

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

ОТЧЕТ
по практической работе №8
по дисциплине «Программирование»
Тема: Линейные структуры данных. Динамические массивы и
двусвязные списки

Студент гр.

0324

Сотина Е.А.

Преподаватель

Глущенко А.Г.

Санкт-Петербург

2020

Цель работы.

Изучение свойств и организация динамических массивов и 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.

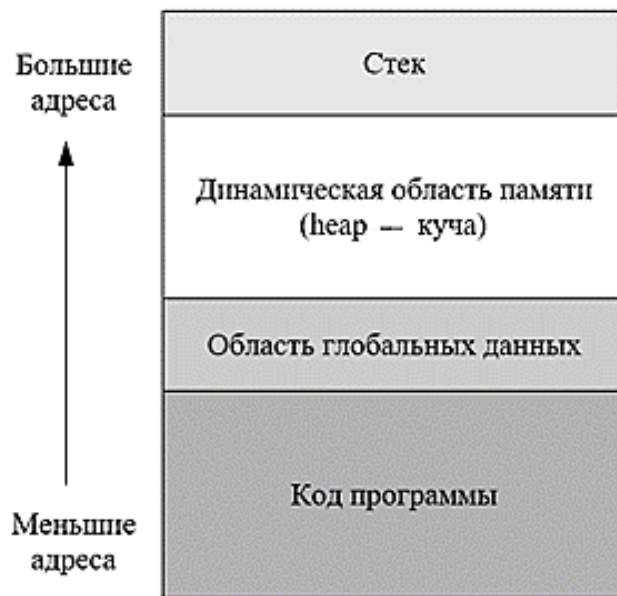


Рис. 1. Схема распределения памяти под программу

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 C++):

```
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 потребуются 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 `C` имеет весомое преимущество над `C++`, 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 (`Data`) и адресную (`Tail`).

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.

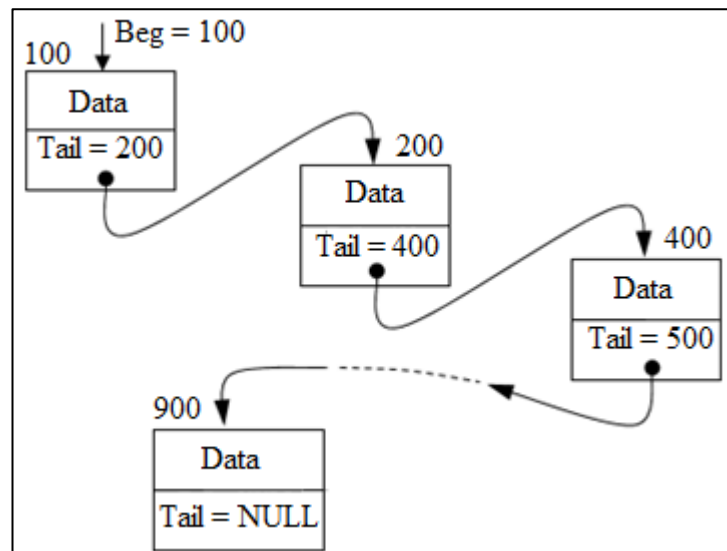


Рис. 2. Схематическое изображение односвязного списка

Для работы со списком достаточно знать только адрес его первого элемента (Beg). 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 Head содержит адрес предыдущего элемента, поле Tail содержит адрес следующего элемента списка. Такая организация списка позволяет перемещаться по его элементам в двух направлениях.

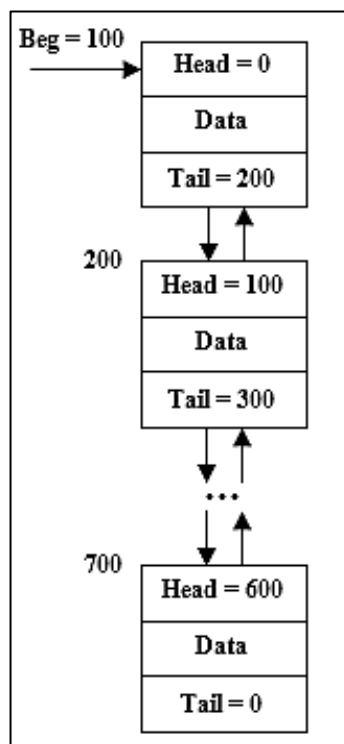


Рис. 3. Схематическое изображение двусвязного списка

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 целочисленного одномерного массива размерности N , где:
 - а) пользователь вводит количество элементов в массиве, который будет автоматически заполняться случайными числами (0 до 99);
 - б) пользователь вводит в консоль элементы массива, N определяется автоматически по количеству введенных элементов;
 - в) массив считывается с файла, N определяется как количество элементов массива в файле.
2. Определение скорости создания динамического массива п. 1.
3. Вставка, удаление и получение элемента массива. Удаление и получение элемента необходимо реализовать по индексу и по значению.
4. Определение скорости вставки, удаления и получения элемента массива п. 3.
5. Формирование двусвязного списка размерности N , где:
 - а) 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;
 - в) список считывается с файла, N определяется как количество элементов списка в файле.
6. Определение скорости создания двусвязного списка п. 5.

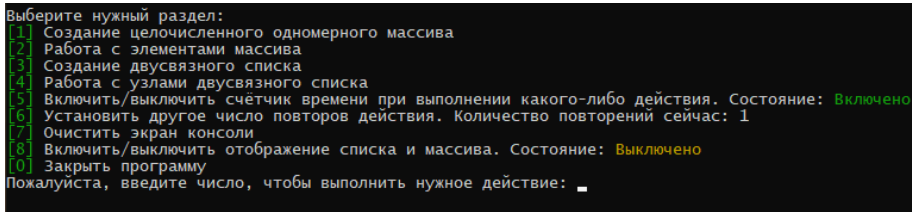
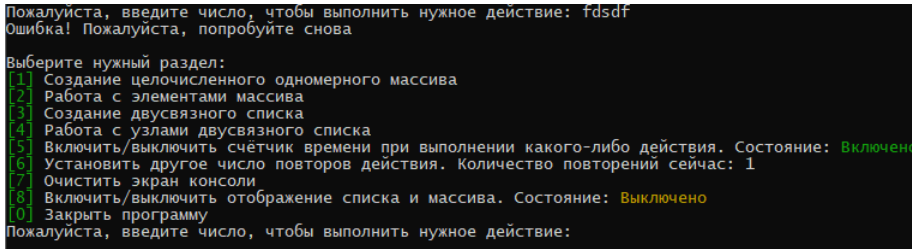
7. Вставка, удаление и получение элемента двусвязного списка. Удаление и получение элемента необходимо реализовать по индексу и по значению.

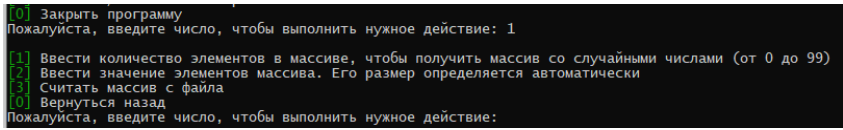
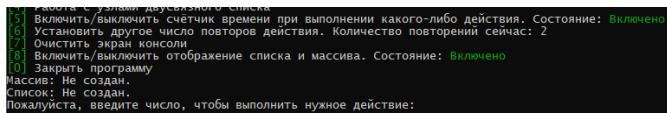
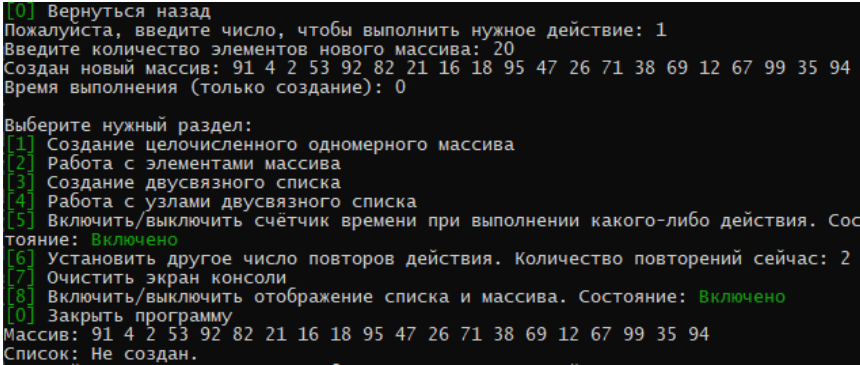
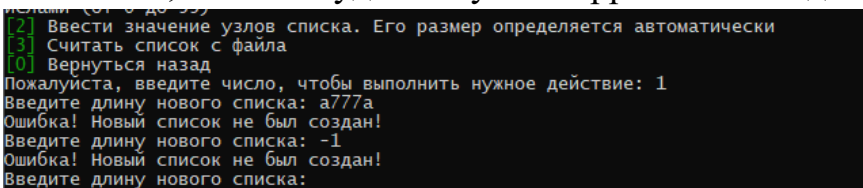
8. Определение скорости вставки, удаление и получения элемента двусвязного списка п. 7.

