**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего образования**

**«Чувашский государственный университет имени И.Н. Ульянова»**

**(ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»)**

**Факультет информатики и вычислительной техники**

**Кафедра вычислительной техники**

**ЗАДАНИЕ**

**студенту-практиканту**

***Ярдыкову Эдуарду Евгеньевичу, ИВТ-41-22***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

ФИО студента-практиканта, группа

для прохождения учебной практики (ознакомительной практики) на

***кафедре вычислительной техники факультета информатики и вычислительной техники ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

наименование профильной организации/подразделения университета

1. [Ведение](http://www.xn-----8kcodrdcygecwgg0byh.xn--p1ai/vvedenie-kursovoi-raboty) и [оформление дневника практики](http://www.xn-----8kcodrdcygecwgg0byh.xn--p1ai/dnevnik-po-praktike).
2. Прохождение инструктажа по охране труда, технике безопасности, пожарной безопасности, а также ознакомление с правилами внутреннего трудового распорядка организации, предоставляющей место для прохождения практики.
3. Выполнение индивидуального задания:

* ознакомление с базой практики (профильной организацией), выпускаемой продукцией, используемыми программными средствами для решения практических задач;
* изучение технологии создания программных средств вычислительной техники и автоматизированных систем;
* знакомство с архитектурой вычислительных систем и оптимизацией программного кода для достижения максимальной производительности.
* приобретение и закрепление навыков разработки алгоритмов и программ;
* проведение исследовательских работ в области технических инноваций и современных тенденций в области вычислительной техники и программирования.
* приобретение навыков разработки и оформления отчетности и (или) программной документации/ документации пользователя;
* оформление отчета по практике в соответствии с рекомендациями п.п. 6,7 программы практики.

1. Планируемый результат:

расширение и закрепление теоретических знаний и умений в области программирования для дальнейшей разработки и успешного внедрения собственных программных решений или проектов.

Руководитель практики от кафедры \_\_\_\_\_\_\_\_\_\_\_/ Юрьева Е.В.

Дата выдачи задания 26 июня 2023 года

Согласовано:

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_\_/ Юрьева Е.В.

Дата согласования 26 июня 2023 года

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего образования**

**«Чувашский государственный университет имени И.Н. Ульянова»**

**(ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»)**

**Факультет информатики и вычислительной техники**

**Кафедра вычислительной техники**

ОТЧЕТ

О УЧЕБНОЙ ПРАКТИКЕ

(ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ)

на базе ***кафедры вычислительной техники факультета информатики и вычислительной техники ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»***

|  |  |  |  |
| --- | --- | --- | --- |
| Обучающийся 1 курса, направление подготовки  «Информатика и вычислительная техника», группа ИВТ-41-22 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(08.07.2023г.) | | Ярдыков Э.Е. |
|  | подпись, дата | |  |
|  |  | |  |
|  |  | |  |
| Руководитель,  ст.преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись, дата | | Юрьева Е.В. |
|  |  |
|  |  | |  |
| ст.преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | Юрьева Е.В. |
|  | подпись, дата | |  |
|  |  | |  |
| Заведующий кафедрой  вычислительной техники, доцент | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ подпись, дата | | Щипцова А.В. |
|  |  |
|  |  | |  |
|  | | Чебоксары 2023 |  |

**СОДЕРЖАНИЕ**

[Введение ……………………………………………………………………………………...…..4](#_Toc471850995)

[Стандартная библиотека STL в C++ …………………………………………………………....5](#_Toc471850995)

Вектор…………………………………………………………………………………………….7

Двусвязный список[………………………………………………………………………………](#_Toc471850997)9

Дек[……………………………………………………………………………](#_Toc471850998)………………….11

Очередь[……………………………………………………………………………](#_Toc471850998)…………….13

Стек[………………………………………………………………………………](#_Toc471850998)……………...15

Множество уникальных элементов [……………………………………………](#_Toc471850998)……………...17

Ассоциативный массив[………………………………………………………………](#_Toc471850998)………...19

[Заключение ……………………………………………………………………………………...2](#_Toc471850999)2

[Решение задач …………………………………………………………………………………..2](#_Toc471850999)3

[Список использованных источников ………………………………………………………….2](#_Toc471850999)5

**Введение.**

Язык программирования C++ считается одним из самых мощных и универсальных инструментов в мире разработки программного обеспечения. Его популярность обусловлена не только высокой производительностью, но и богатым набором возможностей.

