

Московский авиационный институт
(национальный исследовательский университет)
Институт № 8 «Компьютерные науки и прикладная математика»

Практические работы
по курсам «Алгоритмы и структура данных» и
«Практикум программирования»
II семестр

Выполнила студент группы М8О-104Б-23

Чечина Лилия Алексеевна

Преподаватель: Аносова Наталья Павловна

Дата:

Оценка:

Москва 2024

СОДЕРЖАНИЕ

Введение.....	3
Задание VI. Обработка последовательной файловой структуры.....	4
Задание VIII. Линейные списки.....	9
Задание IX. Сортировка и поиск.....	14
Лабораторная работа №21.....	20
Лабораторная работа №23.....	22
Лабораторная работа №24.....	29
Лабораторная работа №26.....	33
Заключение.....	37
Список используемых источников.....	38

ВВЕДЕНИЕ

Цель практикума:

Состоит в том, чтобы научиться программировать, понимать принципы работы алгоритмов сортировки и поиска, уметь разрабатывать базы данных, эффективно использовать различные типы структур данных для решения задач различной сложности, научиться работе с Bash.

Задача курсовой работы:

1. Освоение различных методов сортировки данных (например, сортировка пузырьком, сортировка слиянием, быстрая сортировка) для эффективной работы с массивами и структурами данных.
2. Изучение алгоритмов поиска элемента в массиве (например, линейный поиск и бинарный поиск) для эффективного доступа к данным.
3. Разработка баз данных и использование структур данных (например, очередь, стек, дек) для управления информацией и решения задач.
4. Понимание различных типов структур данных, таких как массивы, связанные списки, деревья и т. д., и использование их для организации и хранения данных.

Задание VI. Обработка последовательной файловой структуры

Теория:

Бинарный файл - это файл, содержимое которого представлено в двоичном формате, то есть данные в таком файле хранятся в виде нулей и единиц. Бинарные файлы используются для хранения информации, которая не должна быть человекочитаемой, а также для хранения информации в виде последовательности битов.

Для работы с бинарными файлами в языке программирования C используются функции стандартной библиотеки <stdio.h>. Для открытия бинарного файла используется функция `fopen()` с указанием режима "rb" (для чтения) или "wb" (для записи).

Пример открытия бинарного файла для чтения:

```
FILE *file = fopen("file.bin", "rb");
if (file == NULL) {
    printf("Ошибка открытия файла\n");
    return 1;
}
```

Для чтения из бинарного файла используются функции `fread()` и `fwrite()`:

// Чтение 10 байт из файла

```
char buffer[10];
fread(buffer, sizeof(char), 10, file);
```

Для записи в бинарный файл используется функция `fwrite()`:

// Запись данных в файл

```
char data[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

```
fwrite(data, sizeof(char), 10, file);
```

После окончания работы с бинарным файлом его необходимо закрыть с помощью функции `fclose()`:

```
fclose(file);
```

- 1) `setfill("")` - чем заполняется пустое пространство
- 2) `setw(число)` - сколько должен быть вывод по ширине
- 3) `map <тип данных ключа, тип данных значения> название`
- 4) `for (int j: set){ }` -- проходка по эл-ам множества

Постановка задачи:

Разработать последовательную структуру данных для представления простейшей базы данных на файлах в СП Си в соответствии с заданным вариантом. Составить программу генерации внешнего нетекстового файла заданной структуры, содержащего представительный набор записей (15-20). Распечатать содержимое сгенерированного файла в виде таблицы и выполнить над ним заданное действие для 2–3 значений параметров запроса *p* и распечатать результат.

Действие по выборке данных из файла оформить в виде *отдельной программы* с параметрами запроса, вводимыми из стандартного входного текстового файла, или получаемыми из командной строки UNIX. Второй способ задания параметров обязателен для работ, оцениваемых на хорошо и отлично. Параметры задаются с помощью ключей `-f` (распечатка файла) или `-p <parameter>` (параметры конкретного варианта задания). Получение параметров из командной строки производится с помощью стандартных библиотечных функций `argc` и `argv`.

Структуры данных и константы, совместно используемые программами, следует вынести в отдельный заголовочный файл.

В процессе отладки и тестирования рекомендуется использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода. Сгенерированные и отформатированные тестовые данные необходимо заранее поместить в текстовые файлы и распечатывать при протоколировании. Рекомендуется подобрать реальные или правдоподобные тестовые данные. Число наборов тестовых данных должно быть не менее трёх. Имя файла с бинарными данными является обязательным параметром второй программы.

Отчёт должен содержать оценку пространственной и временной сложности использованного алгоритма. В состав отчета также рекомендуется включить графическую иллюстрацию структуры файла и запроса на выборку.

12 – 21. Информация об успеваемости студентов данной группы по всем предметам: фамилия, инициалы, пол, номер группы, отметки по экзаменам и зачетам.

20.* Выяснить, в какой группе учится максимальное число студенток с максимальным на курсе средним баллом.

Идея решения:

1) Считываем входные данные с текстового файла в массив с элементами типа `Student`.

Структура `Student` хранит фамилию, инициалы, пол, номер группы, оценки по экзаменам и зачётам.

2) Создаём и заносим в бинарный файл данные наших переменных.

Выводим таблицу из бинарного файла.

- 3) Для нахождения группы с максимальным количеством студентов женского пола с максимальным средним баллом, проходимся по всем элементам базы данных циклом for в поиске максимального среднего балла среди девушек, а также записываем номера их групп в множество. Максимальный средний балл среди студенток найден.
- 4) Проходимся по элементам массива students, избирая девушек с одной группы с максимальным средним баллом и записывая их количество в переменную count.
- 5) Далее — создаём словарь, где ключи — это номера групп, а количество студенток с максимальным средним баллом явл. значением.
- 6) Проходимся по словарю, если значение ключа \geq пред.максимуму, то добавляем в array answer.
- 7) Выводим ответ

Код программы:

```
#include <bits/stdc++.h>
using namespace std;

struct Student {
    string surname; // Фамилия
    string initials; // Инициалы
    char gender; // Пол
    int groupNumber; // Номер группы
    int examGradesMathAnalys; // Оценка по экзамену математики
    int creditGradesEconomy; // Оценка по зачёту по экономике
    int creditGradesLAAG; // Оценка по зачёту по ЛААГ
    int examGradesDM; // Оценка по экзамену дискретной математики
    int examCompScienc; // Оценки по экзамену информатики
};

int main(){
    const int arraySize = 20;
    Student students[arraySize];
    ifstream file("Datebase.txt"); // открываем файл для чтения
    if (!file) {
        cerr << "Error opening file!" << endl;
        return 1;
    }
```

