса неразрывно совмещается с инициализацией, а освобождение - с уничтожением объекта.

Умный указатель - это класс, предназначенный для управления динамически выделенной памятью и освобождения освобождение выдел-

кой памяти при выходе объекта этого класса
из области видимости.