Язык программирования C++ используется во множестве областей и сфер деятельности, благодаря своей универсальности, производительности и богатому набору возможностей. Например, научные и инженерные программы, включая расчеты, симуляции и моделирование, разрабатываются на C++ из-за его производительности и возможности взаимодействия с библиотеками для численных вычислений.

Одним из ключевых преимуществ C++ является близкое к железу управление памятью, что позволяет эффективно оптимизировать ресурс. Это очень необходимо, например, для создания операционных систем, драйверов устройств, встроенных систем управления и других системных компонентов.

Однако одной из самых сильных сторон C++ является его стандартная библиотека шаблонов (STL, Standard Template Library). STL предоставляет разработчикам богатый набор контейнеров, алгоритмов и функций, которые значительно упрощают и ускоряют разработку. Эта библиотека обеспечивает готовые решения для множества задач, начиная от сортировки и поиска данных до работы с динамическими структурами данных, такими как списки и деревья.

С++ и STL идеально дополняют друг друга, позволяя создавать эффективные и надежные приложения, а также значительно упрощают жизнь разработчиков, устраняя необходимость писать множество стандартных функций и алгоритмов с нуля.

**Стандартная библиотека STL в C++**

Библиотека STL (Стандартная библиотека шаблонов) является частью языка программирования C++ и представляет собой набор алгоритмов, контейнеров, итераторов и вспомогательных функций, которые обеспечивают единообразный и эффективный способ работы с различными типами данных. Библиотека STL была разработана для облегчения процесса программирования и стандартизации кода.

Основные компоненты библиотеки STL:

1. Контейнеры: вектор (vector), список (list), стек (stack), очередь (queue), дек (deque) и множество (set), а также адаптеры для массивов (array) и указателей (pointer). Контейнеры предназначены для хранения и управления данными различных типов.
2. Алгоритмы: поиск (find), сортировка (sort), замена (replace), объединение (merge), пересечение (intersection), и т.д. Алгоритмы позволяют выполнять различные операции над данными, хранящимися в контейнерах.
3. Итераторы: обеспечивают удобный и безопасный доступ к элементам контейнеров. Итераторы позволяют перемещаться по элементам контейнера, а также выполнять различные операции с ними.
4. Вспомогательные функции: функции для работы с итераторами (advance, distance, equal\_range), функции для создания и удаления объектов (make\_pair, make\_move\_iterator, unique), и т.п.

Вот основные компоненты библиотеки STL:

1. Контейнеры:
   * std::vector - Динамический массив.
   * std::list - Двусвязный список.
   * std::deque - Двусторонняя очередь.
   * std::queue - Очередь.
   * std::stack - Стек.
   * std::set - Множество уникальных элементов (упорядоченное).
   * std::map - Ассоциативный массив (упорядоченный).
   * std::unordered\_set - Множество уникальных элементов (неупорядоченное).
   * std::unordered\_map - Ассоциативный массив (неупорядоченный).
   * std::multiset - Множество элементов с возможностью дублирования (упорядоченное).
   * std::multimap - Ассоциативный массив с возможностью дублирования (упорядоченный).
2. Итераторы:
   * Input Iterator - Для чтения данных из контейнера.
   * Output Iterator - Для записи данных в контейнер.
3. Алгоритмы:
   * Разнообразные алгоритмы для работы с контейнерами, такие как сортировка, поиск, трансформация и многие другие.
4. Функциональные объекты:
   * Функциональные объекты, такие как функции, функции-члены и лямбда-выражения, которые могут быть использованы с алгоритмами.

Библиотека STL играет важную роль в языке программирования C++, так как она предоставляет программистам мощный набор инструментов для работы с данными и алгоритмами. Использование библиотеки STL позволяет создавать более эффективный и надежный код, а также упрощает его написание и понимание.

