МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра вычислительной техники

**Курсовая работа**

**по дисциплине «программирование»**

**Тема: «Шаблон иерархической структуры данных в памяти**»

Факультет: АВТФ Преподаватель: Новицкая Ю. В.

Группа: АВТ-208

Студенты: Нестеренко М. К.

Вариант: 3.7

Новосибирск 2023

**ОГЛАВЛЕНИЕ**

[1. Описание задания 3](#_Toc122106212)

[2. Структурное описание разработки 4](#_Toc122106213)

[3. Функционально описание разработки 8](#_Toc122106214)

[4. Описание пользовательского интерфейса 20](#_Toc122106215)

[5. Описание работы программы на контрольных примерах 23](#_Toc122106216)

[6. Выводы 24](#_Toc122106217)

[7. Список используемой литературы 25](#_Toc122106218)

[8. Приложение 26](#_Toc122106219)

# Описание задания

Для заданной двухуровневой структуры данных, содержащей указатели на объекты (или сами объекты) - параметры шаблона, разработать полный набор операций (добавление, включение и извлечение по логическому номеру, сортировка, включение с сохранением порядка, загрузка и сохранение строк в бинарном файле, балансировка – выравнивание размерностей структур данных нижнего уровня). Предполагается, что операции сравнения хранимых объектов переопределены стандартным образом (в виде операций и т.д.). Программа должна использовать шаблонный класс с объектами строками и реализовывать указанные выше действия над текстом любого объема, загружаемого из файла. Программа должна реализовывать указанные выше действия. Протестировать структуру данных. Программа тестирования должна содержать меню, обеспечивающее выбор операций.

7. Шаблон структуры данных – массив указателей на заголовки списков. Элемент списка содержит указатель на объект. (При включении элемента последним в список предусмотреть ограничение длины текущего списка и переход к следующему).

# Структурное описание разработки

* Был разработан шаблонный класс – Arr. Он необходим для работы со структурой данных – массив указателей на заголовки списков. В классе Arr реализованы методы по заданию - добавление, включение и извлечение по логическому номеру, сортировка, включение с сохранением порядка, загрузка и сохранение строк в бинарном файле. Пример структуры показана на рисунке 1:

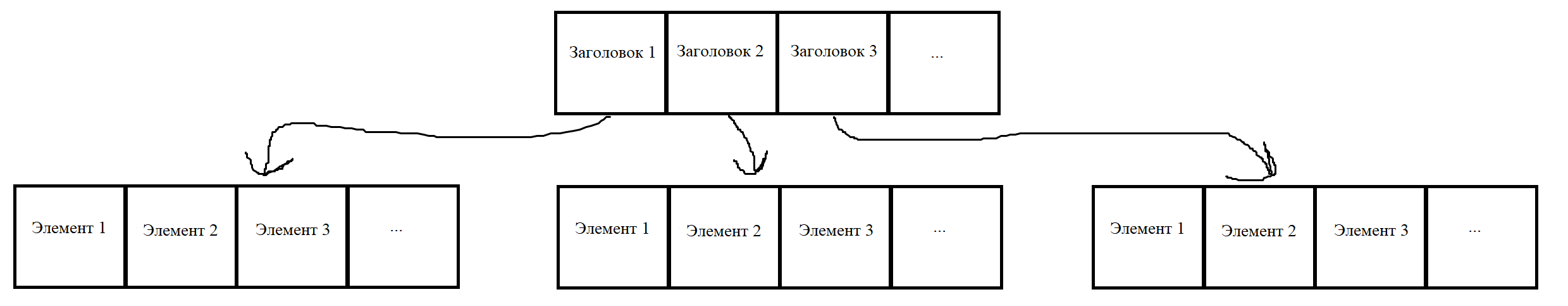


Рисунок 1 – схема структуры

В верхнем уровне хранятся заголовки на списки, размерность которого не ограничена. Заголовок списка хранит в себе массив с ограниченным размером. Элементы содержат в себе данные.

Поля класса Arr:

List<T>\*\* arr = NULL; //Динамический массив с указателями на заголовки списков

int count = 0; //Количество не нулевых указателей в массиве

int size = 1; //Размер динамического массива

В классе был реализован деструктор:

~Arr();

Для работы с классом были реализованы следующие методы:

void CreateList(int); //Создание нового списка

bool Insert(T\*); //Добавление элемента в первый найденный свободный список

bool Insert(T\*, int, int); //Добавление элемента по индексу списка и индексу значения

void DeleteItem(int, int); //Удаление элемента по индексу списка и индексу значения

void DeleteList(int); //Удаление списка из массива

void Sort(); //Общая сортировка массива

bool InsertSaveOrder(T\*); //Вставка элемента с сохранением порядка

void WriteBinFile(char\*); //Запись в бинарный файл

void ReadBinFile(char\*); //Чтение из бинарного файла

void PrintList(int); //Вывод списка по индексу

void Print(); //Вывод массива

int GetCount(); //Геттер для поля count

int GetCountList(int); //Геттер для получения поля Count в классе List

int GetSizeList(int); //Геттер для получения поля size в классе List

bool IsHaveFreeList(); //Возвращает, есть ли свободное место в списке

* Для хранения заголовков в классе Arr был разработан класс List, который хранит в себе информацию об элементах.

Поля класса:

int size = 0; //Размерность массива с элементами

int count = 0; //Количество элементов в массиве

Item<T>\*\* list = NULL; //Динамический список с указателями на элементы

Для класса были реализованы конструктор с параметрами и деструктор:

List(int); //Конструктор с параметрами, который устанавливает размер для динамического массива

~List(); //Деструктор для удаления массива

Методы класса:

void ChangeList(int);

int GetSize(); //Геттер для поля size

int GetCount(); //Геттер для поля count

bool Insert(T\*); //Вставка элемента

void SetItem(T\*, int); //Изменение значения по индексу

T\* Insert(T\*, int); //Вставка элемента по индексу

void DeleteTo(int); //Удаление элемента по индексу

T\* GetValue(int); //Геттер для получения значения из массива по индексу

Item<T>\* GetItem(int); //Геттер для получения элемента из массива по индексу

void ReadBinaryFile(std::ifstream&); //чтение из бинарного файла

void WriteBinaryFile(std::ofstream&); //запись в бинарный файла

void Print(); //Вывод

* Для хранения данных, был создан класс Item, который хранит в себе поле со значением.

Поля класса:

T\* value = NULL; //Указатель на объект

Конструкторы и деструкторы класса:

Item(T\*); //Конструктор с параметрами

Item() {}; //Конструктор по умолчанию

~Item(); //Деструктор для удаления указателя

Перегруженные операторы класса:

bool operator > (const Item<T>&); //Оператор сравнения больше

bool operator < (const Item<T>&); //Оператор сравнения меньше

Методы класса:

void SetValue(T\*); //Сеттер для установления поля

T\* GetValue(); //Получения значения

void ReadBinaryFile(std::ifstream&); //чтение из бинарного файла

void WriteBinaryFile(std::ofstream&); //запись в бинарный файла

void Print(); //Вывод

* Для удобной работы пользователю, было описано меню с пунктами, которые вызывают методы класса Arr.
* Для хранения списка использовался двоичный файл, в котором записаны данные из структуры.
* Также были использованы функции библиотек:

Функция clock() возвращает количество временных тактов, прошедших с начала запуска программы. В программе она использовалась для определения времени работы некоторых методов. Пример использования:

clock\_t start\_time = clock();

Функция sizeof() возвращает количество байтов, которое занимает тип переменной в памяти. Использовалась для записи и чтения бинарного файла. Пример использования:

sizeof(std::string)

Функция setlocale() предназначена для того, чтобы установить локаль, которая будет использована текущей программой. Пример использования:

setlocale(0, "");

Функция SetConsoleCP() устанавливает нужную кодовую таблицу, на поток ввода. Пример использования:

SetConsoleCP(1251);

Функция tolower(char) возвращает переданный символ в нижнем регистре, в программе используется для сравнения по буквам. Пример использования:

if (tolower(str[i]) > tolower(myStr.str[i]))

Функция rand() генерирует числа в диапазоне от 0 до RAND\_MAX. RAND\_MAX — это константа, определённая в библиотеке**.** В проекте использовалась для генерации случайных строчек. Пример использования:

str[i] = symbols[rand() % (sizeof(symbols) - 1)];

Функция srand выполняет инициализацию генератора случайных чисел. Пример реализации:

srand(clock());

Функция system() выполняет заданную, через параметр, системную команду. Она вызывает командный процессор для выполнения команд. После выполнения команды, командный процессор возвращает управление программе. В программе использовалась для управления меню. Пример реализации:

system("cls");

Функция strlen() возвращает длину строки, которую мы передаем как параметр. В проекте она используется для сравнения строк, записи и чтения из файла. Пример реализации:

if (strlen(value) > strlen(item.value))

# Функционально описание разработки

* Для управления массивом со списками был создан шаблонный класс Arr.