```

    }
    for (int i = 0; i < arraySize; i++) {
        file >> students[i].surname >> students[i].initials >> students[i].gender >>
students[i].groupNumber
        >> students[i].examGradesMathAnalys >> students[i].creditGradesEconomy >>
students[i].creditGradesLAAG
        >> students[i].examGradesDM >> students[i].examCompSience;
    }
    file.close();
    int max_sr_Grade = 0;
    set <int> group_set;
    for (int i = 0; i < arraySize; i++) {
        if (students[i].gender == 'F'){
            group_set.insert(students[i].groupNumber);
            max_sr_Grade = max(max_sr_Grade, students[i].examGradesMathAnalys +
students[i].creditGradesEconomy + students[i].creditGradesLAAG + students[i].examGradesDM +
students[i].examCompSience);
        }
    }
    // Проходимся по группам
    int count = 0;
    map <int, int> the_best_girl_group;
    for (int j: group_set){
        for (int i = 0; i < arraySize; i++) {
            if (students[i].groupNumber == j){
                if (students[i].gender == 'F'){
                    if (max_sr_Grade == (students[i].examGradesMathAnalys +
students[i].creditGradesEconomy + students[i].creditGradesLAAG + students[i].examGradesDM +
students[i].examCompSience)){
                        count += 1;
                    }
                }
            }
        }
        the_best_girl_group[j] = count;
        count = 0;
    }
    FILE* F = fopen("students.bin", "wb");
    fwrite(students, sizeof(Student), arraySize, F);
    fclose(F);
    F = fopen("students.bin", "rb");
    Student students2[arraySize];
    fread(students2, sizeof(Student), arraySize, F);
    fclose(F);
    cout << "  Family Initials Gender  Group ExamMathAnalys creditEconomy ExamLAAG ExamDM
ExamKT" << "\n";
    for (int i = 0; i < arraySize; i++) {

```

```

    cout << setfill(' ') << setw(10) << students2[i].surname << " " << setw(8) << students2[i].initials
<< " " << setw(6) << students2[i].gender << " " << setw(6) << students2[i].groupNumber
    << " " << setw(14) << students2[i].examGradesMathAnalys << " " << setw(13) <<
students2[i].creditGradesEconomy << " " << setw(8) << students2[i].creditGradesLAAG
    << " " << setw(6) << students2[i].examGradesDM << " " << setw(6) <<
students2[i].examCompScience << "\n";
}
// Вывод словаря
for (auto it = the_best_girl_group.begin(); it!= the_best_girl_group.end(); it++) {
    cout << it->first << ": " << it->second << endl;
}
}

/*
1) setfill("") - чем заполняется пустое пространство
setw(число) - сколько должен быть вывод по ширине
2) map <тип данных ключа, тип данных значения> название
3) for (int j: set){} -- проходка по эл-ам множества
*/

```

Тесты:

Входные данные:

Family	Initials	Gender	Group	ExamMathAnalys	creditEconomy	ExamLAAG	ExamDM	ExamKT
Antonov	A.A.	M	101	4	5	4	4	3
Petrova	E.G.	F	102	3	3	4	3	5
Sidorov	I.P.	M	101	4	3	5	4	4
Ivanova	T.V.	F	103	5	5	4	3	5
Smirnov	D.M.	M	102	3	4	5	4	3
Kozlova	A.I.	F	101	4	3	3	4	5
Morozov	P.S.	M	102	5	5	3	4	4
Grigoryeva	D.R.	F	103	3	5	5	3	4
Nikolaev	K.A.	M	101	4	3	3	3	4
Vasilieva	E.L.	F	102	5	3	5	4	5
Pavlova	Y.P.	F	101	4	4	4	5	5
Alexandrov	D.V.	M	103	3	4	5	4	5
Sergeeva	N.E.	F	102	4	4	4	4	3
Orlov	S.M.	M	103	5	5	5	4	3
Zakharova	E.A.	F	101	3	5	4	3	3
Kuznetsov	I.N.	M	102	4	3	3	3	4
Antonova	T.S.	F	103	5	3	3	4	3
Zhukov	Y.I.	M	101	4	5	4	5	4
Sokolova	A.O.	F	102	5	3	3	3	3
Ignatiev	P.V.	M	101	3	4	3	3	3

Выходные данные:


```
101: 1
102: 1
103: 1
```

Выводы:

В ходе выполнения задания из практикума я ознакомилась с основными принципами работы с данными, их организацией и хранением. Задание позволило мне освоить навыки работы с базами данных, структурами и бинарными файлами, а также понять их важность и применение в реальной разработке программного обеспечения. Благодаря выполнению этого задания я укрепила свои знания в области баз данных и научилась эффективно работать с файловой системой компьютера. Это задание помогло улучшить мои навыки программирования.

Задание VIII. Линейные списки

Теория:

Очередь (queue) - это абстрактная структура данных, которая представляет собой упорядоченную коллекцию элементов, где каждый элемент имеет уникальный идентификатор и может быть добавлен или удален только из двух концов: переднего (front) и заднего (rear). Очередь поддерживает две основные операции: вставку элемента в конец очереди (enqueue) и извлечение элемента из начала очереди (dequeue).

Двусторонний линейный список (doubly linked list) - это разновидность связного списка, которая состоит из узлов, каждый из которых содержит два указателя: один на предыдущий узел и другой на следующий узел. Это позволяет перемещаться в обоих направлениях по списку, что делает его более гибким, чем односвязный список. Двусторонний линейный список поддерживает операции вставки и удаления элементов в любой позиции списка за константное время.

Постановка задачи:

Составить и отладить программу на языке Си для обработки линейного списка заданной организации с отображением списка на динамические структуры (группы 1, 2, 3, 8) или на массив (только с индексным доступом, без применения ссылок и указателей, для групп 4, 5, 6, 7). Навигацию по списку следует реализовать с применением итераторов. Предусмотреть выполнение одного нестандартного и четырех стандартных действий:

1. Печать списка.
2. Вставка нового элемента в список.
3. Удаление элемента из списка.
4. Подсчет длины списка.

Двусторонний линейный список с нестандартным действием: удаление каждого k-го элемента списка

Идея решения:

1) Реализовываем структуру данных Doubly Linked List, которая имеет следующие методы:

- `printList()`: печатает элементы списка
- `addElement(int data)`: добавляет элемент в конец списка
- `removeElement(int data)`: удаляет первое вхождение элемента из списка
- `getLength()`: возвращает количество элементов в списке
- `deleteEveryKthNode(int k)`: удаляет каждый k-й узел в списке, начиная с первого узла (головы)

В функции `main` показываем, как использовать класс `Doubly Linked List`, добавляя элементы, удаляя элементы, печатая список и удаляя каждый k-й узел.

Код программы:

```
#include <bits/stdc++.h>
using namespace std;

class Node { // класс-узелок: компонент моего лин. списка
public:
    int data;
    Node* next;
    Node* prev;

    Node(int data) : data(data), next(nullptr), prev(nullptr) {} // инициализируем по умолчанию
};

class DoublyLinkedList { // класс двустороннего лин списка
private:
    Node* head;
    Node* tail;
    int size;

public:
    DoublyLinkedList() : head(nullptr), tail(nullptr), size(0) {}

    ~DoublyLinkedList() {
        while (head != nullptr) {
```

```

    Node* temp = head;
    head = head->next;
    delete temp;
}
}

void printList() {
    Node* current = head;
    while (current != nullptr) {
        cout << current->data << " - ";
        current = current->next;
    }
    // cout << "NULL" << endl;
}

void addElement(int data) {
    Node* newNode = new Node(data);
    if (head == nullptr) {
        head = newNode;
        tail = newNode;
    } else {
        newNode->prev = tail;
        tail->next = newNode;
        tail = newNode;
    }
    size++;
}

bool removeElement(int data) {
    Node* current = head;
    while (current != nullptr) {
        if (current->data == data) {
            if (current->prev != nullptr) {
                current->prev->next = current->next;
            } else {
                head = current->next;
            }
            if (current->next != nullptr) {
                current->next->prev = current->prev;
            } else {
                tail = current->prev;
            }
            delete current;
            size--;
            return true;
        }
        current = current->next;
    }
}

```

```

    }
    return false;
}

int getLength() {
    return size;
}

void deleteEveryKthNode(int k) {
    if (k <= 0) {
        return;
    }

    Node* current = head;
    int count = 1;

    while (current != nullptr) {
        if (count % k == 0) {
            Node* temp = current;
            current = current->next;
            if (temp->prev != nullptr) {
                temp->prev->next = temp->next;
            } else {
                head = temp->next;
            }
            if (temp->next != nullptr) {
                temp->next->prev = temp->prev;
            } else {
                tail = temp->prev;
            }
            delete temp;
        } else {
            current = current->next;
        }
        count++;
    }
}

};

int main() {
    DoublyLinkedList list;

    list.addElement(1);
    list.addElement(2);
    list.addElement(3);
    list.addElement(4);
    list.addElement(5);
    list.printList();
}

```

```

cout << "Length of the list: " << list.getLength() << endl;

cout << "Remove element 3" << endl;
list.removeElement(3);
list.printList();

cout << "Length of the list: " << list.getLength() << endl;
cout << "Remove every k=2 el" << "\n";
list.deleteEveryKthNode(2);
list.printList();
cout << "Push el=27" << "\n";
list.addElement(27);
list.printList();
}

```

Тесты:

Входные данные:

```
1 - 2 - 3 - 4 - 5 - Length of the list: 5
```

Выходные данные:

```

Remove element 3
1 - 2 - 4 - 5 - Length of the list: 4
Remove every k=2 el
1 - 4 - Push el=27
1 - 4 - 27 -

```

Выводы:

В ходе выполнения задания, которое требовало использования двустороннего линейного списка, я успешно реализовала данную структуру самостоятельно с помощью стандартного набора действий, таких как добавление элемента, удаление элемента, поиск элемента и т.д. Помимо стандартных действий я добавил в реализацию списка одно нестандартное действие: удаление каждого k-го элемента.

В результате выполнения этого задания я поняла принципы работы двустороннего линейного списка и научилась эффективно работать со

структурами данных. Также я получила опыт в программировании и улучшила свои навыки разработки алгоритмов.

Задание IX. Сортировка и поиск

Теория:

Алгоритм бинарного поиска состоит из следующих шагов:

- 1) Находим середину массива.
- 2) Сравниваем искомое значение с элементом, находящимся в середине массива.
- 3) Если искомое значение равно элементу, то возвращаем его индекс.
- 4) Если искомое значение меньше элемента, то повторяем алгоритм для левой половины массива.
- 5) Если искомое значение больше элемента, то повторяем алгоритм для правой половины массива.
- 6) Если искомое значение не найдено, то возвращаем -1 или другое значение, указывающее на отсутствие элемента в массиве.

Бинарный поиск эффективен для больших массивов, поскольку на каждой итерации он уменьшает объем данных вдвое.

Постановка задачи:

Составить программу на языке Си с использованием процедур и функций для сортировки таблицы заданным методом и двоичного поиска по ключу в таблице.

Программа должна **вводить** значения элементов неупорядоченной таблицы и проверять работу процедуры сортировки в трех случаях: (1) элементы таблицы с самого начала упорядочены; (2) элементы таблицы расставлены в обратном порядке; (3) элементы таблицы не упорядочены. В последнем случае можно использовать встроенные процедуры генерации псевдослучайных чисел.

Для каждого вызова процедуры сортировки необходимо печатать исходное состояние таблицы и результаты сортировки. После выполнения сортировки программа должна вводить ключи и для каждого из них выполнять поиск в упорядоченной таблице с помощью процедуры двоичного поиска и печатать найденные элементы, если они присутствуют в таблице.

В процессе отладки и тестирования рекомендуется использовать команды обработки текстовых файлов ОС UNIX и переадресацию ввода-вывода. Тестовые данные необходимо заранее поместить в текстовые файлы.

В качестве текста для записей таблицы взять фрагмент стихотворения (группы 3-5), прозы (группы 1, 2) или изображение ASCII-графики (группы 6-8). Каждый элемент таблицы, содержащий ключ и текст записи, распечатывать в отдельной строке.

Идея решения:

Комплексное число представим в виде двух частей: действительной и мнимой.

Каждому комплексному числу соответствует строка. Если отсортировать комплексные числа, то

строки встанут в нужный порядок и получится рисунок.

Пирамидальная сортировка с просеиванием:

- 1) Из данного массива строим кучу
- 2) Берём корень полученной кучи и ставим его в конец нового массива
- 3) Удаляем корень и перестраиваем кучу. И так далее...

Программа выводит массив до сортировки и после.

Также используется бинарный поиск, чтобы найти введенное пользователем комплексное число

Если пользователь вводит "0 0", то выполнение программы завершается.

Код программы:

```
#include <bits/stdc++.h>

using namespace std;

const int MAX_SIZE = 1000; //макс кол-во компл. чисел, которые могут быть обработаны

struct Complex { // Структура комплексного числа
    double real; // действительная часть
    double imag; // мнимая часть
};

int compare_complex(const Complex &a, const Complex &b) { // f-ия сравнения комплексных чисел
    /* сравнивает два комплексных числа на основе их величин
    (возведенных в квадрат, чтобы избежать использования sqrt).*/
    double mod_a_squared = a.real * a.real + a.imag * a.imag;
    double mod_b_squared = b.real * b.real + b.imag * b.imag;

    if (mod_a_squared < mod_b_squared) return -1;
    if (mod_a_squared > mod_b_squared) return 1;
    return 0;
}

void heapify(Complex arr[], string lines[], int n, int i) { /* вспомогательная функция для сортировки кучи, которая обеспечивает сохранение свойства кучи.*/
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && compare_complex(arr[left], arr[largest]) > 0) {
        largest = left;
    }
}
```

```

    if (right < n && compare_complex(arr[right], arr[largest]) > 0) {
        largest = right;
    }

    if (largest != i) {
        swap(arr[i], arr[largest]);
        swap(lines[i], lines[largest]);
        heapify(arr, lines, n, largest);
    }
}

void heapSort(Complex arr[], string lines[], int n) { /*реализует сортировку кучи для сортировки
массива
Complex чисел и соответствующих строк строк.*/
    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, lines, n, i);
    }

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        swap(lines[0], lines[i]);
        heapify(arr, lines, i, 0);
    }
}

int binarySearch(const Complex arr[], int n, const Complex &key) {
    /* реализует двоичный поиск для поиска Complexчисла в отсортированном массиве.*/
    int l = 0;
    int r = n - 1;
    while (l <= r) {
        int m = l + (r - l) / 2;
        int cmp = compare_complex(arr[m], key);
        if (cmp == 0) {
            return m;
        }
        if (cmp < 0) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    return -1;
}

int main() {
    ifstream file_in("data_random.txt"); // Считывание файла
    if (!file_in.is_open()) {

```



```

    cerr << "Error" << endl;
    return 1;
}

Complex complex_numbers[MAX_SIZE];
string lines[MAX_SIZE];
int num_lines = 0;

string line;
while (getline(file_in, line)) {
    istringstream iss(line);
    Complex c;
    if (!(iss >> c.real >> c.imag)) {
        cerr << "Error" << endl;
        return 1;
    }
    if (num_lines < MAX_SIZE) {
        complex_numbers[num_lines] = c;
        lines[num_lines] = line;
        num_lines++;
    }
}

file_in.close();

// Вывод входных данных
cout << "Input data:" << endl;
for (int i = 0; i < num_lines; i++) {
    cout << lines[i] << endl;
}

heapSort(complex_numbers, lines, num_lines);

ofstream file_out("output.txt"); // Запись файла
if (!file_out.is_open()) {
    cerr << "Error" << endl;
    return 1;
}

for (int i = 0; i < num_lines; i++) {
    file_out << lines[i] << endl;
}

file_out.close();

// Вывод выходных данных на экран
cout << "Sorted data: " << endl;

```

```

ifstream file_out_read("output.txt");
if (!file_out_read.is_open()) {
    cerr << "Error opening output.txt file for reading." << endl;
    return 1;
}

while (getline(file_out_read, line)) {
    cout << line << endl;
}

file_out_read.close();

Complex key;
do {
    cout << "Enter the real and imaginary parts of a complex number separated by spaces (or enter '0 0' to exit): ";
    cin >> key.real >> key.imag;

    // Если пользователь ввел 0 0, завершаем выполнение
    if (key.real == 0 && key.imag == 0) {
        break;
    }

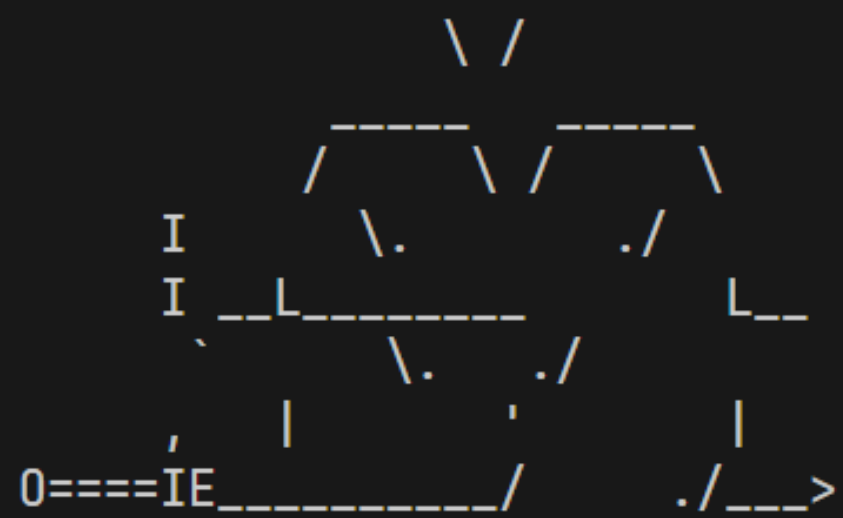
    // Поиск введенного числа
    int index = binarySearch(complex_numbers, num_lines, key);
    if (index != -1) {
        cout << "Complex number found: " << lines[index] << endl;
    } else {
        cout << "Complex number not found." << endl;
    }
} while (key.real != 0 || key.imag != 0);
}
/*
Комплексное число представим в виде двух частей: действительной и мнимой.
Каждому комплексному числу соответствует строка. Если отсортировать комплексные числа,
то
строки встанут в нужный порядок и получится рисунок.
Пирамидальная сортировка с просеиванием:
1) Из данного массива строим кучу
2) Берём корень полученной кучи и ставим его в конец нового массива
3) Удаляем корень и перестраиваем кучу. И так далее...
Программа выводит массив до сортировки и после.
Также используется бинарный поиск, чтобы найти введенное пользователем комплексное
число
Если пользователь вводит "0 0", то выполнение программы завершается.
*/

```

Тесты:

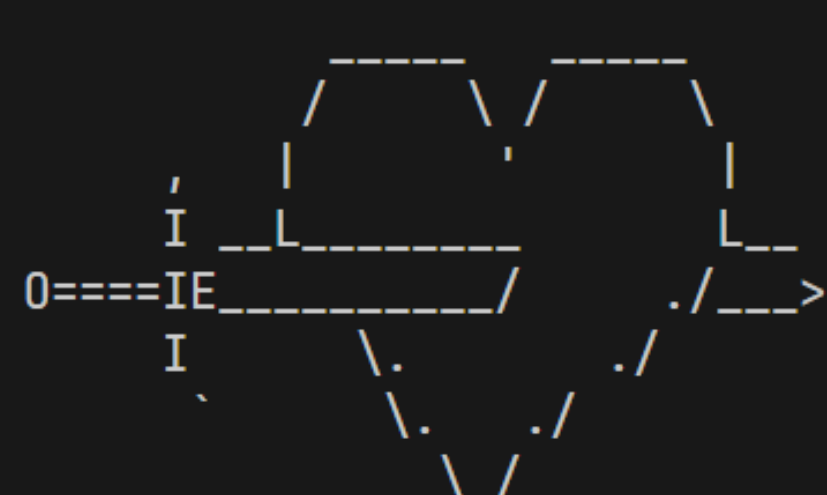
Входные данные:

```
Input data:
7.43 10.01
1.10 1.28
2.28 2.44
6.88 7.26
4.74 4.47
7.04 9.13
4.68 3.78
4.89 5.70
```



Выходные данные:

```
Sorted data:
1.10 1.28
2.28 2.44
4.68 3.78
4.74 4.47
4.89 5.70
6.88 7.26
7.04 9.13
7.43 10.01
```



```
Enter the real and imaginary parts of a complex number separated by spaces (or enter '0 0' to exit):
7.43 10.01
Complex number found: 7.43 10.01
Enter the real and imaginary parts of a complex number separated by spaces (or enter '0 0' to exit):
5.00 4.00
Complex number not found.
Enter the real and imaginary parts of a complex number separated by spaces (or enter '0 0' to exit):
0 0
```

Выводы:

В данной лабораторной работе я изучила способы представления комплексных чисел в компьютере. Также научилась сравнивать между собой комплексные числа и сортировать их с помощью пирамидальной

сортировки с просеиванием, а также находить значения с массиве с использованием бинарного поиска.

Использование пирамидальной сортировки позволяет эффективно упорядочить комплексные числа, а бинарный поиск помогает найти нужное число быстро.

Лабораторная работа №21

Теория:

`pwd` — отображение текущего рабочего каталога

`mkdir <name_of_directory>` — создать директорию

`ls` — просмотр текущей директории

`cd ..` — перейти на 1 уровень каталога выше

`nano myscript.sh` — войти в скрипт

выйти из скрипта: `ctrl+x;Y; enter`

`./myscript.sh` — запуск скрипта в данной директории

`bash script .sh /path/to/directory` — запуск скрипта в директории по указанному пути

`-e` — проверка на существование файла

`-d` — проверка на то, является ли данный файл каталогом

`echo` — команда, используемая для отображения строки текста/значений, которые передаются в качестве аргумента в кавычках “аргумент”

`$1` — позиционный параметр, передаваемый скрипту или функции

`$0` — имя текущего скрипта

Постановка задачи:

Рекурсивно обойти указанный каталог и вывести на экран полные пути поддиректорий.

Идея решения:

1. Открываем командную строку Linux.
2. Командой `ls` проверяем в какой директории мы находимся.

3. С помощью команды `mkdir` создаём директорию. Затем используем `cd`, чтоб зайти в неё. Создаём несколько поддиректорий. С помощью команды `cd ..` поднимаемся на каталог выше.
4. Создаём скрипт с помощью команды `nano` и далее вводим название файла "sh".
5. Описываем скрипт, после чего выходим из `nano`.
6. Делаем скрипт исполняемым, с помощью команды `+x ./myscript.sh`
7. Запускаем выполнение скрипта командой `bash script.sh /path/to/directory`

Код программы:

```
flowers_queen@ubuntu24:~$ ls
directory1 myscript.sh test type.
flowers_queen@ubuntu24:~$
```

```
flowers_queen@ubuntu24:~$ cd directory1
flowers_queen@ubuntu24:~/directory1$ ls
subdirectory1 subdirectory2 subdirectory3 subdirectory4 subdirectory5
flowers_queen@ubuntu24:~/directory1$ cd ..
flowers_queen@ubuntu24:~$ +x ./myscript.sh
```

```
GNU nano 6.2 myscript.sh
#!/bin/bash
#Задача: рекурсивный обход указанного каталога и вывод на экран полных путей поддиректорий

if [ "$1" = "?" ]; then
    echo "Вывод на экран полных путей поддиректорий указанного каталога"
    echo "bash $0 [каталог, в котором нужно совершить обход]"
elif [ -d $1 ]; then
    cd $1
    find -type d
else
    echo "Данной папки не существует"
fi
```

```
bash: script.sh: No such file or directory
flowers_queen@ubuntu24:~$ bash myscript.sh directory1
.
./subdirectory3
./subdirectory4
./subdirectory5
./subdirectory2
./subdirectory1
```

Тесты:

```
flowers_queen@ubuntu24:~$ cd directory1
flowers_queen@ubuntu24:~/directory1$ ls
subdirectory1 subdirectory2 subdirectory3 subdirectory4 subdirectory5
flowers_queen@ubuntu24:~/directory1$ cd ..
flowers_queen@ubuntu24:~$ +x ./myscript.sh
```