**Вектор**

Вектор — это один из наиболее популярных контейнеров в библиотеке STL C++. Он представляет собой динамический массив, который автоматически увеличивает свой размер при добавлении элементов. Вот некоторые из его преимуществ:

1. **Динамический размер**: Вектор может изменять свой размер по мере необходимости, что делает его удобным для работы с коллекциями переменной длины.
2. **Быстрый доступ к элементам**: Элементы вектора нумеруются с нуля, и вы можете быстро получать доступ к элементам по индексу.
3. **Эффективная вставка и удаление в конце**: Вставка и удаление элементов в конец вектора выполняются за амортизированное константное время.
4. **Легкая передача данных**: Векторы обеспечивают легкую передачу данных между функциями и классами.
5. **Буферизация памяти**: Векторы часто используют буферизацию памяти, что может привести к более эффективному использованию памяти и улучшению производительности.

Некоторые из основных методов вектора:

* push\_back()- Добавляет элемент в конец вектора.
* pop\_back()- Удаляет элемент с конца вектора.
* size()- Возвращает текущий размер вектора (количество элементов).
* capacity()- Возвращает текущую емкость вектора (количество элементов, которые могут быть сохранены без выделения дополнительной памяти).
* at()- Позволяет получить доступ к элементу по индексу с проверкой выхода за пределы вектора.
* front()- Возвращает первый элемент вектора.
* back()- Возвращает последний элемент вектора.

Пример использования std::vector можно увидеть на рисунке 1.

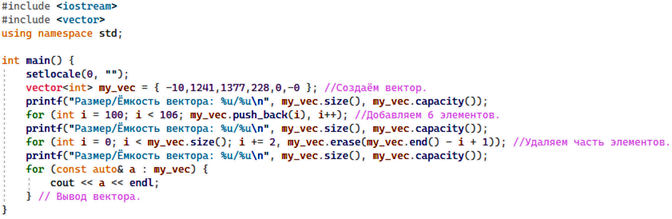


Рисунок 1 - Пример использования std::vector

**Двусвязный список**

Двусвязный список в C++ (представленный библиотекой std::list) состоит из узлов, содержащих значения и ссылающихся на предыдущие и последующие узлы. Эта структура списка предлагает некоторые преимущества и недостатки в сравнении с другими структурами, например, массивами и векторами. Преимущества двусвязного списка:

1. **Вставка и удаление**: Двусвязный список позволяет эффективно выполнять вставку и удаление элементов в любой части списка без необходимости копирования других элементов.
2. **Память**: Он использует память более эффективно, чем массивы или векторы, так как каждый элемент связан только с предыдущим и следующим элементами.
3. **Итерация в обратном направлении**: Возможность итерации в обратном направлении, так как каждый элемент имеет указатель на предыдущий элемент.

Однако двусвязный список имеет некоторые недостатки, такие как дополнительное потребление памяти для хранения указателей и более сложные операции доступа к элементам по индексу.

Некоторые из основных методов std::list:

* push\_back() - Добавляет элемент в конец списка.
* push\_front() - Добавляет элемент в начало списка.
* pop\_back() - Удаляет последний элемент списка.
* pop\_front() - Удаляет первый элемент списка.
* insert() - Вставляет элемент в указанную позицию.
* erase() - Удаляет элемент из списка.
* size() - Возвращает текущий размер списка.
* empty() - Проверяет, пуст ли список.
* begin() - Возвращает итератор, указывающий на начало списка.
* end() - Возвращает итератор, указывающий на конец списка.
* rbegin() - Возвращает итератор для обратной итерации с конца.
* rend() - Возвращает итератор для обратной итерации с начала.

Пример использования std::list можно увидеть на рисунке 2.

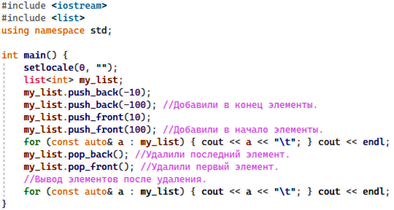


Рисунок 2 - Пример использования std::list

**Дек**

Двусторонняя очередь, или deque (от англ. double-ended queue), в C++ является контейнером, аналогичным вектору, однако обеспечивает эффективное выполнение операций вставки и удаления элементов как с начала, так и с конца. Это свойство делает двустороннюю очередь полезной в случаях, когда необходимо быстрое добавление и удаление элементов с обоих концов контейнера.