Описание класса Arr:

Деструктор очищает память. Циклом перебирает каждый элемент в массиве указателей и с помощью операции управления памятью удаляет элемент.

Arr<T>::~Arr() //Деструктор

{

for (int i = 0; i < count; i++) //перебираем каждый указатель и удаляем

{

delete arr[i];

}

count = 0;

}

Метод создания нового списка, аргумент размер списка. Если впервые добавляет список, то выделяем память для массива указателей на списки и заполняем их пустыми значениями. Если превышен размер массива, то создаем новый массив, который хранит в 2 раза больше указателей, переносим указатели из старого массива, и заменяем указатель поля arr. После пройденных условий, записываем указатель на новый список в массив.

void Arr<T>::CreateList(int sizeList) //Создание нового списка.

{

if (count == 0) //Если массив пустой

{

arr = new List<T>\*[size] { NULL };

}

else if(count == size) //Если превышен размер, то

{

size \*= 2; //Размер массив указателей удваивается

List<T>\*\* newArr = new List<T>\*[size] { NULL };

for (int i = 0; i < count; i++) //циклом перезаписываем указатели в новый массив

{

newArr[i] = arr[i];

}

arr = newArr; //заменяем старый на новый

}

List<T>\* list = new List<T>(sizeList);

arr[count] = list; //записываем в массив

count++; //Увеличиваем счетчик списков

}

Метод вставки в первый найденный свободный список. Циклом перебираем каждый элемент массива, сравниваем размер с количеством. Соотношение указывает на переполненность списка. Если список оказался свободным, то вставляем элемент в конец. Метод возвращает значение bool, которое обозначает, удалось вставить переданное значении или нет.

bool Arr<T>::Insert(T\* value) //Добавление элемента в первый найденый свободный список

{

bool isInsert = false; //флажок, что был вставлен элемент

for (int i = 0; i < count; ++i) //циклом смотрим все списки

{

if (arr[i]->GetSize() != arr[i]->GetCount())

{

arr[i]->Insert(value); //Если да, то вставляем значение

isInsert = true; //меняем флажок

break; //прекращаем цикл

}

}

return isInsert; //Возвращаем флажок

}

Метод вставки значения по индексу списка и индексу значения. Сначала проверяем, что список не переполнен. Если он оказался свободным, то вызываем метод класса List – Insert (вставка по индексу). Метод возвращает значение bool, которое обозначает, удалось вставить переданное значении или нет.

bool Arr<T>::Insert(T\* value, int indexList, int indexValue)

{

bool isInsert = false; //флажок, что мы вставили элемент

if (arr[indexList]->GetCount() != arr[indexList]->GetSize())

{

arr[indexList]->Insert(value, indexValue);

isInsert = true; //меняем флажок

}

return isInsert; //возвращаем флажок

}

Метод удаления списка из массива. На вход принимает индекс списка. Сначала удаляется из памяти указатель, потом циклом смещаются указатели, а последний приравниваем к нулю. При удалении также уменьшаем количество списков.

void Arr<T>::DeleteList(int indexList) //Удаление списка из массива

{

delete arr[indexList]; //удаляем из памяти

for (int i = indexList; i < count - 1; i++) //циклом смещаем все элементы назад

{

arr[i] = arr[i + 1];

}

arr[count - 1] = NULL; //изменям ячейку в списке на нулевую

count--; //уменьшаем счетчик

}

Метод сортировки всего массива. Сначала посчитаем общее количество элементов, чтобы другим циклом записать элементы списков в линейный массив, такой массив удобнее сортировать. С помощью пузырьковой сортировки (сортировка представлена на рисунке 2), массив был изменен. Так как в линейный массив были записаны указатели на класс Item, и значения менялись с помощью метода класса, то переводить обратно в двумерный массив не имеет смысла.

void Arr<T>::Sort() //Общая сортировка массива

{

int totalCountItems = 0; //Посчитаем общее количество элементов

for (int i = 0; i < count; i++) //Циклом пробегаемся по каждому списку

{

totalCountItems += arr[i]->GetCount(); //И прибавляем количество элементов

}

Item<T>\*\* items = new Item<T>\*[totalCountItems];

int indexItems = 0; //индекс в массиве

int indexList = 0; //индекс списка, с которого мы переносим элементы

while(true) //Циклом записываем в линейный массив

{

if (indexList == count) //Если индекс перешел границу, то завершаем цикл

break;

for (int i = 0; i < arr[indexList]->GetCount(); i++) //Идем по каждому списку

{

items[indexItems] = arr[indexList]->GetItem(i);

indexItems++; //увеличиваем индекс у Items

}

indexList++; //Переходим к следующему списку

}

for (int i = 0; i < indexItems - 1; i++) //Сортируем пузырьком

{

for (int k = indexItems - 1; k > i; k--) //От конца до i

{

if (\*items[k] < \*items[k - 1]) //Сраниваем элементы

{

T\* temp = items[k]->GetValue();

items[k]->SetValue(items[k - 1]->GetValue());

items[k - 1]->SetValue(temp);

}

}

}

}

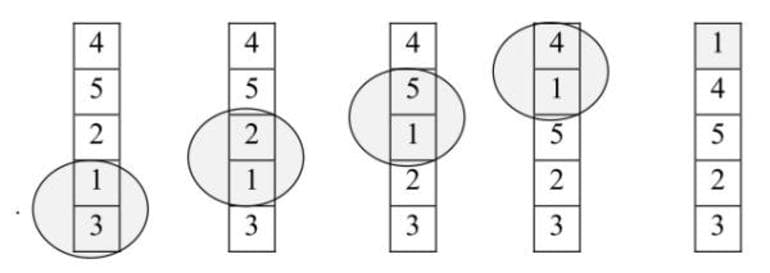


Рисунок 2 – Сортировка пузырьком

Метод вставки с сохранением порядка (наглядно представлен на рисунке 3). Сначала объявляем переменные, которые понадобятся для правильной вставки. Циклом перебираем списки и находим первый элемент, который окажется меньше по значению. С помощью описанного метода класса List происходит вставка элемента, если при этом список был полон, то метод вернет вытесненный элемент, то есть последний. Дальше, следующим циклом перебираем оставшиеся списки и вставляем в начальную позицию до тех пор, пока метод вставки не вернет пустую ссылку. Метод возвращает значение bool, которое обозначает, удалось вставить переданное значение или нет.

bool Arr<T>::InsertSaveOrder(T\* value) //Вставка элемента с сохранением порядка

{

bool isInsert = false; //флаг, что элемент вставлен

Item<T>\* item = new Item<T>(value);//для сравнения

Item<T>\* oldItem = new Item<T>(); //старый элемент, который будет вытеснен

int indexList = 0; //индекс списка, на котором был вытеснен элемент

for (int i = 0; i < count; i++) //перебираем все списки

{

for (int k = 0; k < arr[i]->GetCount(); k++) //перебираем все элементы

{

if (\*item < \*arr[i]->GetItem(k)) //сравниваем их, < переопределен

{

oldItem->SetValue(arr[i]->Insert(value, k));

indexList = i + 1;

isInsert = true; //ставим флажок на true

break;

}

}

if (isInsert)

break;

}

if (oldItem->GetValue() != NULL) //если мы получили вытесненный элемент

{

for (int i = indexList; i < count; i++) //опять перебираем

{

if (arr[i]->GetCount() == 0)

{

arr[i]->Insert(oldItem->GetValue(), 0);

break;

}

for (int k = 0; k < arr[i]->GetCount(); k++)

{

if (\*oldItem < \*arr[i]->GetItem(k)) //сравниваем

{

oldItem->SetValue(arr[i]->Insert(oldItem->GetValue(), k));

}

if (oldItem->GetValue() == NULL)

break;

}

if (oldItem->GetValue() == NULL)

break;