```
bash: script.sh: No such file or directory
flowers_queen@ubuntu24:~$ bash myscript.sh directory1
.
./subdirectory3
./subdirectory4
./subdirectory5
./subdirectory2
./subdirectory1
```

Вывод:

В ходе данной лабораторной работы я закрепила свои знания о Bash. Выполнила поставленную задачу, научилась писать скрипты и работать в nano.

Лабораторная работа №23

Теория:

Бинарное дерево - это структура данных, в которой каждый узел содержит не более двух потомков, называемых левым и правым ребёнком. Обычно бинарные деревья используются для хранения данных, которые имеют иерархическую структуру.

Каждый узел бинарного дерева содержит три элемента: значение, указатель на левого потомка и указатель на правого потомка. Значение узла может быть любым объектом, например, числом или строкой. Указатели на потомков могут быть пустыми, если у узла нет потомков.

Бинарные деревья могут быть разными типами, в зависимости от того, как они упорядочены. Например, бинарное дерево поиска - это бинарное дерево, в котором каждый узел больше или равен своим левым потомкам и меньше или равен своим правым потомкам. Это позволяет эффективно выполнять операции поиска, вставки и удаления.

Другой тип бинарного дерева - это сбалансированное бинарное дерево, в котором разница высоты левого и правого поддеревьев не превышает 1. Это позволяет поддерживать высоту дерева в пределах $O(\log n)$, где n - количество узлов в дереве. Примерами сбалансированных бинарных деревьев являются AVL-дерево и красно-черное дерево.

Определения

Глубиной вершины дерева называется длина пути в эту вершину из корня. **Глубиной (высотой)** дерева называется максимальная глубина его вершин. **Листом** или **терминальной вершиной** дерева называется вершина, не имеющая поддеревьев. **Степенью вершины** называется число исходящих из нее ветвей. **Степенью дерева** называется максимальная степень его вершин. **Шириной уровня** дерева называется число вершин на данной глубине. **Шириной** дерева называется максимальная ширина по всем уровням. **Подобие** деревьев отличается от **равенства** возможным несовпадением значений данных, хранящихся в узлах. **AVL-деревом** называется двоичное дерево, в котором высоты левого и правого поддеревьев отличаются не более, чем на 1. Двоичное дерево называется **В-деревом**, если в нем нет ни одного узла степени 1.

Постановка задачи:

Составить программу на языке Си для построения и обработки дерева общего вида или упорядоченного двоичного дерева, содержащего узлы типа float, int, char или enum (для групп 1,8; 3,4; 2,5; 6,7 соответственно). Основные функции работы с деревьями реализовать в виде универсальных процедур или функций. После того, как дерево создано, его обработка должна производиться в режиме текстового меню со следующими действиями:

- *добавление нового узла* (для двоичного дерева положение нового узла определяется в соответствии с требованием сохранения порядка; для дерева общего вида должен задаваться *отец* добавляемого узла. Добавляемый узел становится *самым младшим сыном*);
- *текстовая визуализация дерева* (значение каждого узла выводится в отдельной строке, с отступом, пропорциональным глубине узла, в порядке старшинства узлов);
- *удаление узла* (двоичное дерево перестраивается в соответствии с требованием сохранения целостности и порядка; для дерева общего вида удаляется все поддерево, исходящее из удаляемого узла. Должно быть предусмотрено корректное освобождение памяти);
- *вычисление значения некоторой функции от дерева* (целой или логической), в соответствии с номером варианта.

Идея решения:

Программа реализует структуру данных-дерево со следующими методами:

createNode(int data, struct Node *parent) - создает новый узел с заданными данными и родительским узлом.

addChild(struct Node *parent, int data) - добавляет новый дочерний узел с заданными данными к родительскому узлу.

deleteSubtree(struct Node *parent, struct Node *node) - удаляет поддерево, корень которого находится в данном узле, и соответствующим образом обновляет дочерний указатель родительского узла.

deleteRoot(struct Node **root) - удаляет все дерево, корневое в данном корневом узле.

printTree(struct Node *node, int depth) - печатает дерево в порядке приоритета в глубину.

findMaxDepth(struct Node *node, int depth) - находит максимальную глубину залегания дерева.

Основная функция программы создает пустое дерево, а затем позволяет пользователю взаимодействовать с ним с помощью системы меню.

Пользователь может добавлять узлы в дерево, печатать дерево, удалять поддерево, находить максимальную глубину дерева и удалять корневой узел вместе с его поддеревьями.

Чтобы добавить новый узел в дерево, пользователю предлагается ввести данные родительского узла и нового дочернего узла. Затем программа находит родительский узел и добавляет к нему новый дочерний узел.

Чтобы распечатать дерево, программа использует алгоритм поиска в глубину, чтобы обойти дерево и распечатать данные каждого узла вместе с его глубиной.

Для удаления поддерева пользователю предлагается ввести данные узла, поддерево которого должно быть удалено. Затем программа находит узел и удаляет все его поддерево.

Чтобы найти максимальную глубину дерева, программа использует рекурсивный алгоритм, который проходит по дереву и отслеживает максимальную глубину, обнаруженную на данный момент.

Для удаления корневого узла вместе с его поддеревьями программа использует функцию освобождения памяти, выделенной для всего дерева. Затем пользователю будет предложено ввести данные нового корневого узла для создания нового дерева.

Код программы:

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    struct Node* parent;
    struct Node* child;
    struct Node* neighbour;
};

Node* createNode(int data, Node* parent) {
    Node* newNode = new Node; // выделяем память
    newNode->data = data;
    newNode->parent = parent;
    newNode->child = nullptr;
    newNode->neighbour = nullptr;
    return newNode;
}

void addChild(Node* parent, int data) { // добавить потомка
    Node* newNode = createNode(data, parent);
    if (parent->child == nullptr) {
        parent->child = newNode;
    } else {
        Node* neighbour = parent->child;
        while (neighbour->neighbour != nullptr) {
            neighbour = neighbour->neighbour;
        }
    }
}
```



```

    }
    neighbour->neighbour = newNode;
}
}

void deleteSubtree(Node* parent, Node* node) {
    if (node == nullptr) return;

    deleteSubtree(node, node->child);
    deleteSubtree(node, node->neighbour);
    delete node; // освобождаем память, занятую узлом
}

void deleteRoot(Node* &root) { // удаление корня
    if (root == nullptr) return;

    deleteSubtree(nullptr, root->child);
    delete root;
    root = nullptr;
}

void printTree(Node* node, int depth) { // Вывод дерева
    if (node == nullptr) return;
    for (int i = 0; i < depth; i++) {
        cout << " ";
    }
    cout << node->data << endl;
    printTree(node->child, depth + 1);
    printTree(node->neighbour, depth);
}

int maxDepth = 0;

void findMaxDepth(Node* node, int depth) { // f-ия нахождения максимальной глубины
    if (node == nullptr) {
        return;
    }
    if (depth > maxDepth) {
        maxDepth = depth;
    }
    findMaxDepth(node->child, depth + 1);
    findMaxDepth(node->neighbour, depth);
}

Node* foundParent = nullptr;
void findParent(Node* node, int data) {
    if (node == nullptr || foundParent != nullptr) return;

```

```

    if (node->data == data) {
        foundParent = node;
        return;
    }
    findParent(node->child, data);
    findParent(node->neighbour, data);
}

Node* deleteNode = nullptr;
Node* parentNode = nullptr;
void findDeleteNode(Node* node, int data, Node* parent) {
    if (node == nullptr || deleteNode != nullptr) return;
    if (node->data == data) {
        deleteNode = node;
        parentNode = parent;
        return;
    }
    findDeleteNode(node->child, data, node);
    findDeleteNode(node->neighbour, data, parent);
}

int main() {
    Node* root = nullptr;
    int rootData;
    cout << "Enter the root value: ";
    cin >> rootData;
    root = createNode(rootData, nullptr);

    int menu;
    do {
        cout << "Menu:" << endl;
        cout << "1. Add node" << endl;
        cout << "2. Display the tree" << endl;
        cout << "3. Delete subtree" << endl;
        cout << "4. Find the depth of the maximum vertex" << endl;
        cout << "5. Remove root Node and all subtrees" << endl;
        cout << "6. Exit" << endl;
        cout << "Select an action: ";
        cin >> menu;

        switch (menu) {
            case 1:
                int data, parentData;
                cout << "Enter the new node value: ";
                cin >> data;
                cout << "Enter the parent node value: ";
                cin >> parentData;