Название “deque” происходит от английского термина “double-ended queue”, что означает “двусторонняя очередь”. Такое название отражает тот факт, что двусторонняя очередь позволяет вставлять и удалять элементы как с начала очереди (front), так и с ее конца (back). Это отличает двустороннюю очередь от других структур данных, таких как стеки и очереди, которые обычно поддерживают только одну операцию добавления и удаления элемента.

Преимущества двусторонней очереди:

1. **Эффективные операции вставки и удаления**: двусторонняя очередь позволяет выполнять операции вставки и удаления элементов как в начало, так и в конец контейнера за константное время.
2. **Произвольный доступ к элементам**: вы можете получить доступ к элементам двусторонней очереди по индексу, подобно вектору.
3. **Буферизация памяти**: подобно вектору, двусторонняя очередь может использовать буферизацию памяти для более эффективного управления ресурсами.

Некоторые из основных методов двусторонней очереди (std::deque):

* push\_back() - Добавляет элемент в конец очереди.
* push\_front() - Добавляет элемент в начало очереди.
* pop\_back() - Удаляет последний элемент очереди.
* pop\_front() - Удаляет первый элемент очереди.
* at() - Получает доступ к элементу по индексу с проверкой выхода за пределы.
* size() - Возвращает текущий размер очереди.
* empty() - Проверяет, пуста ли очередь.
* begin() - Возвращает итератор, указывающий на начало очереди.
* end() - Возвращает итератор, указывающий на конец очереди.

Пример использования std::deque можно увидеть на рисунке 3.

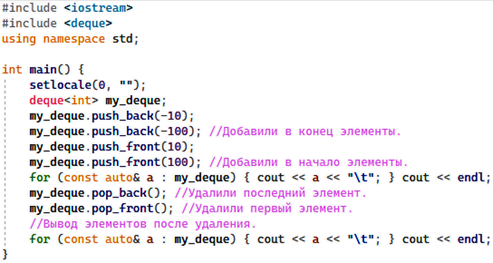


Рисунок 3 - Пример использования std::deque

**Очередь**

Очередь в C++ (реализованная с помощью std::queue) — это абстрактное представление структуры данных типа FIFO (First In, First Out), что означает, что элемент добавляется в конец очереди (rear) и удаляется из начала очереди (front). Очередь часто используется для организации элементов в порядке их добавления, чтобы обеспечить упорядоченное обслуживание этих элементов.

Преимущества очереди:

1. **Соблюдение порядка элементов**: Очередь гарантирует, что элементы будут обрабатываться в том порядке, в котором они были добавлены.
2. **Простота использования**: Очередь предоставляет простой интерфейс для добавления и извлечения элементов.
3. **Эффективность**: Операции добавления и удаления элементов из начала очереди выполняются за константное время.

Некоторые из основных методов очереди (std::queue):

* push() - Добавляет элемент в конец очереди.
* pop() - Удаляет элемент из начала очереди.
* front() - Возвращает элемент в начале очереди (первый элемент).
* back() - Возвращает элемент в конце очереди (последний элемент).
* size() - Возвращает текущий размер очереди.
* empty() - Проверяет, пуста ли очередь.

Пример использования std::queue можно увидеть на рисунке 4.

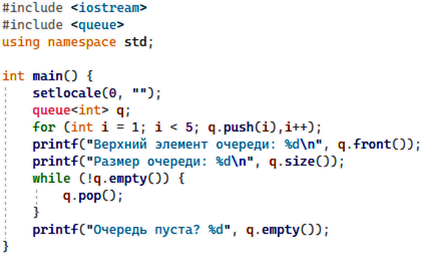


Рисунок 4 - Пример использования std::queue

**Стек**

Стек в C++, представленный библиотечным шаблоном std::stack, является абстракцией структуры данных, соответствующей концепции LIFO (Last In First Out). Это подразумевает, что элемент, добавленный последним, будет удален первым. Такая организация стека делает его полезным при работе с данными в обратном порядке их поступления.