}

}

return isInsert; //возвращаем флажок

}

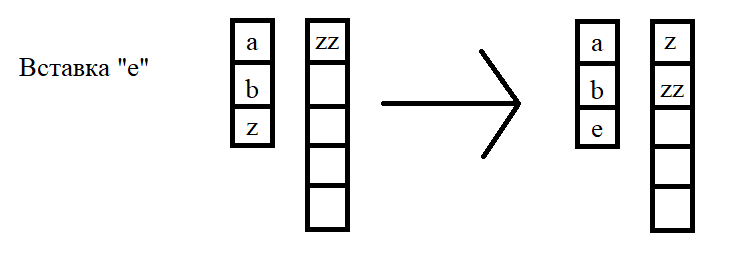


Рисунок 3 – Вставка с сохранением порядка

Метод записи в бинарный файл. На вход принимает строку с именем файла. Сначала открываем поток, потом записываем в начало файла количество списков всего. Циклом также записываем каждый список, вызывая описанный метод класса List, который записывает в бинарный файл, передаём аргументом поток.

void Arr<T>::WriteBinFile(char\* nameFile) //Запись в бинарный файл

{

std::ofstream fout(nameFile, std::ios::binary | std::ios::out);

fout.write((char\*)&count, sizeof(int)); //записываем сколько элементов в массиве

for (int i = 0; i < count; i++) //циклом перебираем каждый список и записываем в файл

{

arr[i]->WriteBinaryFile(fout);

}

fout.close(); //закрываем поток

}

Метод чтения из бинарного файла. На вход принимает строку с именем файла. Сначала открывается поток, потом вызывается деструктор, чтобы очистить весь массив. Из файла читаем количество списков, и выделяем память под списки. Циклом считываем каждый список, вызывая метод, реализованный в классе List.

void Arr<T>::ReadBinFile(char\* nameFile) //Чтение из бинарного файла

{

std::ifstream fin(nameFile, std::ios::binary | std::ios::in); //открываем поток для чтения

this->~Arr(); //удаляем все элементы предыдущего массива

fin.read((char\*)&count, sizeof(int)); //считываем из файла количество списков

arr = new List<T>\*[count]; //выделяем память под массив

for (int i = 0; i < count; i++) //циклом пробегаемся по файлу

{

List<T>\* list = new List<T>(0);

list->ReadBinaryFile(fin);

arr[i] = list; //записываем в массив указатель на считанный список

}

fin.close(); //закрываем поток

}

Метод, который помогает определить, есть ли хоть один список со свободным местом. С помощью сравнения размера и количества элементов, можно понять, заполнен список или нет.

bool Arr<T>::IsHaveFreeList()

{

for (int i = 0; i < count; i++) //пробегаемся по каждому списку

{

if (arr[i]->GetCount() != arr[i]->GetSize()) //сравниваем размер с количеством

{

return true; //если нашли свободный список, то возвращаем true

}

}

return false; //если не вернули true, то вернем false

}

* Для заголовков списков, который хранит массив, был разработан шаблонный класс List.

Конструктор с параметрами, который задает размер для количества элементов. С помощью метода класса выделяет память для массива, где будут храниться эти элементы.

List<T>::List(int \_size)

{

ChangeList(\_size);

}

С помощью деструктора происходит освобождение памяти. Циклом происходит перебор каждого элемента и с помощью операции управления памятью удаляет элемент.

List<T>::~List() //Деструктор для удаления массива

{

for (int i = 0; i < count; i++) //перебираем каждый указатель и удаляем

{

delete list[i];

}

count = 0;

}

Изменение размера списка и выделение памяти по входному параметру \_size.

void List<T>::ChangeList(int \_size) //Выделение памяти для массива list

{

size = \_size; //Устанавливаем поле, равное переданному значению

list = new Item<T>\*[size] { NULL };

}

Метод вставки элемента в конец списка. Сначала идёт проверка, что список не переполнен. Если он не до конца заполнен, то элемент вставляется в конец. Иначе возвращаем false, означающий, что метод не смог найти подходящее место для вставки.

bool List<T>::Insert(T\* value) //Вставка элемента

{

if (size != count) //Проверяем, что массив не переполнен

{

Item<T>\* item = new Item<T>(value); //создаем новый элемент

list[count] = item; //записываем указатель в массив

count++; //инкрементируем поле count

return true; //возвращаем true, т.к вставили элемент

}

return false; //возвращаем false, если массив оказался переполнен

}

Метод вставки элемента по индексу списка. В начале объявляются переменные, которые необходимы для дальнейшей работы. Сначала идёт проверка, что количество элементов равно количеству, в этом случае, записывается последний элемент, чтобы его вернуть из метода. Дальше циклом перемещаемся по массиву и смещаем указатели до переданного индекса. Переданное значение записывается в массив. Если список оказался свободным, то счётчик количества элементов увеличивается.

T\* List<T>::Insert(T\* value, int index) //Вставка элемента по индексу

{

Item<T>\* item = new Item<T>(value); //создаем новый элемент

T\* oldItem = NULL; //Старый элемент, который будет вытеснен переданным

if (size == count) //Если список переполнен, то получаем последний элемент

{

oldItem = list[count - 1]->GetValue();

}

for (int i = count; i > index; i--) //циклом смещаем все элементы вперед, заканчивая индексом, куда мы будем вставлять

{

list[i] = list[i - 1];

}

list[index] = item; //перезаписываем элемент в массиве на переданный

if (oldItem == NULL) //Если массив оказался не переполненным, то увеличиваем поле count

{

count++;

}

return oldItem;

}

Метод удаления смещает указатели в массиве назад, а элемент по индексу, который передали аргументом, удаляется из памяти. Также уменьшаем счётчик количества элементов.

void List<T>::DeleteTo(int index) //Удаление элемента по индексу

{

delete list[index]; //Удаляем из памяти

for (int i = index; i < count - 1; i++) //циклом смещаем все элементы назад

{

list[i] = list[i + 1];

}

list[count - 1] = NULL; //изменям ячейку в списке на нулевую

count--; //уменьшаем счетчик

}

Метод записи в бинарный файл принимает поток, в который записывается размер списка, счётчик элементов и сами элементы, с помощью метода, который реализован в классе Item.

void List<T>::WriteBinaryFile(std::ofstream& fout) //Запись в бинарный файла

{

fout.write((char\*)&size, sizeof(int)); //записываем размер списка

fout.write((char\*)&count, sizeof(int)); //записываем количество элементов списка

for (int i = 0; i < count; ++i) //Циклом записываем каждый элемент

{

list[i]->WriteBinaryFile(fout);

}

}

Метод чтения бинарного файла принимает поток. Из файла считывается размер списка, счётчик элементов и, с помощью метода чтения элемента класса Item, сам элемент.

void List<T>::ReadBinaryFile(std::ifstream& fin) //Чтения из бинарный файл

{

fin.read((char\*)&size, sizeof(int)); //сначала считываем размер списка

ChangeList(size); //выделяем память под массив

fin.read((char\*)&count, sizeof(int)); //считываем количество элементов

for (int i = 0; i < count; ++i) //циклом читаем по файлу элементы

{

Item<T>\* item = new Item<T>();

item->ReadBinaryFile(fin);

list[i] = item; //записываем в массив

}

}

Метод, который выводит список в консоль. Циклом перебирает каждый элемент и вызывает метод, который описан в классе Item.

void List<T>::Print() //Вывод

{

std::cout << "с максимальным размером " << size << std::endl;

for (int i = 0; i < count; ++i) //Циклом выводим каждый элемент

{

list[i]->Print();

if (i + 1 != count)

std::cout << ", ";

else

std::cout << std::endl;

}

}

* Для хранения значения был создан класс Item.

Конструктор класса принимает на вход указатель на значение и записывает в поле.

