**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Линейные структуры данных. Динамические массивы и двусвязные списки

|  |  |  |
| --- | --- | --- |
| Студент гр. 3372 |  | Кипень В. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2024

**Цель работы.**

Изучение свойств и организация двусвязных списков; получение практических навыков в работе с динамическими массивами и двусвязными списками; проведение сравнительной характеристики скорости вставки, получения и удаления элементов из них.

Основные теоретические положения.

Схема распределения памяти под программу показана на рис. 1.

Рис. 1. Схема распределения памяти под программу

Область кода программы предназначена для хранения инструкций функций программы, обеспечивающих обработку данных.

Данные в программе представляются переменными и константами.

Для хранения глобальных данных предназначена область глобальных данных.

Стек программы используется при вызове функций для передачи параметров и хранения локальных данных.

Распределение памяти для хранения всех обычных переменных осуществляется компилятором, адреса и объемы соответствующих участков памяти (в области глобальных данных) жестко закреплены за этими переменными на все время работы программы и изменены быть не могут.

Однако во многих задачах невозможно заранее предсказать, сколько места (количество переменных, объемы массивов и т. д.) потребуется для решения задачи – это так называемые задачи с неопределенной размерностью.

Решить эту проблему можно лишь в том случае, если иметь механизм, позволяющий создавать новые объекты по мере возникновения необходимости в этих объектах или изменять объемы памяти, выделенные под эти объекты (например, объемы массивов).

Между областью глобальных данных и стеком располагается так называемая динамическая область памяти, которую и можно использовать в процессе работы программы для реализации механизма динамического управления памятью.

Для того чтобы создать в динамической области некоторый объект, необходима одна обычная переменная-указатель (не динамическая переменная). Сколько таких объектов понадобится для одновременной обработки, столько необходимо иметь обычных переменных-указателей. Таким образом, проблема задач неопределенной размерности созданием одиночных динамических объектов решена быть не может.

Решить эту проблему поможет возможность создавать в динамической области памяти массивы объектов с таким количеством элементов, которое необходимо в данный момент работы программы, т. е. создание динамических массивов. Действительно, для представления массива требуется всего одна переменная-указатель, а в самом массиве, на который ссылается этот указатель, может быть столько элементов, сколько требуется в данный момент времени.

Для создания одномерного динамического массива, элементами которого являются, например, действительные числа, используется следующий синтаксис инструкции new (стиль С++):

double \*Arr = new double [100];

Освободить динамическую область от этого массива можно с помощью инструкции delete:

delete [] Arr;

После этого занятый участок памяти будет возвращен в список свободной памяти и может быть повторно использован для размещения других динамических объектов.

Язык C++ поддерживает и «старый», заимствованный от языка C, стиль работы с динамической областью. Довольно часто бывает полезно использовать именно этот механизм управления динамической памятью.

В языке C отсутствуют инструкции new и delete. Вместо них для управления динамической памятью используются библиотечные функции:

// Блок прототипов функций

{

void \*malloc (size);

void \*calloc(num, size);

void free( void \*memblock);

void \*realloc( void \*memblock, size);

}

Функция malloc выделяет в динамической области size байт памяти и возвращает адрес этого участка в виде указателя (void \*).

Поскольку возвращаемый указатель не привязан ни к какому типу данных, при работе с ним потребуется явное приведение типов данных (см. пример далее).

Функция calloc выделяет в динамической области size \* num байт памяти и возвращает адрес этого участка в виде указателя (void \*).

Функция free освобождает участок динамической памяти по адресу memblock и возвращает его в список свободной памяти для повторного использования.

Функция realloc позволяет изменить размер (уменьшить или увеличить) ранее выделенной по адресу memblock памяти, установив новый размер выделенного участка равным size байт. При увеличении размера выделенного участка данные, которые хранились в старом участке, копируются в новый участок памяти. При уменьшении объема выделенного участка данные, которые хранились в нем, усекаются до нового размера. Функция возвращает указатель на область памяти нового размера.