```

```

        foundParent = nullptr;
        findParent(root, parentData);
        if (foundParent == nullptr) {
            cout << "Parent not found." << endl;
            break;
        }
        addChild(foundParent, data);
        break;

    case 2:
        cout << "Tree:" << endl;
        printTree(root, 0);
        break;

    case 3:
        int deleteData;
        cout << "Enter the node value to delete subtree: ";
        cin >> deleteData;
        deleteNode = nullptr;
        parentNode = nullptr;
        findDeleteNode(root, deleteData, nullptr);
        if (deleteNode == nullptr) {
            cout << "Node not found." << endl;
            break;
        }
        deleteSubtree(parentNode, deleteNode);
        break;

    case 4:
        maxDepth = 0;
        findMaxDepth(root, 0);
        cout << "Max depth: " << maxDepth << endl;
        break;

    case 5:
        deleteRoot(root);
        cout << "Root Node and all subtrees are removed." << endl;
        break;

    case 6:
        cout << "Exiting program." << endl;
        break;

    default:
        cout << "Invalid input, try again." << endl;
    }
} while (menu != 6);

```

```
}
```

Тесты:

```
Enter the root value: 5
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 1
Enter the new node value: 7
Enter the parent node value: 5
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 1
Enter the new node value: 9
Enter the parent node value: 5
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
```

```
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 1
Enter the new node value: 8
Enter the parent node value: 7
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 1
Enter the new node value: 10
Enter the parent node value: 9
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 1
Enter the new node value: 45
```

```

Enter the parent node value: 9
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 2
Tree:
5
  7
    8
  9
    10
    45
Menu:
1. Add node
2. Display the tree
3. Delete subtree
4. Find the depth of the maximum vertex
5. Remove root Node and all subtrees
6. Exit
Select an action: 6
Exiting program.

```

Вывод:

В ходе данной лабораторной работы мы научились работать с бинарным деревом, реализовывать функции данной структуры, а также определять максимальную глубину.

Лабораторная работа №24

Теория:

Преобразование арифметических выражений с помощью дерева - это процесс, при котором арифметическое выражение представляется в виде дерева, где каждый узел соответствует оператору или операнду выражения. Это позволяет удобно выполнять различные операции над выражением, такие как вычисление его значения, оптимизация, преобразование в другие формы и т.д.

Для построения дерева арифметического выражения обычно используются два типа узлов: узлы операторов (например, узлы для операций сложения, вычитания, умножения и деления) и узлы операндов (числовые значения или переменные).

Преобразование арифметических выражений с помощью дерева может включать в себя выполнение следующих операций:

1. Вычисление значений выражения: обход дерева снизу вверх, начиная с листьев (узлов-операндов) и вычисляя значения операторов на каждом уровне, пока не будет получен результат выражения.
2. Оптимизация выражения: преобразование дерева для упрощения и сокращения выражения, например, сведение подряд идущих операций умножения и деления, упрощение константных выражений и т.д.
3. Преобразование выражения в другую форму: например, в постфиксную или префиксную нотацию.
4. Построение дерева выражения из другого представления, например, из инфиксной записи выражения.

Преобразование арифметических выражений с использованием дерева - это мощный инструмент, который помогает удобно работать с выражениями и выполнять различные операции над ними.

Постановка задачи:

Упростить выражение $((a/b)/c)$ до $(a/(b*c))$ с помощью дерева.

Идея решения:

- 1) Создаём структуру Node, содержащую данные, указатели на правый и левый потомки.
- 2) Ф-ия slice возвращает подстроку из строки
- 3) Ф-ия buildTree, когда встречаем оператор без скобок,
создаём новый узел и рекурсивно строим левое и правое поддеревья.
- 4) Ф-ия printInOrder печатает данные в узлах дерева в порядке обхода:
сначала левое поддерево, затем узел, затем правое поддерево.
Если узел содержит оператор, то он заключается в скобки.
- 5) Ф-ия simplifyFraction упрощает выражение дроби, представленное деревом, проверяет,
содержит ли узел оператор деления с дробью в левой части.
Если это так, заменяет деление умножением и обновляет левый
и правый дочерние узлы.
- 6) В int main() выводим строку до преобразования, применяем написанные ранее функции, выводим строку после преобразования.

Код программы:

```
#include <iostream>
#include <string>
using namespace std;

struct Node {
    string data;
    Node* left;
    Node* right;

    Node(string val) : data(val), left(nullptr), right(nullptr) {}
    Node(string val, Node* l, Node* r) : data(val), left(l), right(r) {}
};

string slice(const string& s, int start, int end) { // возвращает подстроку из строки
// s между индексами start и end, исключая скобки.
    string a;
    for (int x = start; x < end; x++) {
        if (s[x] != '(' && s[x] != ')') {
            a += s[x];
        }
    }
    return a;
}

Node* buildTree(string expression) {
    Node* root = nullptr;
    bool f = false;
    int parenCount = 0;
    /* Когда встречаем оператор без скобок,
    создаём новый узел и рекурсивно строим левое и правое поддеревья.*/
    for (int i = 0; i < expression.length(); i++) {
        if (expression[i] == '/' && parenCount == 0) {
            root = new Node("/");
            root->left = buildTree(slice(expression, 0, i));
            root->right = buildTree(slice(expression, i + 1, expression.length()));
            break;
        } else if (expression[i] == '(') {
            parenCount++;
        } else if (expression[i] == ')') {
            parenCount--;
        }
    }

    if (root == nullptr) {
        root = new Node(expression);
    }
}
```

```

    }

    return root;
}

/*печатает данные в узлах дерева в порядке обхода:
сначала левое поддерево, затем узел, затем правое поддерево.
Если узел содержит оператор, то он заключается в скобки.*/
void printInOrder(Node* root) {
    if (root == nullptr) {
        return;
    }

    bool isOperator = (root->data == "/" || root->data == "*");
    if (isOperator && root->left) {
        cout << "(";
    }
    printInOrder(root->left);
    cout << root->data;
    printInOrder(root->right);
    if (isOperator && root->right) {
        cout << ")";
    }
}

/*функция упрощает выражение дроби, представленное деревом, проверяет,
содержит ли узел оператор деления с дробью в левой части.
Если это так, заменяет деление умножением и обновляет левый
и правый дочерние узлы.*/
void simplifyFraction(Node* root) {
    if (root == nullptr) return;

    if (root->data == "/" && root->left && root->left->data == "/") {
        root->right = new Node("*", root->left->right, root->right);
        root->left = root->left->left;
    }
}

int main() {
    string expression = "(a/b)/c";
    Node* root = buildTree(expression);
    cout << "Before simplification: ";
    cout << "(" << expression << ")" << "\n";
    simplifyFraction(root);
    cout << "After simplification: ";
    printInOrder(root);
    cout << "\n";
}

```



```
// Программа принимает строку в виде дробного выражения,  
// строит дерево выражения, затем выполняет упрощение многочленов  
// с использованием дерева. Программа упрощает  
// выражение ((a/b)/c) до (a/(b*c)).
```

Тесты:

```
Before simplification: a/(b/c)  
After simplification: a * c / b
```

Вывод:

В ходе данной лабораторной работы мы научились работать с деревьями и преобразовывать арифметические выражения с их помощью.