Item<T>::Item(T\* \_value) //Конструктор с параметрами

{

value = \_value;

}

Для удобства работы с объектами класса, были перегружен оператор сравнения. Сначала вычисляется минимальная длина, чтобы при сравнении по буквам не выйти за границу памяти. Сравниваемые буквы переводятся в нижний регистр. Если строки оказались по буквам равны, то следующее сравнение происходит по длине.

bool Item<T>::operator < (const Item<T>& item) //Оператор сравнения меньше

{

int minLen; //минимальная длина

if (strlen(value) > strlen(item.value)) //условием вычисляем минимальную длину

minLen = strlen(item.value);

else

minLen = strlen(value);

for (int i = 0; i < minLen; ++i) //Циклом проходим каждую букву

{

if (tolower(value[i]) > tolower(item.value[i]))

return false;

else if (tolower(value[i]) < tolower(item.value[i]))

return true;

}

if (strlen(value) < strlen(item.value))

return true;

return false;

}

Также были реализованы методы

Метод чтения из бинарного файла принимает на вход поток. Сначала считывается длина строки, чтобы правильно определить количество байт, выделенное под строку. В конец строки добавляется символ конца строки.

void Item<T>::ReadBinaryFile(std::ifstream& fin) //чтение из бинарный файл

{

int length = 0;

fin.read((char\*)&length, sizeof(int)); //считываем длину из файла

value = new char[length];

fin.read(value, length); //считываем строку из файла

value[length] = '\0';//пишем символ конца строки в считанную строку

}

Метод записи из бинарного файла принимает на вход поток. Сначала вычисляется длина строки, чтобы правильно в дальнейшем считать файл и записать строку с размерностью длины.

void Item<T>::WriteBinaryFile(std::ofstream& fout) //Запись в бинарный файла

{

int length = strlen(value); //вычисляем длину строки

fout.write((char\*)&length, sizeof(int));

fout.write(value, length);

}

# Описание пользовательского интерфейса

Для удобной работы пользователю было создано меню, которое вызывает все необходимые методы. Управление программой показано на рисунке 4.

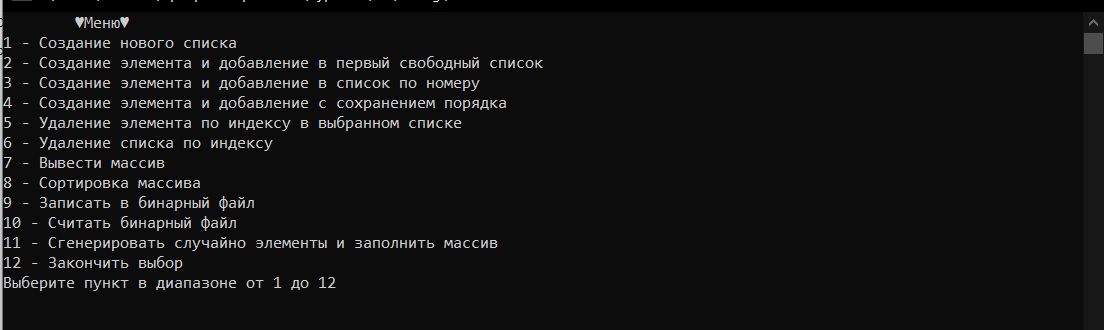


Рисунок 4 – Меню

Пункт 1 позволяет добавить в массив новый список, длину которой задаёт пользователь. Демонстрация пункта показана на рисунке 5.



Рисунок 5 – 1 пункт

Пункт 2 позволяет создать элемент и добавить в свободный список. Демонстрация пункта показана на рисунке 6.

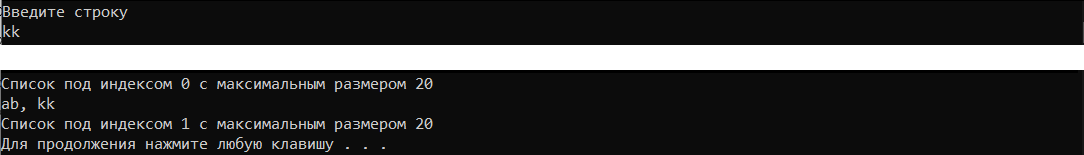


Рисунок 6 – 2 пункт

Пункт 3 позволяет создать элемент и добавить элемент по индексу списка и также выбором индекса. Демонстрация пункта показана на рисунке 7.

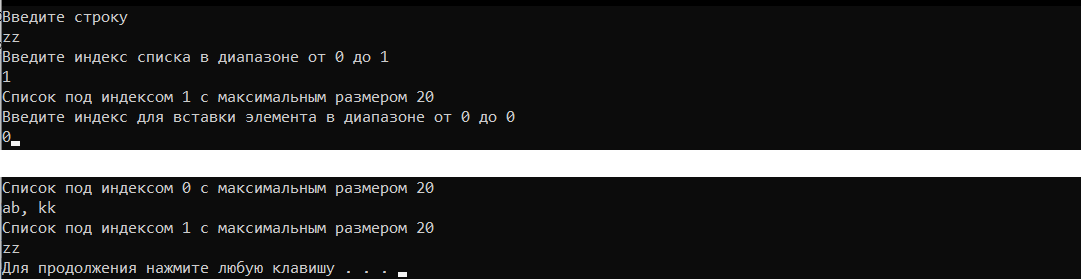


Рисунок 7 – 3 пункт

Пункт 4 позволяет создать элемент и добавить с сохранением порядка. Демонстрация пункта показана на рисунке 8.

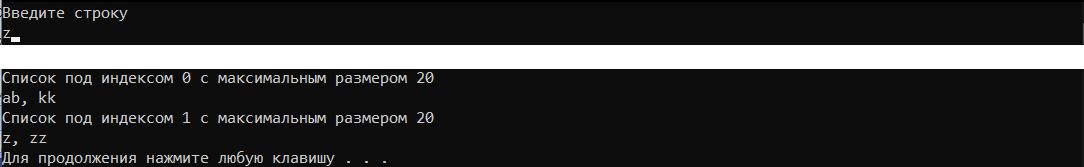


Рисунок 8 – 4 пункт

Пункт 5 позволяет удалить элемент по индексу списка и индексу. Демонстрация пункта показана на рисунке 9.

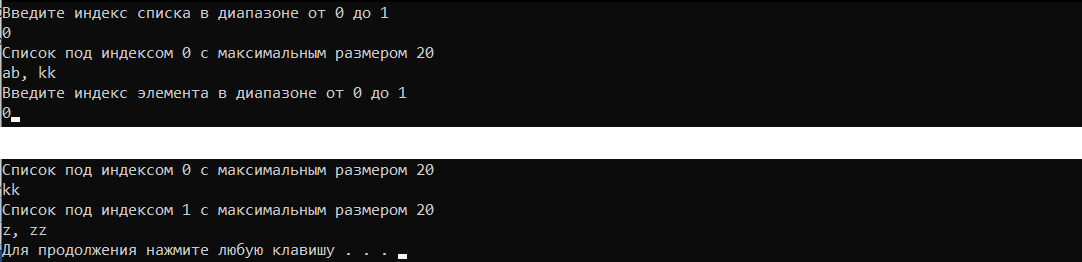


Рисунок 9 – 5 пункт

Пункт 6 позволяет удалить список по индексу. Демонстрация пункта показана на рисунке 10.

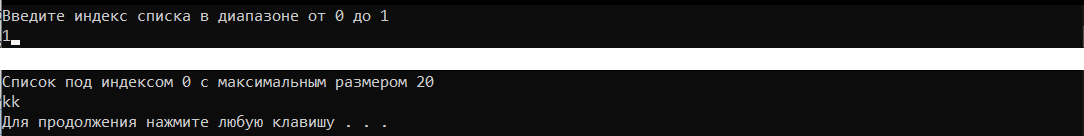


Рисунок 10 – 6 пункт

Пункт 7 позволяет вывести все элементы, которые содержатся в списках. Демонстрация пункта показана на рисунке 11.



Рисунок 11 – 7 пункт

Пункт 8 позволяет отсортировать элементы, которые содержат списки. Демонстрация пункта показана на рисунке 12.

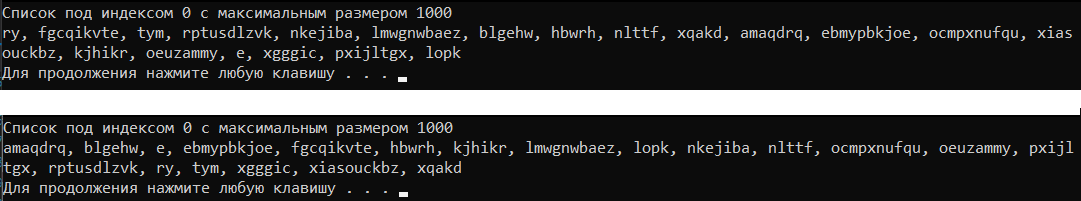


Рисунок 12 – 8 пункт

Пункт 9 позволяет записать в бинарный файл. Содержимое бинарного файла показано на рисунке 13.