Преимущества стека:

1. **Соблюдение порядка элементов**: Стек гарантирует, что последний добавленный элемент будет первым, который выйдет из стека.
2. **Простота использования**: Стек предоставляет простой интерфейс для добавления и извлечения элементов.
3. **Эффективность**: Операции добавления и удаления элементов из вершины стека выполняются за константное время.

Некоторые из основных методов стека (std::stack):

* push() - Добавляет элемент на вершину стека.
* pop() - Удаляет элемент с вершины стека.
* top() - Возвращает элемент на вершине стека (последний добавленный элемент).
* size() - Возвращает текущий размер стека.
* empty() - Проверяет, пуст ли стек.

Пример использования std::stack можно увидеть на рисунке 5.



Рисунок 5 - Пример использования std::stack

**Множество уникальных элементов**

1) std::set — это ассоциативный контейнер, который хранит уникальные элементы в отсортированном порядке. Он основан на структуре данных бинарное дерево поиска, что позволяет ему выполнять операции вставки, удаления и поиска за логарифмическое время.

* Упорядоченное множество: элементы хранятся в отсортированном порядке (по умолчанию по возрастанию).
* Уникальность: гарантирует, что каждый элемент уникален.
* Преимущества: идеально подходит для случаев, когда нужно хранить уникальные элементы в упорядоченном виде.

Пример использования std::set представлен на рисунке 6.

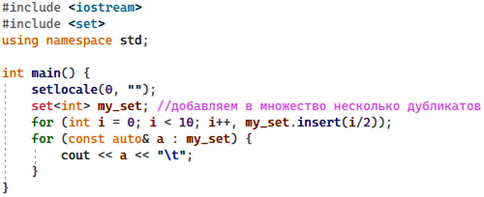


Рисунок 6 - Пример использования std::set

2) std::unordered\_set — это еще один ассоциативный контейнер, но он использует хеш-таблицу для хранения элементов. Это позволяет ему быстрее выполнять основные операции, но с меньшей гарантией порядка элементов. Однако, он может оказаться более эффективным в случаях, когда порядок элементов не важен.

* Неупорядоченное множество: элементы не хранятся в определенном порядке, но доступ к элементам быстрее.
* Уникальность: гарантирует, что каждый элемент уникален.
* Преимущества: быстрый поиск элементов (амортизированное константное время).

Пример использования std::unordered\_set можно увидеть на рисунке 7.

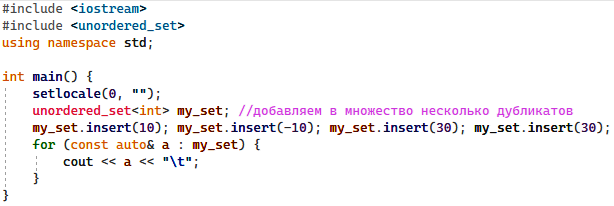


Рисунок 7 - Пример использования std::unordered\_set

3) std::multiset отличается от std::set тем, что он позволяет хранить несколько экземпляров одного и того же значения. Это может быть полезно, если вы хотите сохранить список уникальных значений, но также хотите иметь возможность хранить дубликаты.

* Упорядоченное множество с дубликатами: элементы хранятся в отсортированном порядке, но могут содержать дубликаты.
* Уникальность: не гарантирует уникальность элементов, поэтому может содержать несколько одинаковых элементов.
* Преимущества: подходит, когда нужно хранить упорядоченные элементы с возможностью дублирования.

Пример использования std::multiset представлен на рисунке 8.

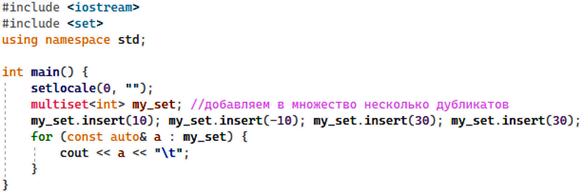


Рисунок 8 - Пример использования std::multiset

**Ассоциативный массив**

std::map, std::unordered\_map и std::multimap являются контейнерами в библиотеке STL языка C++. Они представляют собой ассоциативные массивы (словари), которые хранят пары “ключ-значение”.