Работа с одномерным динамическим массивом осуществляется так же, как и с обычным. При этом стиль использования динамических массивов С имеет весомое преимущество над С++, которое заключается в изменении размерности массива. Дело в том, что в C++ нет функций увеличения размерности. Увеличить размер массива можно, создав новый динамический массив нужной размерности, скопировав данные из старого массива в новый и освободив память от старого массива.

Одномерный однонаправленный список представляет собой совокупность отдельных элементов, каждый из которых содержит две части – информационную () и адресную ().

Информационная часть предназначена для хранения полезных данных и может иметь практически любой тип. Адресная часть каждого элемента содержит адрес следующего элемента списка. Схематическое изображение такого списка представлено на рис. 2.

Рис. 2. Схематическое изображение односвязного списка

Для работы со списком достаточно знать только адрес его первого элемента (). Зная адрес первого элемента списка можно последовательно получить доступ к любому другому его элементу.

Поскольку каждый элемент списка должен иметь две части, логичнее всего представить его в виде следующей структуры:

struct list

{

int data;

list \*tail;

};

Типовыми операциями при работе со списками являются:

1) создание списка;

2) освобождение памяти от списка (удаление списка);

3) доступ к заданному элементу списка для манипуляций с его информационной частью;

4) добавление нового элемента к списку;

5) удаление элемента из списка;

6) перестановка элемента списка на новую позицию внутри списка.

Достоинством подобных структур является простота добавления, удаления и перестановки элементов списка, которые осуществляются путем манипуляций с адресными частями без перезаписи всего списка.

Одним из недостатков односвязных списков является то, что узел (элемент списка) имеет указатель только на следующий элемент. Вернуться из текущего элемента к предыдущему явным способом невозможно.

Каждый узел двусвязного (двунаправленного) линейного списка содержит два поля указателей – на следующий и на предыдущий узлы. Указатель на предыдущий узел корня списка содержит нулевое значение. Указатель последнего узла также содержит нулевое значение.

Поскольку каждый элемент списка должен иметь три части, логичнее всего представить его в виде следующей структуры:

struct list

{

int data;

list \*head;

list \*tail;

};

На рис. 3 показано схематическое представление двусвязного списка. Поле содержит адрес предыдущего элемента, поле содержит адрес следующего элемента списка. Такая организация списка позволяет перемещаться по его элементам в двух направлениях.

Рис. 3. Схематическое изображение двусвязного списка

Основные действия, производимые над узлами двусвязного линейного списка (ДЛС):

1) инициализация списка;

2) добавление узла в список;

3) удаление узла из списка;

4) удаление корня списка;

5) вывод элементов списка;

6) вывод элементов списка в обратном порядке;

7) взаимообмен двух узлов списка.

Порядок действия очень похож на односвязный линейный список, но необходимо учитывать, что в двусвязном списке имеется два указателя: на следующий и предыдущий элементы.

**Постановка задачи.**

Необходимо реализовать программу, которая выполняет следующие действия.

1. Формирование двусвязного списка размерности N, где:

a) пользователь вводит количество элементов в списке, который будет автоматически заполняться случайными числами (0 до 99);

б) пользователь вводит в консоль элементы списка, N определяется автоматически по количеству введенных элементов;

2. Определение скорости создания двусвязного списка п. 2.

3. Вставка, удаление, обмена и получение элемента двусвязного списка. Удаление и получение элемента необходимо реализовать по индексу и по значению.

4. Определение скорости вставки, удаление и получения элемента двусвязного списка п. 3.