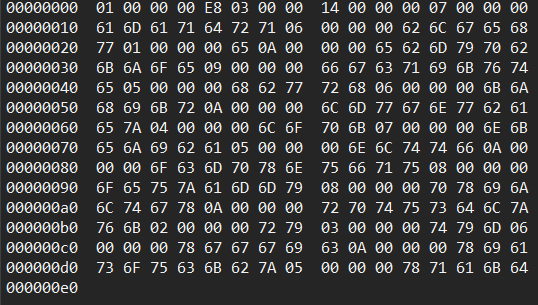


Рисунок 13 – 9 пункт

Пункт 10 позволяет считать бинарный файл. Содержимое массива показано на рисунке 14.

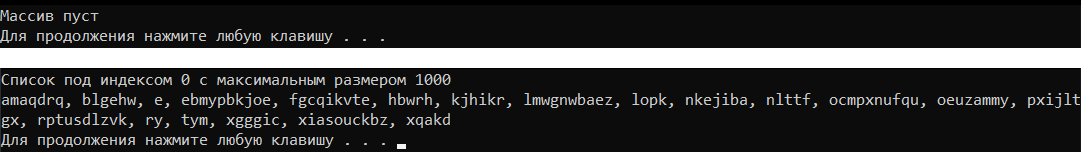


Рисунок 14 – 10 пункт

Пункт 11 позволяет сгенерировать случайно элементы. Содержимое массива показано на рисунке 15.

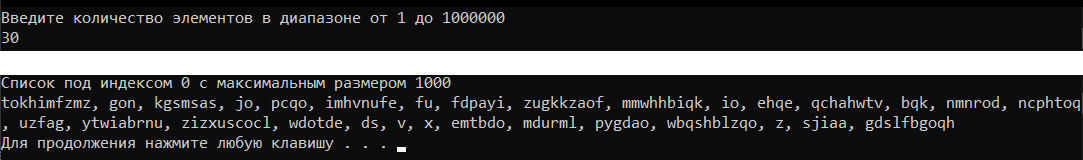


Рисунок 15 – 11 пункт

# Описание работы программы на контрольных примерах

Оценим время работу сортировки на 500, 1000, 2500 и 5000 элементах. Для этого заполним массив. На 500 элементах сортировки понадобилось 0.263 секунд, для 1000 – 0.821 секунд, для 2500 – 5.518 секунд, для 5000 – 23.812 секунд. В худшем случае затраты времени будут равны O(n^2), когда элементы отсортированы в обратном порядке. Время работы в программе представлено на рисунке 16, 17, 18 и 19.

https://sun9-23.userapi.com/impg/35d252cpbC7iJvZBMxRuzRNkVKyLiBEuj-K6dw/Lsk0NS8G5mw.jpg?size=390x53&quality=96&sign=2dfc4a57504926a683d4feeea370901b&type=album

Рисунок 16. Время работы на 500 элементах

https://sun9-8.userapi.com/impg/5GsiCFMLhnH3tt6Tu5vKWppkVMYi_dGc8cdMUQ/_Y3j0wBYi5g.jpg?size=420x50&quality=96&sign=6109c16af8ce15e8b3526f78ac90b330&type=album

Рисунок 17. Время работы на 1000 элементах

https://sun9-25.userapi.com/impg/mTg5BuDBeJgPxJ-kcjpMfAiCSk44xKKrwPOrDA/RxJ62WMKbXw.jpg?size=415x48&quality=96&sign=1d93abc150b1c85c793f98fa2307d19d&type=album

Рисунок 18. Время работы на 2500 элементах

https://sun4-15.userapi.com/impg/_p_hFNsXEze8uuCo8wUpX3g5tdWsTP4h_3cuAQ/1q6IKnI8MsM.jpg?size=418x46&quality=96&sign=8d25d333477f73b58fcf0ddb37caf07a&type=album

Рисунок 19. Время работы на 5000 элементах

График зависимости времени от количества строк представлен на рисунке 17.

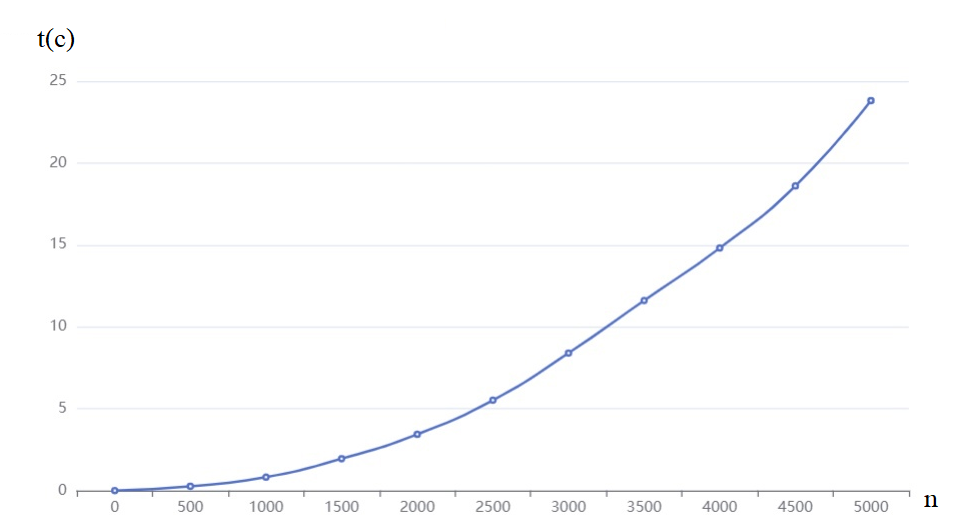


Рисунок 17 – Зависимость времени от количества строк

По графику можно заметить квадратичную зависимость у сортировки, что соответствует сложности.

# Выводы

В ходе написания курсовой работы была реализована программа, которая представляет собой простой текстовый редактор, использующий структуру данных – массив указателей на заголовки списков. Для этого были разработаны шаблонные классы Arr, List и Item. Для структуры данных были реализованы операции добавление, включение и извлечение по логическому номеру, сортировка, включение с сохранением порядка, загрузка и сохранение строк в бинарном файле.

В программа была протестирована сортировка на 500, 1000, 2500 и 5000 элементах. Определили времени и построили график, на котором видно квадратичную зависимость.

Плюсом программы является вынесение логики работы у классов. Например, класс Arr хранит в себе массив объектов List, в котором описаны методы по заданию. То есть, управление элементами происходит в классе List. Также, из-за того, что элементы хранятся в массиве, то методы отрабатывают быстрее по сравнению, например, с односвязным списком, где для получения элемента иногда приходится пробегать через все элементы.

Минусом является скорость работы сортировки, так как на большом количестве элементов сортировка может занять ощутимое количество времени.

# Список используемой литературы

1. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. — СПб.: Питер, 2003. – 461 с.
2. Объектно-ориентированное программирование в C++ / Р. Лафоре — СПб.: Питер, 2018. – 928 с.
3. Страуструп Бьярне Программирование: принципы и практика использования C++, испр. изд. : Пер. с англ. — М. : ООО “И.Д. Вильямс”, 2011. — 1248 с.

# Приложение

Arr.cpp

#include <iostream>

#include "Arr.h"

#include "Item.h"

#include <fstream>

template Arr<char>;

template <typename T>

Arr<T>::~Arr() //Деструктор

{

for (int i = 0; i < count; i++) //перебираем каждый указатель и удаляем

{

delete arr[i];

}

count = 0;

}

template <typename T>

void Arr<T>::CreateList(int sizeList) //Создание нового списка

{

if (count == 0) //Если массив пустой

{

arr = new List<T>\*[size] { NULL }; //выделяем память и заполняем нулевыми указателями

}

else if(count == size) //Если превышен размер, то

{

size \*= 2; //Размер массив указателей удваивается

List<T>\*\* newArr = new List<T>\*[size] { NULL }; //Создается новый массив и выделяется память для него в два раза больше, чем в прошлом

for (int i = 0; i < count; i++) //циклом перезаписываем указатели в новый массив

{

newArr[i] = arr[i];

}

arr = newArr; //заменяем старый на новый

}

List<T>\* list = new List<T>(sizeList); //Создаем новый список с переданным значением

arr[count] = list; //записываем в массив

count++; //Увеличиваем счетчик списков

}

template <typename T>

bool Arr<T>::Insert(T\* value) //Добавление элемента в первый найденый свободный список