Должна быть возможность запуска каждого пункта многократно, если есть возможность (если в списке/массиве нет элементов, то нельзя ничего удалить и об этом нужно сообщить пользователю). Необходимо сравнить результаты. Для этого пункты 1–4 и 5–8 должны принимать одинаковые значения.

Выполнение работы.

Код программы представлен в приложении А.

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 тип будущей вводимой последовательности	<p>Меню:</p>  <p>Проверка на ввод символов, которые не входят в диапазон выбора:</p> 

Дополнительное меню	
П. 1-4 открывают вспомогательное меню	<p>Способы создания массива:</p> 
Прочие вспомогательные возможности	
Помимо основных задач, пользователь может настроить наиболее удобный для него вариант выполнения программы (п. 5-8)	<p>Отредактированные настройки отображаются в меню:</p> 
Создание массива/списка	
Программа предусматривает три способа создания массива/списка: она может самостоятельно заполнить массив/список случайными числами при вводе его длины, может запомнить введенный пользователем ряд чисел или считать его с файла. Реагирует на некорректный ввод и отображается перед пользователем о каждом действии	<p>При создании массива случайных чисел, пользователь должен ввести его длину:</p>  <p>При некорректном вводе, пользователю сообщается об ошибке. Пользователь должен вводить количество элементов, пока не будет получен корректный ввод:</p> 

Продолжение Таблицы

Вставка нового элемента	
<p>Подразумевается, что новое значение элемента поступает в любое место массива/списка, поэтому выполнение данной задачи имеет свои особенности: в массиве элемент вставляется в конец, а в списке после второго элемента.</p>	<p>Выполнение вставки элемента в список при повторе в 2 раза:</p> <pre>Список: 48 93 1 6 50 70 29 90 5 46 48 64 42 40 6 88 42 90 35 16 Пожалуйста, введите число, чтобы выполнить нужное действие: 4 Выберите, что хотите сделать: [1] Вставить новый элемент в список [2] Удалить узел списка (по индексу) [3] Удалить узел списка (по значению) [4] Получить значение узла списка по индексу [5] Получить все индексы узла списка по указанному значению [0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 1 Введите значение нового элемента: 2 Создан новый список: 48 2 2 93 1 6 50 70 29 90 5 46 48 64 42 40 6 88 42 90 35 16 Длина списка - 22 элементов. Время выполнения: 0</pre> <p>При неправильном вводе производится выход из действия:</p> <pre>01 Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 1 Введите значение нового элемента: пав5 Ошибка! Массив не был изменён!</pre>
Удаление элементов	