1) std::map является упорядоченным ассоциативным контейнером. Он хранит пары “ключ-значение” в отсортированном виде по ключу. Для реализации своей структуры он использует красно-чёрные деревья. Операции вставки, удаления, поиска имеют логарифмическую сложность.

* Упорядоченный ассоциативный массив: элементы хранятся в отсортированном порядке ключей (по умолчанию по возрастанию ключей).
* Уникальность ключей: гарантирует, что каждый ключ в мапе уникален.
* Преимущества: идеально подходит для случаев, когда нужно хранить уникальные ключи и ассоциированные с ними значения в упорядоченном виде.

Пример использования std::map можно увидеть на рисунке 9.

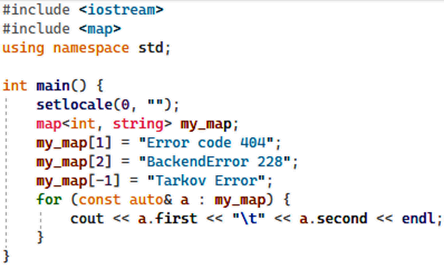


Рисунок 9 - Пример использования std::multiset

2) В отличие от std::map, std::unordered\_map использует для хранения хэш-таблицы. Это обеспечивает лучшую производительность, но порядок элементов не гарантируется. Некоторые операции могут выполняться медленнее, чем для std::map.

* Неупорядоченный ассоциативный массив: элементы не хранятся в определенном порядке, но доступ к элементам быстрее.
* Уникальность ключей: гарантирует, что каждый ключ в мапе уникален.
* Преимущества: быстрый поиск элементов (амортизированное константное время).

Пример использования std::unordered\_map представлен на рисунке 10.

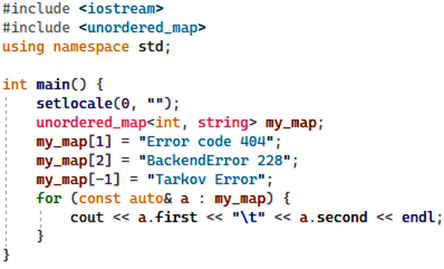


Рисунок 10 - Пример использования std::unordered\_map

std::multimap позволяет сохранять несколько элементов с одинаковыми ключами. Это может пригодиться, если нужно хранить коллекцию объектов с уникальными ключами, но при этом есть необходимость сохранять дубликаты.

* Упорядоченный ассоциативный массив с дубликатами ключей: Элементы хранятся в отсортированном порядке ключей, и ключи могут дублироваться.
* Не гарантирует уникальность ключей.
* Преимущества: подходит, когда нужно хранить упорядоченные ключи с возможностью дублирования.

Пример использования std::multimap представлен на рисунке 11.

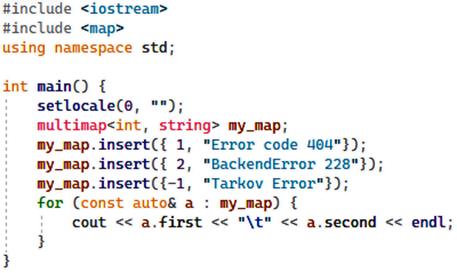


Рисунок 11 - Пример использования std::multimap

**Заключение**

В процессе изучения стандартной библиотеки STL в C++ я открыл для себя множество полезных контейнеров и алгоритмов, которые значительно упрощают работу с данными при программировании. Мне удалось познакомиться с такими контейнерами как векторы, списки, очереди, стеки, множества, каждый из которых предлагает разные способы организации данных.

Помимо этого, я изучил и ассоциативные контейнеры - мапы и множества - позволяющие связывать ключи с соответствующими значениями и осуществлять эффективный поиск и доступ к данным.

Также мне удалось ознакомиться с целым рядом алгоритмов, предоставляемых STL - сортировка, поиск, различные операции с множествами и т.д. - которые можно применять к различным контейнерам при обработке данных.