{

bool isInsert = false; //флажок, что был вставлен элемент

for (int i = 0; i < count; ++i) //циклом смотрим все списки

{

if (arr[i]->GetSize() != arr[i]->GetCount()) //Проверяем, полностью ли заполнен список

{

arr[i]->Insert(value); //Если да, то вставляем значение

isInsert = true; //меняем флажок

break; //прекращаем цикл

}

}

return isInsert; //Возвращаем флажок

}

template <typename T>

bool Arr<T>::Insert(T\* value, int indexList, int indexValue) //Добавление элемента по индексу списка и индексу значения

{

bool isInsert = false; //флажок, что мы вставили элемент

if (arr[indexList]->GetCount() != arr[indexList]->GetSize()) //если список не переполнен

{

arr[indexList]->Insert(value, indexValue); //вызываем метод вставки значения по индексу

isInsert = true; //меняем флажок

}

return isInsert; //возвращаем флажок

}

template <typename T>

void Arr<T>::DeleteItem(int indexList, int indexValue) //Удаление элемента по индексу списка и индексу значения

{

arr[indexList]->DeleteTo(indexValue); //вызываем метод удаления по индексу

}

template <typename T>

void Arr<T>::DeleteList(int indexList) //Удаление списка из массива

{

delete arr[indexList]; //удаляем из памяти

for (int i = indexList; i < count - 1; i++) //циклом смещаем все элементы назад

{

arr[i] = arr[i + 1];

}

arr[count - 1] = NULL; //изменям ячейку в списке на нулевую

count--; //уменьшаем счетчик

}

template <typename T>

void Arr<T>::Sort() //Общая сортировка массива

{

//Для удобства сортировки двухуровневой структуры данных переведем ее в линейный вид

int totalCountItems = 0; //Посчитаем общее количество элементов

for (int i = 0; i < count; i++) //Циклом пробегаемся по каждому списку

{

totalCountItems += arr[i]->GetCount(); //И прибавляем количество элементов

}

Item<T>\*\* items = new Item<T>\*[totalCountItems]; //Создадим массив равный количеству посчитанных элементов

int indexItems = 0; //индекс в массиве

int indexList = 0; //индекс списка, с которого мы переносим элементы

while(true) //Циклом записываем в линейный массив

{

if (indexList == count) //Если индекс перешел границу, то завершаем цикл

break;

for (int i = 0; i < arr[indexList]->GetCount(); i++) //Идем по каждому списку

{

items[indexItems] = arr[indexList]->GetItem(i);

indexItems++; //увеличиваем индекс у Items

}

indexList++; //Переходим к следующему списку

}

for (int i = 0; i < indexItems - 1; i++) //Сортируем пузырьком

{

for (int k = indexItems - 1; k > i; k--) //От конца до i

{

if (\*items[k] < \*items[k - 1]) //Сраниваем элементы

{

T\* temp = items[k]->GetValue(); //Записываем временное значение

items[k]->SetValue(items[k - 1]->GetValue());

items[k - 1]->SetValue(temp);

}

}

}

}

template <typename T>

bool Arr<T>::InsertSaveOrder(T\* value) //Вставка элемента с сохранением порядка

{

bool isInsert = false; //флаг, что элемент вставлен

Item<T>\* item = new Item<T>(value);//для сравнения

Item<T>\* oldItem = new Item<T>(); //старый элемент, который будет вытеснен

int indexList = 0; //индекс списка, на котором был вытеснен элемент

for (int i = 0; i < count; i++) //перебираем все списки

{

for (int k = 0; k < arr[i]->GetCount(); k++) //перебираем все элементы

{

if (\*item < \*arr[i]->GetItem(k)) //сравниваем их, < переопределен

{

oldItem->SetValue(arr[i]->Insert(value, k));//получаем вставкой по индексу вытесненный элемент

indexList = i + 1; //запоминаем с какого места начинаем дальше вставлять, если элемент был вытеснен

isInsert = true; //ставим флажок на true

break;

}

}

if (isInsert)

break;

}

if (oldItem->GetValue() != NULL) //если мы получили вытесненный элемент

{

for (int i = indexList; i < count; i++) //опять перебираем

{

if (arr[i]->GetCount() == 0) //если элементов в списке равно 0, то сразу вставляем

{

arr[i]->Insert(oldItem->GetValue(), 0);

break;

}

for (int k = 0; k < arr[i]->GetCount(); k++)

{

if (\*oldItem < \*arr[i]->GetItem(k)) //сравниваем

{

oldItem->SetValue(arr[i]->Insert(oldItem->GetValue(), k));//получаем вставкой по индексу вытесненный элемент

}

if (oldItem->GetValue() == NULL) //если один список не вытеснил, то прерываем полностью цикл

break;

}

if (oldItem->GetValue() == NULL) //и прерываем второй цикл

break;

}

}

return isInsert; //возвращаем флажок

}

template <typename T>

void Arr<T>::WriteBinFile(char\* nameFile) //Запись в бинарный файл

{

std::ofstream fout(nameFile, std::ios::binary | std::ios::out); //открываем поток для записи

fout.write((char\*)&count, sizeof(int)); //записываем сколько элементов в массиве

for (int i = 0; i < count; i++) //циклом перебираем каждый список и записываем в файл

{

arr[i]->WriteBinaryFile(fout);

}

fout.close(); //закрываем поток

}

template <typename T>

void Arr<T>::ReadBinFile(char\* nameFile) //Чтение из бинарного файла

{

std::ifstream fin(nameFile, std::ios::binary | std::ios::in); //открываем поток для чтения

this->~Arr(); //удаляем все элементы предыдущего массива

fin.read((char\*)&count, sizeof(int)); //считываем из файла количество списков

arr = new List<T>\*[count]; //выделяем память под массив

for (int i = 0; i < count; i++) //циклом пробегаемся по файлу

{

List<T>\* list = new List<T>(0); //создаем новый список с размерностью 0, т.к в процессе считывания из файла он выделяет память заново

list->ReadBinaryFile(fin);

arr[i] = list; //записываем в массив указатель на считанный список

}

fin.close(); //закрываем поток

}

template <typename T>

void Arr<T>::PrintList(int index) //Вывод списка по индексу

{

std::cout << "Список под индексом " << index << " ";

arr[index]->Print();

}

template <typename T>

void Arr<T>::Print() //Вывод массива

{

if (count == 0)

std::cout << "Массив пуст\n";

else

{

for (int i = 0; i < count; i++)

{

PrintList(i);

}

}

}

template <typename T>

int Arr<T>::GetCount() //Геттер для поля count

{

return count;

}

template <typename T>

int Arr<T>::GetCountList(int index) //Геттер для получения поля Count в классе List

{

return arr[index]->GetCount();

}

template <typename T>

int Arr<T>::GetSizeList(int index) //Геттер для получения поля Count в классе List

{

return arr[index]->GetSize();

}

template <typename T>

bool Arr<T>::IsHaveFreeList() //Метод, который помогает определить, есть ли свободное место для вставки

{

for (int i = 0; i < count; i++) //пробегаемся по каждому списку

{

if (arr[i]->GetCount() != arr[i]->GetSize()) //сравниваем размер с количеством

{

return true; //если нашли свободный список, то возвращаем true

}

}

return false; //если не вернули true, то вернем false

}

Item.cpp

#include <iostream>

#include "Item.h"

#include <fstream>

template Item<char>;

template <typename T>

Item<T>::Item(T\* \_value) //Конструктор с параметрами

{

value = \_value;

}

template <typename T>

Item<T>::~Item() //Деструктор для удаления указателя

{

delete value; //Удаляем значение

}

template <typename T>

void Item<T>::SetValue(T\* \_value) //Сеттер для установления поля

{

value = \_value; //Меняем поле на переданное значение

}

template <typename T>

T\* Item<T>::GetValue() //Получения значения

{

return value;

}

template <typename T>

bool Item<T>::operator > (const Item<T>& item) //Оператор сравнения больше

{

int minLen; //минимальная длина

if (strlen(value) > strlen(item.value)) //условием вычисляем минимальную длину

minLen = strlen(item.value);

else

minLen = strlen(value);

for (int i = 0; i < minLen; ++i) //Циклом проходим каждую букву

{

if (tolower(value[i]) > tolower(item.value[i])) //сравниваем по нижнему регистру, если буква больше у value, то строка больше

return true;

else if (tolower(value[i]) < tolower(item.value[i])) //если буква больше у переданного элемента

return false;