\* Необходимо реализовать динамический массив аналогично п1-п4 и сравнить производительность динамического массива и списка

**Выполнение работы.**

|  |  |
| --- | --- |
| Ввод пользователем и обработка данных | Работа алгоритма и вывод на экран |
| Меню | |
| При запуске программы перед пользователем появляется окно с меню, где он может выбрать пункт, который требуется выполнить | Меню:    Проверка на ввод символов, которые не входят в диапазон выбора: |
| После меню с навигацией, выводится информация о студентах, которые нашлись в файле на момент запуска программы, и предложение добавить нового студента |  |
| Создание двусвязного списка | |
| Пункт 1, создание двусвязного списка а) путём указания размерности и рандомных чисел б) путём ввода элементов с клавиатуры | Пункт 1.а)    Пункт 1.б) |
| Операции со списком | |
| Пункт 2, операции с двусвязным списком (получение, вставка, перестановка и удаление) |  |
| Вывод списка на экран | |
| Пункт 3, вывод двусвязного списка на экран |  |
| Создание динамического массива | |
| Пункт 4, создание динамического массива  а) путём указания размерности и рандомных чисел б) путём ввода элементов с клавиатуры | Пункт 4.а)    Пункт 4.б) |
| Операции с массивом | |
| Пункт 5, операции с динамическим массивом (получение, вставка, перестановка и удаление) |  |
| Вывод массива на экран | |
| Пункт 6, вывод динамического массива на экран |  |

**Выводы.**

Реализовано:

1. Формирование двусвязного списка размерности N, где:  
   a) пользователь вводит количество элементов в списке, который будет автоматически заполняться случайными числами (0 до 99);

б) пользователь вводит в консоль элементы списка, N определяется

автоматически по количеству введенных элементов;

1. Определение скорости создания двусвязного списка п. 2.
2. Вставка, удаление, обмена и получение элемента двусвязного списка. Удаление и получение элемента необходимо реализовать по индексу и по значению.
3. Определение скорости вставки, удаление и получения элемента двусвязного списка п. 3.
4. Формирование динамического массива размерности N, где:  
   a) пользователь вводит количество элементов в списке, который будет автоматически заполняться случайными числами (0 до 99);
5. б) пользователь вводит в консоль элементы массива, N определяется

автоматически по количеству введенных элементов;

1. Определение скорости создания динамического массива п. 5.
2. Вставка, удаление, обмена и получение элемента динамического массива. Удаление и получение элемента необходимо реализовать по индексу и по значению.
3. Определение скорости вставки, удаление и получения элемента динамического массива п. 8.

Приложение А

рабочий код

#include <iostream>

#include <chrono>

using namespace std;

struct List

{

int data;

List \*head;

List \*tail;

};

int getRandomValueFromRange(int leftLimit, int rightLimit) {

return leftLimit + rand() % (rightLimit - leftLimit + 1);

}

*// Linked list functions*

int countLength(List \* &beg) {

if (!beg) return 0;

int counter = 0;

List\* curr = beg;

while (curr) {

curr = curr->tail;

counter++;

}

return counter;

}

bool checkIndex(List\* &beg, int idx) {

return (idx >= 0 && idx < countLength(beg));

}

void printNode(List\* curr) {

cout << "\nItem\n"

<< " Data: " << curr->data << "\n"

<< " Tail: " << curr->tail << "\n"

<< " Addr: " << curr << "\n"

<< " Head: " << curr->head << "\n";

}

List\* findItemByIndex(List\* &beg, unsigned idx) {

List\* curr = beg;

int counter = 0;

if (!checkIndex(beg, idx)) return NULL;

while (curr) {

if (counter == idx) return curr;

curr = curr->tail;

counter++;

}

return NULL;

}

void printItemsByData(List\* &beg, int data) {

List\* curr = beg;

while (curr) {

if (curr->data == data) printNode(curr);

curr = curr->tail;

}

}

int getIndexOfItem(List\* beg, int item, int startIdx = 0) {

List\* curr = beg;

int counter = 0;

if (startIdx) if (!checkIndex(beg, startIdx)) return -1;

while (curr) {

if (counter >= startIdx) {

if (curr->data == item) return counter;

}

curr = curr->tail;

counter++;

}

return -1;

}

int getNumberOfEntries(List\* &beg, int item) {

List\* curr = beg;

int counter = 0;

while (curr) {

if (curr->data == item) counter++;

curr = curr->tail;

}

return counter;

}

List \*createList(int length)

{

if (!length) return NULL;