Лабораторная работа №26

Теория:

В информатике очередь — это линейная структура данных, которая следует принципу FIFO (First-In-First-Out). Это означает, что первый элемент, который добавляется в очередь, удаляется первым.

Очередь состоит из двух основных операций:

Постановка в очередь: Эта операция добавляет элемент в конец очереди.

Вывод из очереди: Эта операция удаляет элемент из передней части очереди.

Вот простая аналогия, которая поможет вам визуализировать, как работает очередь: представьте себе очередь людей, ожидающих покупки билетов на концерт. Первый человек, который пришел, также является первым, кто покупает билет, за ним следует второй человек и так далее. Именно так работает очередь.

Сортировка пузырьком - это простой алгоритм сортировки, который работает путем последовательного сравнения и обмена соседних элементов в массиве.

Алгоритм сортировки пузырьком работает следующим образом:

Начинаем с первого элемента массива и сравниваем его с соседним элементом.

Если первый элемент больше соседнего, то мы меняем их местами.

Затем переходим к следующей паре элементов и повторяем процесс.

Продолжаем этот процесс до тех пор, пока не пройдем весь массив.

После каждого прохода по массиву, наибольший элемент будет перемещен в конец массива.

Процесс повторяется, пока массив не будет отсортирован.

Сортировка пузырьком имеет сложность $O(n^2)$, что делает ее неэффективной для больших массивов данных. Однако, она проста в реализации и может быть полезна для небольших массивов данных.

Постановка задачи:

Составить и отладить модуль определений и модуль реализации по заданной схеме модуля определений для абстрактного (пользовательского) типа данных (стека, очереди, списка или дека, в зависимости от варианта задания). Составить программный модуль, сортирующий экземпляр указанного абстрактного типа данных заданным методом, используя только операции, импортированные из модуля UUDT.

Очередь, задача: сравнивать каждые 2 подряд идущие элемента, если первый больше второго,

то менять их местами, метод сортировки пузырьком.

Очередь отображается на массив.

Идея решения:

- 1) Создаём структуру очереди
- 2) Создаём и реализовываем функции для очереди: `new_Queue`, `is_Empty`, `delete_Queue`
- 3) Создаём и реализовываем функцию сортировки пузырьком `bubble_Sort`
- 4) Создаём функцию `Creation` для создания, заданной пользователем очереди
- 5) В `int main()` показываем пользователю введенную им очередь до сортировки и после, применяя функцию `bubble_Sort`, затем освобождаем память с помощью `delete_Queue`

Код программы:

```
/* Очередь, задача: сравнивать каждые 2 подряд идущие элемента, если первый больше
второго,
то менять их местами, метод сортировки пузырьком */
#include <bits/stdc++.h>
using namespace std;
```

```

struct Queue {
    int length;
    int* data;
    int start;
    int depth;
};

Queue* new_Queue(int length){
    if (length < 0){
        return nullptr; // возвращаем nullptr, если длина очереди меньше 0
    }
    Queue* queue_ = new Queue; // динамически выделяем память для структуры Queue
    queue_->length = length;
    queue_->data = nullptr;
    if (length > 0){
        queue_->data = new int[length];
        queue_->start = 0;
        queue_->depth = length - 1;
    }
    return queue_;
}

bool is_Empty(const Queue &q){
    return (q.depth == 0);
}

void delete_Queue(Queue &queue_) {
    if (queue_.length != 0) delete [] queue_.data;
    queue_.data = nullptr;
    queue_.depth = 0;
    queue_.start = 0;
    queue_.length = 0;
}

void bubble_Sort(int list[], int list_len) // f-ия для сортировки пузырьком
{
    while(list_len--)
    {
        bool swapped = false;
        for(int i = 0; i < list_len; i++)
        {
            if(list[i] > list[i + 1])
            {
                swap(list[i], list[i + 1]);
                swapped = true;
            }
        }
    }
}

```

```

        if(swapped == false)
            break;
    }
}

Queue* Creation(){
    cout << "Enter queue length: " << "\n";
    int n;
    cin >> n;
    Queue* queue_ = new_Queue(n);
    is_Empty(*queue_);
    cout << "Enter queue items separated by spaces: " << "\n";
    for (int i = 0; i < queue_>length; i++){
        cin >> queue_>data[i];
    }
    return queue_;
}

int main(){
    Queue* queue_ = Creation();
    cout << "Bubble sorted queue: " << "\n";
    bubble_Sort(queue_>data, queue_>length);
    for (int i = 0; i < queue_>length; i++){
        cout << queue_>data[i] << " ";
    }
    delete_Queue(*queue_);
    // Если бы передала queue по ссылке f-ции deleteQueue, то эта была бы ссылка на указатель
}

/*
Идея решения:
1) Создаём структуру очереди
2) Создаём и реализовываем функции для очереди: new_Queue, is_Empty, delete_Queue
3) Создаём и реализовываем функцию сортировки пузырьком bubble_Sort
4) Создаём функцию Creation для создания, заданной пользователем очереди
5) В int main() показываем пользователю введённую им очередь до сортировки и после,
применяя функцию bubble_Sort, затем освобождаем память с помощью delete_Queue
*/

```

Тесты:

```
Enter queue items separated by spaces:  
3 2 4 1 6  
Bubble sorted queue:  
1 2 3 4 6
```

Вывод:

В ходе выполнения данной лабораторной работы мы изучили сортировку пузырьком, повторили структуру данных: очередь.

Заключение

В ходе выполнения практикума по программированию я обучилась разработке собственных структур данных, различным методам сортировок, работе с бинарными файлами, работе с базами данных. Также улучшила свои навыки разработки.

Этот опыт значительно улучшил мои навыки программирования и дал возможность применить полученные знания на практике. Работа с данными структурами и алгоритмами позволила мне лучше понять основы программирования и значимость оптимизации кода. В результате я глубже поняла важность эффективности и элегантности решений.

Список используемых источников:

1. <https://en.cppreference.com/w/cpp/language/struct>
2. <https://www.geeksforgeeks.org/data-file-handling-c-cc/>
3. <https://github.com/search?q=doubly+linked+list+c%2B%2B>
4. <https://en.wikipedia.org/wiki/Heapsort>
5. https://en.wikipedia.org/wiki/Bubble_sort
6. <https://en.cppreference.com/w/cpp/container/map>
7. <https://en.cppreference.com/w/cpp/io/manip/setw>,
<https://en.cppreference.com/w/cpp/io/manip/setfill>