}

//Если все буквы равны, то сравниваем по длине

if (strlen(value) > strlen(item.value)) //если значение сравниваемое по длине меньше, то строка больше

return true;

return false; //После всех пройденных условий, элементы оказались равны. Значит строка не больше, а равна

}

template <typename T>

bool Item<T>::operator < (const Item<T>& item) //Оператор сравнения меньше

{

int minLen; //минимальная длина

if (strlen(value) > strlen(item.value)) //условием вычисляем минимальную длину

minLen = strlen(item.value);

else

minLen = strlen(value);

for (int i = 0; i < minLen; ++i) //Циклом проходим каждую букву

{

if (tolower(value[i]) > tolower(item.value[i])) //сравниваем по нижнему регистру, если буква больше у value, то строка больше

return false;

else if (tolower(value[i]) < tolower(item.value[i])) //если буква больше у переданного элемента

return true;

}

//Если все буквы равны, то сравниваем по длине

if (strlen(value) < strlen(item.value)) //если значение сравниваемое по длине больше, то строка меньше

return true;

return false; //После всех пройденных условий, элементы оказались равны. Значит строка не меньше, а равна

}

template <typename T>

void Item<T>::ReadBinaryFile(std::ifstream& fin) //чтение из бинарный файл

{

int length = 0;

fin.read((char\*)&length, sizeof(int)); //считываем длину из файла

value = new char[length];

fin.read(value, length); //считываем строку из файла

value[length] = '\0';//пишем символ конца строки в считанную строку

}

template <typename T>

void Item<T>::WriteBinaryFile(std::ofstream& fout) //Запись в бинарный файла

{

int length = strlen(value); //вычисляем длину строки

fout.write((char\*)&length, sizeof(int)); //записываем ее в бинарный файл с размерностью в 4 байта(столько занимает int)

fout.write(value, length); //записываем строку, она занимает байт равную длине(1 символ = 1 байт)

}

template <typename T>

void Item<T>::Print() //Вывод

{

std::cout << value;

}

List.cpp

#include <iostream>

#include "List.h"

#include <fstream>

template List<char>;

template <typename T>

void List<T>::ChangeList(int \_size) //Выделение памяти для массива list

{

size = \_size; //Устанавливаем поле, равное переданному значению

list = new Item<T>\*[size] { NULL }; //Выделяем память по переданному значению и заполняем пустыми ссылками

}

template <typename T>

List<T>::List(int \_size) //Конструктор с параметрами, который устаналивает размер для динамического массива

{

ChangeList(\_size);

}

template <typename T>

List<T>::~List() //Деструктор для удаления массива

{

for (int i = 0; i < count; i++) //перебираем каждый указатель и удаляем

{

delete list[i];

}

count = 0;

}

template <typename T>

int List<T>::GetSize() //Геттер для поля size

{

return size;

}

template <typename T>

int List<T>::GetCount() //Геттер для поля count

{

return count;

}

template <typename T>

void List<T>::SetItem(T\* value, int index) //Изменение значения

{

list[index]->SetValue(value);

}

template <typename T>

bool List<T>::Insert(T\* value) //Вставка элемента

{

if (size != count) //Проверяем, что массив не переполнен

{

Item<T>\* item = new Item<T>(value); //создаем новый элемент

list[count] = item; //записываем указатель в массив

count++; //инкрементируем поле count

return true; //возвращаем true, т.к вставили элемент

}

return false; //возвращаем false, если массив оказался переполнен

}

template <typename T>

T\* List<T>::Insert(T\* value, int index) //Вставка элемента по индексу

{

Item<T>\* item = new Item<T>(value); //создаем новый элемент

T\* oldItem = NULL; //Старый элемент, который будет вытеснен переданным

if (size == count) //Если список переполнен, то получаем последний элемент

{

oldItem = list[count - 1]->GetValue();

}

for (int i = count; i > index; i--) //циклом смещаем все элементы вперед, заканчивая индексом, куда мы будем вставлять

{

list[i] = list[i - 1];

}

list[index] = item; //перезаписываем элемент в массиве на переданный

if (oldItem == NULL) //Если массив оказался не переполненным, то увеличиваем поле count

{

count++;

}

return oldItem; //возвращаем старый элемент. Либо пустой указатель, либо вытесненный элемент

}

template <typename T>

void List<T>::DeleteTo(int index) //Удаление элемента по индексу

{

delete list[index]; //Удаляем из памяти

for (int i = index; i < count - 1; i++) //циклом смещаем все элементы назад

{

list[i] = list[i + 1];

}

list[count - 1] = NULL; //изменям ячейку в списке на нулевую

count--; //уменьшаем счетчик

}

template <typename T>

T\* List<T>::GetValue(int index) //Геттер для получения значения из массива по индексу

{

return list[index]->GetValue(); //просто берем из массива элемент по индексу

}

template <typename T>

Item<T>\* List<T>::GetItem(int index) //Геттер для получения элемента из массива по индексу

{

return list[index]; //просто берем из массива элемент по индексу

}

template <typename T>

void List<T>::WriteBinaryFile(std::ofstream& fout) //Запись в бинарный файла

{

fout.write((char\*)&size, sizeof(int)); //записываем размер списка

fout.write((char\*)&count, sizeof(int)); //записываем количество элементов списка

for (int i = 0; i < count; ++i) //Циклом записываем каждый элемент

{

list[i]->WriteBinaryFile(fout);

}

}

template <typename T>

void List<T>::ReadBinaryFile(std::ifstream& fin) //Чтения из бинарный файл

{

fin.read((char\*)&size, sizeof(int)); //сначала считываем размер списка

ChangeList(size); //выделяем память под массив

fin.read((char\*)&count, sizeof(int)); //считываем количество элементов

for (int i = 0; i < count; ++i) //циклом читаем по файлу элементы

{

Item<T>\* item = new Item<T>();

item->ReadBinaryFile(fin);

list[i] = item; //записываем в массив

}

}

template <typename T>

void List<T>::Print() //Вывод

{

std::cout << "с максимальным размером " << size << std::endl;

for (int i = 0; i < count; ++i) //Циклом выводим каждый элемент

{

list[i]->Print();

if (i + 1 != count)

std::cout << ", ";

else

std::cout << std::endl;

}

}

main.cpp

#include <iostream>

#include "Arr.h"

#include <ctime>

#include <Windows.h>

void Menu();

int main()

{

setlocale(0, "");

SetConsoleCP(1251);

srand(clock()); //Для получения разной последовательности чисел

Menu(); //Вызываем меню

}

//Функция вывода пунктов меню и что они делают

void PrintHeader()

{

//3 символ в аски это сердечко :)

std::cout << "\t" << (char)3 << "Меню" << (char)3 << "\n"

<< "1 - Создание нового списка\n"

<< "2 - Создание элемента и добавление в первый свободный список\n"

<< "3 - Создание элемента и добавление в список по номеру\n"

<< "4 - Создание элемента и добавление с сохранением порядка\n"

<< "5 - Удаление элемента по индексу в выбранном списке\n"

<< "6 - Удаление списка по индексу\n"

<< "7 - Вывести массив\n"

<< "8 - Сортировка массива\n"

<< "9 - Записать в бинарный файл\n"

<< "10 - Считать бинарный файл\n"

<< "11 - Сгенерировать случайно элементы и заполнить массив\n"

<< "12 - Закончить выбор\n";

}

//Функция для ввода строки из консоли

char\* CreateString()

{

std::cout << "Введите строку\n";

char\* str = new char[50]; //Выделяем максимальную длину для строки 50

std::cin.ignore(); //Игнорируем для корректной работы getline

std::cin.getline(str, 50); //Считываем строчку из консоли

return str;

}

//Функция для генерации случайной строчки

char\* GenerateString(const int len) {

static const char symbols[] = "abcdefghijklmnopqrstuvwxyz"; //массив возможных символов для генерации

char\* str = new char[len]; //выделяем память под новую строчку

for (int i = 0; i < len; ++i) {

str[i] = symbols[rand() % (sizeof(symbols) - 1)]; //заполняем каждый элемент строки случайной буквой

}

str[len] = '\0'; //ставим знак конца строки

return str;

}

//Функция заполнения массива случайными строчками

void FillArray(Arr<char>\* arr, int sizeList, int countStr)

{

for (int i = 0; i < countStr; i++)

{

char\* str = GenerateString(rand() % 10 + 1); //генерируем случайную строчку

if (!arr->IsHaveFreeList()) //Если в массиве нет свободного места для вставки то

{

arr->CreateList(sizeList); //создаем новый список с размером, который мы передали