List \*curr = 0,

\*next = 0;

for (int i = 1; i <= length; ++i)

{

curr = new List;

curr -> tail = next;

if (next) {

next -> head = curr;

}

next = curr;

}

curr -> head = 0;

return curr;

}

List\* addItem(List\* &beg, int data) {

List\* lastItem = findItemByIndex(beg, countLength(beg) - 1);

List\* item = new List;

item->data = data;

item->head = lastItem;

item->tail = 0;

if (lastItem) lastItem->tail = item;

else beg = item;

return item;

}

List\* insItem(List\* &beg, unsigned index, int data=0)

{

if(!checkIndex(beg, index) && index != countLength(beg)) {

cout << "\nIncorrect index value\n";

return NULL;

}

List \* item = new List;

if (data == 0) data = getRandomValueFromRange(0, 99);

item->data = data;

if (!index || !beg) {

beg->head = item;

item->head = NULL;

item->tail = beg;

beg = item;

return item;

}

List\* prevItem = beg;

--index;

while (prevItem->tail && (index--)) {

prevItem = prevItem->tail;

}

item->head = prevItem;

if (prevItem->tail) prevItem->tail->head = item;

item->tail = prevItem->tail;

prevItem->tail = item;

return item;

}

void delItemByIndex(List\* &beg, int index)

{

if(!checkIndex(beg, index)) {

cout << "\nIncorrect index value\n";

return;

}

List\* item = NULL;

if (index == 0) {

item = beg -> tail;

delete beg;

beg = item;

if (countLength(beg) == 0) {

delete beg;

beg = NULL;

return;

}

beg -> head = NULL;

return;

}

item = findItemByIndex(beg, index - 1);

List\* dItem = item -> tail;

item -> tail = dItem -> tail;

List\* nItem = findItemByIndex(beg, index + 1);

if (nItem) nItem -> head = item;

delete dItem;

}

void delItemByValue(List\* &beg, int value)

{

while(getNumberOfEntries(beg, value) > 0) {

int index = getIndexOfItem(beg, value);

delItemByIndex(beg, index);

}

}

void swapElementsByIndexes(List\* &beg, int index1, int index2) {

if (!checkIndex(beg, index1) || !(checkIndex(beg, index2))) {

cout << "\nIncorrect index value(-s)\n";

return;

}

if (index2 < index1) {

int buffer = index1;

index1 = index2;

index2 = buffer;

}

List\* el1m = findItemByIndex(beg, index1 - 1);

List\* el1 = findItemByIndex(beg, index1);

List\* el1p = findItemByIndex(beg, index1 + 1);

List\* el2m = findItemByIndex(beg, index2 - 1);

List\* el2 = findItemByIndex(beg, index2);

List\* el2p = findItemByIndex(beg, index2 + 1);

List\* head2 = el2->head;

List\* tail2 = el2->tail;

List\* head1 = el1->head;

List\* tail1 = el1->tail;

switch (index2 - index1) { *// abs(index1 - index2)*

case 0:

return;

case 1:

el2->tail = el1;

el1->head = el2;

break;

default:

el2->head = head1;

el2->tail = tail1;

el1->head = head2;

el1p->head = el2;

el2m->tail = el1;

break;

}

el2->head = head1;

el1->tail = tail2;

if (index1 != 0) el1m->tail = el2;

else beg = el2;

if (index2 != countLength(beg) - 1) el2p->head = el1;

}

void printList(List\* &beg) {

if (!beg) {

cout << "\nThe list is empty(((\n";

return;

}

List\* curr = beg;

while (curr) {

printNode(curr);

curr = curr->tail;

}

}

void clearList(List\* &beg) {

if (!beg) return;

List\* curr = beg;

while (true) {

if (curr->head) delItemByIndex(beg, getIndexOfItem(beg, curr->head->data));

if (!curr->tail) break;

curr = curr->tail;

}

delItemByIndex(beg, getIndexOfItem(beg, curr->data));

delete beg;

beg = NULL;