Использование STL существенно упрощает процесс кодирования и повышает его эффективность: многие структуры данных и алгоритмы уже имеются в готовом виде, так что нам не приходится тратить время и усилия на их разработку или усовершенствование. Это помогает сэкономить немало времени и сил при разработке и оптимизации программ.

**Решение задач**

Задача 1: Сортировка массива

Описание задачи: Дан массив целых чисел. Необходимо отсортировать его в порядке возрастания.

Решение этой задачи представлено на рисунке 12.



Рисунок 12 – Решение задачи 1

Задача 2: Сумма четных чисел

Описание задачи: Дан вектор целых чисел. Найти сумму всех нечетных чисел в векторе.

Решение этой задачи показано на рисунке 13.

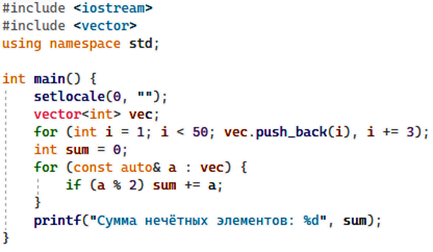


Рисунок 13 – Решение задачи 2

Задача 3: Описание кода ошибки

Описание задачи: Получить описание ошибки по её коду, полученному из ввода пользователя.

Решение этой задачи можно увидеть на рисунке 14.

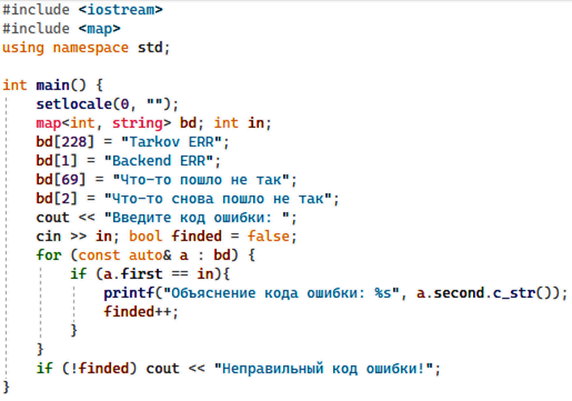


Рисунок 14 – Решение задачи 3

**Список использованных источников**

<https://tproger.ru/articles/stl-cpp>

<https://ru.cppreference.com/w/>

<http://www.codenet.ru/progr/cpp/stl/using-stl.php>

<https://cplusplus.com/doc/>

<https://www.bestprog.net/ru/2021/09/26/c-standard-template-library-general-concepts-ru/>

<https://en.wikipedia.org/wiki/Standard_Template_Library>

<https://devdocs.io/cpp/>

**МИНОБРНАУКИ РОССИИ**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего образования**

**«Чувашский государственный университет имени И.Н. Ульянова»**

**(ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»)**

**Факультет информатики и вычислительной техники**

**Кафедра вычислительной техники**

РАБОЧИЙ ГРАФИК (ПЛАН)

ПРОВЕДЕНИЯ УЧЕБНОЙ ПРАКТИКИ

(ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКИ)

на базе ***кафедры вычислительной техники факультета информатики и вычислительной техники ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»***

***Ярдыкова Эдуарда Евгеньевича, ИВТ-41-22***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

ФИО обучающегося, группа

*09.03.01 Информатика и вычислительная техника, направленность (профиль) Программное обеспечение средств вычислительной техники и автоматизированных систем*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

направление подготовки /направленность (профиль)

| №  п/п | Разделы (этапы)  практики | Виды работ на практике, включая самостоятельную работу студентов | Трудоемкость,  час | Дата |
| --- | --- | --- | --- | --- |
| 1. | Организация практики, подготовительный этап | Оформление на практику, прохождение инструктажа по охране труда, технике безопасности, пожарной безопасности, а также ознакомление с правилами внутреннего трудового распорядка организации, предоставляющей место для прохождения практики | 9 | 26.06.2023 |
| 2. | Производственный этап | Обучение и работа на рабочем месте в качестве стажера-практиканта в соответствии с индивидуальным заданием | 72 | 26.06.2023-  05.07.2023 |
| 3. | Подготовка отчета | Сбор, обработка и систематизация фактического и литературного материала | 24 | 06.07.2023-  08.07.2023 |
| 4. | Защита отчета | Получение отзыва на рабочем месте  Публичная защита отчета | 3 | 20.07.2023 |
|  | ИТОГО |  | 108 |  |

Руководитель практики от кафедры \_\_\_\_\_\_\_\_\_\_\_/ Юрьева Е.В.