}

arr->Insert(str); //Вставляем созданную строчку в массив

}

}

//Функция для вввода в консоль номера

int SelectNumber(int min, int max)

{

if (min > max)

max = min;

int number;

std::cout << " в диапазоне от " << min << " до " << max << std::endl;

while (true)

{

std::cin >> number;

if (!std::cin) //Если ввели какое-то некорректное значение, то очищаем консоль

{

std::cin.clear();

while (std::cin.get() != '\n');

std::cout << "Вы ввели некорректное значение!\n";

}

else if(number < min || number > max) //Если число не входит в диапазон, то выводим сообщение об ошибке

{

std::cout << "Значение не входит в диапазон!\n";

}

else //Если число прошло все условия, то возвращаем, прерывая бесконечный цикл

{

break;

}

}

return number;

}

void Menu()

{

Arr<char>\* arr = new Arr<char>(); //Наш массив

int point; //Переменная выбранного пункта

int minPoint = 1, maxPoint = 12; //Минимальный номер пункта и максимальный номер пункта

int minSizeList = 1, maxSizeList = 1000; //Минимальная длина списка и максимальная длина списка

char nameFile[] = "List.bin"; //Название бинарного файла

while (true)

{

system("cls");

PrintHeader(); //Выводим пункты

std::cout << "Выберите пункт";

point = SelectNumber(minPoint, maxPoint); //Вводим пункт

if (point == maxPoint) //Если выбран пункт, который завершает меню, то выходим из бесконечного цикла

break;

system("cls");

switch (point)

{

case 1: //Создание нового списка

{

std::cout << "Введите размер создаваемо списка";

int sizeList = SelectNumber(minSizeList, maxSizeList);

arr->CreateList(sizeList);

break;

}

case 2: //Создание и вставка нового элемента

{

if (arr->IsHaveFreeList()) //Проверка, что есть свободный список

{

char\* str = CreateString();

arr->Insert(str);

}

else

{

std::cout << "В массиве нет свободных списков\n";

system("pause");

}

break;

}

case 3: //Создание и вставка элемента по номеру

{

if (arr->IsHaveFreeList()) //Проверка на свободный список

{

char\* str = CreateString();

std::cout << "Введите индекс списка";

int indexList = SelectNumber(0, arr->GetCount() - 1);

arr->PrintList(indexList);

if (arr->GetCountList(indexList) != arr->GetSizeList(indexList))

{

std::cout << "Введите индекс для вставки элемента";

int indexItem = SelectNumber(0, arr->GetCountList(indexList));

arr->Insert(str, indexList, indexItem);

}

else

{

std::cout << "Список переполнен\n";

system("pause");

}

}

else

{

std::cout << "В массиве нет свободных списков\n";

system("pause");

}

break;

}

case 4: //Создание и вставка с сохранением порядка

{

if(arr->IsHaveFreeList())

{

char\* str = CreateString();

arr->InsertSaveOrder(str);

}

else

{

std::cout << "В массиве нет свободных списков\n";

system("pause");

}

break;

}

case 5: //Удаление значения из списка

{

if (arr->GetCount())

{

std::cout << "Введите индекс списка";

int indexList = SelectNumber(0, arr->GetCount() - 1);

arr->PrintList(indexList);

std::cout << "Введите индекс элемента";

int indexItem = SelectNumber(0, arr->GetCountList(indexList) - 1);

arr->DeleteItem(indexList, indexItem);

}

else

{

std::cout << "В массиве нет списков\n";

system("pause");

}

break;

}

case 6: //Удаление списка из массива

{

if (arr->GetCount())

{

std::cout << "Введите индекс списка";

int indexList = SelectNumber(0, arr->GetCount() - 1);

arr->DeleteList(indexList);

}

else

{

std::cout << "В массиве нет списков\n";

system("pause");

}

break;

}

case 7: //Вывод массива

{

arr->Print();

system("pause");

break;

}

case 8: //Сортировка массива

{

clock\_t start\_time = clock();

arr->Sort();

clock\_t end\_time = clock();

std::cout << "Массив отсортирован за " << (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC << "с\n";

system("pause");

break;

}

case 9: //Запись в бинарный файл массив

{

clock\_t start\_time = clock();

arr->WriteBinFile(nameFile);

clock\_t end\_time = clock();

std::cout << "Массив записан в файл за " << (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC << "с\n";

system("pause");

break;

}

case 10: //Чтение из бинарного файла

{

clock\_t start\_time = clock();

arr->ReadBinFile(nameFile);

clock\_t end\_time = clock();

std::cout << "Массив считан из файла за " << (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC << "с\n";

system("pause");

break;

}

case 11: //Заполнение массива случайными строками

{

std::cout << "Введите количество элементов";

int count = SelectNumber(1, 1000000);

clock\_t start\_time = clock();

FillArray(arr, maxSizeList, count);

clock\_t end\_time = clock();

std::cout << "Массив заполнился за " << (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC << "с\n";

system("pause");

break;

}

default:

{

break;

}

}

}

}

Arr.h

#ifndef ARR

#define ARR

#include <iostream>

#include <fstream>

#include "List.h"

//Класс массива

template <typename T>

class Arr

{

private:

List<T>\*\* arr = NULL; //Динамический массив с указателями на заголовки списков

int count = 0; //Количество не нулевых указателей в массиве

int size = 1; //Размер динамического массива

public:

~Arr(); //Деструктор

void CreateList(int); //Создание нового списка

bool Insert(T\*); //Добавление элемента в первый найденый свободный список

bool Insert(T\*, int, int); //Добавление элемента по индексу списка и индексу значения

void DeleteItem(int, int); //Удаление элемента по индексу списка и индексу значения

void DeleteList(int); //Удаление списка из массива

void Sort(); //Общая сортировка массива

bool InsertSaveOrder(T\*); //Вставка элемента с сохранением порядка

void WriteBinFile(char\*); //Запись в бинарный файл

void ReadBinFile(char\*); //Чтение из бинарного файла

void PrintList(int); //Вывод списка по индексу

void Print(); //Вывод массива

int GetCount(); //Геттер для поля count

int GetCountList(int); //Геттер для получения поля Count в классе List

int GetSizeList(int); //Геттер для получения поля size в классе List

bool IsHaveFreeList(); //Метод, который помогает определить, есть ли свободное место для вставки

};

#endif ARR

Item.h

#ifndef ITEM

#define ITEM

#include <iostream>

#include <fstream>

//Класс элемента списка, который хранит в себе значение.

template <typename T>

class Item

{

private:

T\* value = NULL; //Указатель на объект

public:

Item(T\*); //Конструктор с параметрами

Item() {}; //Конструктор по умолчанию

~Item(); //Деструктор для удаления указателя

void SetValue(T\*); //Сеттер для установления поля

T\* GetValue(); //Получения значения

void ReadBinaryFile(std::ifstream&); //чтение из бинарного файла

void WriteBinaryFile(std::ofstream&); //запись в бинарный файла

void Print(); //Вывод

bool operator > (const Item<T>&); //Оператор сравнения больше

bool operator < (const Item<T>&); //Оператор сравнения меньше

};

#endif ITEM

List.h

#ifndef LIST

#define LIST

#include <iostream>

#include <fstream>

#include "Item.h"

//Класс списка, который хранит в себе сами элементы.

template <typename T>

class List

{

private:

int size = 0; //Размерность массива с элементами

int count = 0; //Количество элементов в массиве

Item<T>\*\* list = NULL; //Динамический список с указателями на элементы

void ChangeList(int);

public:

List(int); //Конструктор с параметрами, который устаналивает размер для динамического массива

~List(); //Деструктор для удаления массива

int GetSize(); //Геттер для поля size

int GetCount(); //Геттер для поля count

bool Insert(T\*); //Вставка элемента

void SetItem(T\*, int); //Изменение значения по индексу

T\* Insert(T\*, int); //Вставка элемента по индексу

void DeleteTo(int); //Удаление элемента по индексу

T\* GetValue(int); //Геттер для получения значения из массива по индексу

Item<T>\* GetItem(int); //Геттер для получения элемента из массива по индексу

void ReadBinaryFile(std::ifstream&); //чтение из бинарного файла

void WriteBinaryFile(std::ofstream&); //запись в бинарный файла

void Print(); //Вывод

};

#endif LIST