}

*// Array functions*

bool checkArrayIndex(int\* &arr, const unsigned size, unsigned idx) {

return (idx >= 0 && idx < size);

}

int countArrayEntries(int\* &arr, const unsigned size, int data) {

int counter = 0;

for (int i = 0; i < size; i++) if (arr[i] == data) counter++;

return counter;

}

void swapArrayElementsByIndexes(int\* &arr, const unsigned size, unsigned idx1, unsigned idx2) {

if (!checkArrayIndex(arr, size, idx1) || !checkArrayIndex(arr, size, idx2)) {

cout << "\nIncorrect index value(-s)\n";

return;

}

int buffer = arr[idx1];

arr[idx1] = arr[idx2];

arr[idx2] = buffer;

}

void printElement(int data, unsigned idx = -1) {

cout << "\nItem\n"

<< " Data: " << data << "\n";

if (idx != -1) cout << " Index: " << idx << "\n";

}

int findArrayItemByIndex(int\* &arr, const unsigned size, const unsigned idx) {

if (!checkArrayIndex(arr, size, idx)) {

cout << "\nIncorrect index value\n";

return 0;

}

for (int i = 0; i < size; i++) if (i == idx) return arr[i];

return 0;

}

unsigned getIndexOfArrayItem(int\* &arr, const unsigned size, int data, unsigned startIdx = 0) {

for (int i = startIdx; i < size; i++) if (arr[i] == data) return i;

return -1;

}

void printArrayItemsByData(int\* &arr, const unsigned size, int data) {

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

if (arr[i] == data) printElement(arr[i], i);

}

}

void delArrayItemByIndex(int\* &arr, unsigned &size, unsigned idx) {

if (!checkArrayIndex(arr, size, idx)) {

cout << "\nIncorrect index value\n";

return;

}

int\* res = new int[size - 1];

for (int i = 0, j = 0; i < size; i++) {

if (i != idx) {

res[j] = arr[i];

j++;

}

}

delete[] arr;

size--;

arr = res;

}

void delArrayItemByValue(int\* &arr, unsigned &size, int data) {

while (countArrayEntries(arr, size, data) > 0) {

int index = getIndexOfArrayItem(arr, size, data);

delArrayItemByIndex(arr, size, index);

}

}

void insArrayItem(int\* &arr, unsigned &size, unsigned idx, int data){

if (!checkArrayIndex(arr, size, idx) && idx != size) {

cout << "\nIncorrect index value\n";

return;

}

size++;

int \*res = new int[size];

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

res[i] = arr[i];

}

res[idx] = data;

for (int i = idx; i < size - 1; i++) {

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

}

delete[] arr;

arr = res;

}

void clearArray(int\* &arr, unsigned &size) {

delete[] arr;

arr = NULL;

size = 0;

}

void printArray(int\* &arr, const unsigned size) {

for (int i = 0; i < size; i++) printElement(arr[i], i);

}

void increaseByNumber(int\* &arr, const unsigned size, int number) {

for (int i = 0; i < size; i++) arr[i] += number;

}

void insNegativesFromArrayToList(List\* &beg, int\* &arr, const unsigned size) {

for (int i = 0; i < size; i++) { *// не стал отсотрировывать массив, чтобы сохранить исходный порядок элементов*

if (arr[i] < 0) addItem(beg, arr[i]);

}

}

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

List\* list = NULL;

int \*arr = NULL;

unsigned arrSize = 0;

int index, data;

char actionType;

char choiseType;

auto start = chrono::steady\_clock::now();

auto end = chrono::steady\_clock::now();

while(true) {

cout << "\nNavigation:" << "\n"

<< "1) Create a new list" << "\n"

<< "2) Operations with list" << "\n"

<< "3) Print list" << "\n"

<< "4) Create a new array" << "\n"

<< "5) Operations with array" << "\n"

<< "6) Print array" << "\n"

<< "7) IDZ #12" << "\n";

cin.clear(); *// Clearing the input stream from possible errors*

cin.sync();

short int workPoint;

cout << "Select point of work (number 1 to 7): ";

cin >> workPoint;

switch (workPoint)