Дата выдачи графика 26 июня 2023 года

Согласовано:

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_/ Юрьева Е.В.

Дата согласования 26 июня 2023 года

ДНЕВНИК

ПРОХОЖДЕНИЯ УЧЕБНОЙ ПРАКТИКИ (ОЗНАКОМИТНЛЬНОЙ ПРАКТИКИ)

на базе ***кафедры вычислительной техники факультета информатики и вычислительной техники ФГБОУ ВО «ЧГУ им. И.Н. Ульянова»***

***Ярдыкова Эдуарда Евгеньевича, ИВТ-41-22***

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

ФИО студента-практиканта, группа

*09.03.01 Информатика и вычислительная техника, направленность (профиль) Программное обеспечение средств вычислительной техники и автоматизированных систем*

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

направление подготовки /направленность (профиль)

| №  п/п | Разделы (этапы)  практики | Виды работ на практике, включая самостоятельную работу студентов | Трудоемкость,  час | Дата |
| --- | --- | --- | --- | --- |
| 1. | Организация  практики, подготовительный  этап | Оформление на практику, прохождение инструктажа по охране труда, технике безопасности, пожарной безопасности, а также ознакомление с правилами внутреннего трудового распорядка организации, предоставляющей место для прохождения практики | 9 | 26.06.2023 |
| 2. | Производственный  этап | Обучение и работа на рабочем месте в качестве стажера-практиканта в соответствии с индивидуальным заданием: | 72 |  |
| Начало изучения библиотеки STL. Узнал из чего состоит стандартная библиотека.  Составил план дальнейшего изучения этой библиотеки. | 6 | 26.06.2023 |
| Начал изучать контейнеры, в частности вектор. Узнал, как устроен вектор; как добавлять, удалять и обращаться к элементам вектора. Закрепил полученные знания решением задач. | 9 | 27.06.2023 |
| Продолжил изучение контейнеров. Понял структуру двусвязного списка, а также смог реализовать свой двусвязный список на основе структуры. | 9 | 28.06.2023 |
| Узнал о таком контейнере, как дек. Понял, как им пользоваться и для чего он нужен. | 9 | 29.06.2023 |
| Изучил очередь. Понял, в каких задач следует использовать именно очередь для большей оптимизации кода. Закрепил материал решением задач. | 9 | 30.07.2023 |
| Познакомился с таким контейнером, как стек. Изучил его особенности, его методы, смог реализовать свой собственный стек. Решил несколько задач для закрепления. | 9 | 01.07.2023 |
| Узнал, что есть как упорядоченное множество уникальных элементов, так и неупорядоченное. | 9 | 03.07.2023 |
| Познакомился с ассоциативным массивом. Узнал о его преимуществах и ситуаций, в которых он необходим. Закрепил знания решением задач. | 9 | 04.07.2023 |
| Закончил изучение контейнеров, закрепил знания решением задач, в которых необходимы изученные мною контейнеры. Лучше изучил остальные компоненты библиотеки для более глубокого понимания всех контейнеров. | 9 | 05.07.2023 |
| 3. | Подготовка отчета | Сбор, обработка и систематизация фактического и литературного материала | 24 | 06.07.2023-  08.07.2023 |
| 4. | Защита отчета | Получение отзыва на рабочем месте  Публичная защита отчета | 3 | 08.07.2023 |
|  | ИТОГО |  | 108 |  |

Студент практикант \_\_\_\_\_\_\_\_\_\_\_\_/Ярдыков Э.Е.

Руководитель практики от профильной организации \_\_\_\_\_\_\_\_\_/Юрьева Е.В.

Дата составления 08 июля 2023 года