{

case 1: {

cout << "Choose the way (eng)\n"

<< "(A) - Enter the dimension and fill it with random\n"

<< "(B) - Enter the numbers whatever you want\n";

char creatingType;

cin >> creatingType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (creatingType == 'B' || creatingType == 'b') {

cout << "Enter items, to stop it - enter any char\n";

clearList(list);

int item;

cin.clear(); *// Clearing the input stream from possible errors*

cin.sync();

start = chrono::steady\_clock::now();

while (cin >> item) addItem(list, item);

end = chrono::steady\_clock::now();

cout << "\nList length: " << countLength(list) << "\n";

} else {

cout << "\nEnter elements count: ";

clearList(list);

unsigned listSize = 0;

cin >> listSize;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

list = createList(listSize);

List\* curr = list;

while (curr) {

curr->data = getRandomValueFromRange(0, 99);

curr = curr->tail;

}

end = chrono::steady\_clock::now();

}

printList(list);

cout << "\nTime to Create: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

}

case 2: {

cout << "Choose the action (eng)\n"

<< "(G) - Get element\n"

<< "(I) - Insert element\n"

<< "(S) - Swap elements\n"

<< "(D) - Delete element\n";

List\* foundedItem = NULL;

cin >> actionType;

switch (actionType)

{

case 'G':

cout << "By index or by value (I/V)?: ";

cin >> choiseType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (choiseType == 'I' || choiseType == 'i') {

cout << "Enter an index of element (length is " << countLength(list) << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

foundedItem = findItemByIndex(list, index);

end = chrono::steady\_clock::now();

if (!foundedItem) {

cout << "\nYou entered an incorrect value\n";

break;

}

printNode(foundedItem);

} else {

cout << "Enter a value of element: ";

cin >> data;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

printItemsByData(list, data);

end = chrono::steady\_clock::now();

}

cout << "\nTime to Get: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

case 'I':

cout << "Enter an index of new element (length is " << countLength(list) << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

cout << "Enter a value of element (or enter any char to random): ";

cin >> data;

start = chrono::steady\_clock::now();

if (!cin.good()) insItem(list, index);

else insItem(list, index, data);

end = chrono::steady\_clock::now();

printList(list);

cout << "\nTime to Insert: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

case 'S':

cout << "Enter an index of first element (length is " << countLength(list) << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

cout << "Enter an index of second element (length is " << countLength(list) << "): ";

int index2;

cin >> index2;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

swapElementsByIndexes(list, index, index2);

cout << "\nList after:\n";

printList(list);

break;

case 'D':

cout << "By index or by value (I/V)?: ";

cin >> choiseType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (choiseType == 'I' || choiseType == 'i') {

cout << "Enter an index of element (length is " << countLength(list) << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

delItemByIndex(list, index);

end = chrono::steady\_clock::now();

} else {

cout << "Enter a value of element: ";

cin >> data;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

delItemByValue(list, data);

end = chrono::steady\_clock::now();

}

cout << "\nList after (length is " << countLength(list) << "):\n";

printList(list);

cout << "\nTime to Delete: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

default:

cout << "\nYou entered an incorrect value\n";

break;

}

break;

}

case 3: {

printList(list);

break;

}

case 4: {

cout << "Choose the way (eng)\n"

<< "(A) - Enter the dimension and fill it with random\n"

<< "(B) - Enter the numbers whatever you want\n";

unsigned curSize = 0;

char creatingType;

cin >> creatingType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (creatingType == 'B' || creatingType == 'b') {

cout << "Enter items, to stop it - enter any char\n";

int item;

cin.clear(); *// Clearing the input stream from possible errors*

cin.sync();

clearArray(arr, arrSize);

start = chrono::steady\_clock::now();

while (cin >> item) insArrayItem(arr, curSize, curSize, item);

end = chrono::steady\_clock::now();

arrSize = curSize;

cout << "\nList length: " << arrSize << "\n";

} else {

cout << "\nEnter elements count: ";

clearArray(arr, arrSize);

cin >> arrSize;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

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

insArrayItem(arr, curSize, i, getRandomValueFromRange(0, 99));

}

end = chrono::steady\_clock::now();

}

printArray(arr, arrSize);

cout << "\nTime to Create: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

}

case 5: {

cout << "Choose the action (eng)\n"

<< "(G) - Get element\n"

<< "(I) - Insert element\n"

<< "(S) - Swap elements\n"

<< "(D) - Delete element\n";

int foundedItem = 0;

cin >> actionType;

switch (actionType)

{

case 'G':

cout << "By index or by value (I/V)?: ";

cin >> choiseType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (choiseType == 'I' || choiseType == 'i') {

cout << "Enter an index of element (length is " << arrSize << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

foundedItem = findArrayItemByIndex(arr, arrSize, index);

end = chrono::steady\_clock::now();

if (!foundedItem) {

cout << "\nYou entered an incorrect value\n";

break;

}

printElement(foundedItem);

} else {

cout << "Enter a value of element: ";

cin >> data;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

printArrayItemsByData(arr, arrSize, data);

end = chrono::steady\_clock::now();

}

cout << "\nTime to Get: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

case 'I':

cout << "Enter an index of new element (length is " << arrSize << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

cout << "Enter a value of element (or enter any char to random): ";

cin >> data;

start = chrono::steady\_clock::now();

if (!cin.good()) insArrayItem(arr, arrSize, index, getRandomValueFromRange(0, 99));

else insArrayItem(arr, arrSize, index, data);

end = chrono::steady\_clock::now();

printArray(arr, arrSize);

cout << "\nTime to Insert: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

case 'S':

cout << "Enter an index of first element (length is " << arrSize << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

cout << "Enter an index of second element (length is " << arrSize << "): ";

int index2;

cin >> index2;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

swapArrayElementsByIndexes(arr, arrSize, index, index2);

cout << "\nArray after:\n";

printArray(arr, arrSize);

break;

case 'D':

cout << "By index or by value (I/V)?: ";

cin >> choiseType;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

if (choiseType == 'I' || choiseType == 'i') {

cout << "Enter an index of element (length is " << arrSize << "): ";

cin >> index;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

delArrayItemByIndex(arr, arrSize, index);

end = chrono::steady\_clock::now();

} else {

cout << "Enter a value of element: ";

cin >> data;

if (!cin.good()) {

cout << "\nYou entered an incorrect value\n";

break;

}

start = chrono::steady\_clock::now();

delArrayItemByValue(arr, arrSize, data);

end = chrono::steady\_clock::now();

}

cout << "\nArray after (length is " << arrSize << "):\n";

printArray(arr, arrSize);

cout << "\nTime to Delete: " << chrono::duration\_cast<chrono::microseconds>(end - start).count() << " mcs" << "\n";

break;

default:

cout << "\nYou entered an incorrect value\n";

break;

}

break;

}

case 6: {

printArray(arr, arrSize);

break;

}

case 7: {

if (!arrSize) {

cout << "\nArray is empty(\n";

break;

}

int randomNumber = getRandomValueFromRange(0, 5);

cout << "\nArray before:\n";

printArray(arr, arrSize);

cout << "\nRandom number is " << randomNumber << "\n";

increaseByNumber(arr, arrSize, randomNumber);

cout << "\nArray after:\n";

printArray(arr, arrSize);

cout << "\nList before:\n";

printList(list);

insNegativesFromArrayToList(list, arr, arrSize);

cout << "\nList after:\n";

printList(list);

break;

}

default: {

cout << "\n" << "You did not enter a number in the range from 1 to 7";

break;

}

}

cin.clear(); *// Clearing the input stream from possible errors*

cin.sync();

char stopFlag;

cout << "\n" << "Continue the program? (Y/N) ";

cin >> stopFlag;

if (stopFlag != 'Y' && stopFlag != 'y') break;

}

return 0;

}