<p>Пользователю доступно два способа удаления элемента: по индексу или по значению. По значению удаляются элементы столько раз, сколько введено количество повторений</p>	<p>Удаление элемента по индексу:</p> <pre>[4] Получить значение узла списка по индексу [5] Получить все индексы узла списка по указанному значению [0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 2 Введите индекс элемента, который хотите удалить (нумерация элементов с нуля): 0 Создан новый список: 2 93 1 6 50 70 29 90 5 46 48 64 42 40 6 88 42 90 35 16 Длина списка - 20 элементов. Время выполнения: 0</pre> <p>Удаление элемента по значению:</p> <pre>Массив: 91 4 2 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 2 2 Список: Не создан. Пожалуйста, введите число, чтобы выполнить нужное действие: 2 Выберите, что хотите сделать: [1] Вставить новый элемент в массив [2] Удалить элемент массива (по индексу) [3] Удалить элемент массива (по значению) [4] Получить значение элемента массива по индексу [5] Получить все индексы элемента массива по указанному значению [0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 3 Введите значение элемента, который хотите удалить: 2 Элемент удалён Новый массив: 91 4 53 92 82 21 16 18 95 47 26 71 38 69 12 67 99 35 94 2 Длина массива - 20 элементов. Время выполнения: 0.006</pre>

Продолжение Таблицы

Получение адресов элементов	
Пользователь может получить элементы как по индексу, так и по значению.	<p>Вывод значение элемента по индексу:</p> <pre>[0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 4 Введите индекс элемента, который хотите получить (нумерация элементов с нуля): 0 Получен элемент 41 с адресом 00DF2ED0 Время выполнения: 0</pre>
	<p>Вывод индексов элементов по значению:</p> <pre>[0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 5 Введите значение элемента, который хотите получить: 5 Получен элемент 5 с адресом 00DEE358 Время выполнения: 0</pre>
	<p>При отсутствии нужного элемента, пользователю об этом сообщается:</p> <pre>[0] Вернуться назад Пожалуйста, введите число, чтобы выполнить нужное действие: 4 Введите индекс элемента, который хотите получить (нумерация элементов с нуля): 50 Ошибка! Элемент не был найден.</pre>
	<pre>Пожалуйста, введите число, чтобы выполнить нужное действие: 5 Введите значение элемента, который хотите получить: 1 Не найдено Время выполнения: 0.002</pre>

Окончание Таблицы

Сравнение результатов (при повторе в 50000 раз и 30 элементов)	
Создание массива и списка	Массив: 0.114 с
	Список: 0.339 с
Считывание с файла 50030 элементов (без повтора)	Массив: 0.179 с
	Список: 0.174 с
Вставка нового элемента	Массив: 0.008 с
	Список: 0.008 с
Удаление элемента по индексу	Массив: 5.613 с
	Список: 0.011 с
Удаление элемента по значению	Массив: 4.276 с
	Список: 0.008 с
Получение элемента по индексу (50000 элемент)	Массив: 0 с
	Список: 12.483 с
Получение индексов элементов по значению (с выводом)	Массив: 29.768 с
	Список: 36.033 с

Выводы.

ПРИЛОЖЕНИЕ А

КОД ПРОГРАММЫ

Название файла: lab2.cpp

```
#include <iostream>
#include <time.h>
#include <fstream>
using namespace std;

// ДИНАМИЧЕСКИЙ ОДНОМЕРНЫЙ ЦЕЛОЧИСЛЕННЫЙ МАССИВ
// == Прочие действия ==
// Вывод массива
void outputArr(int*& arr, int size)
{
    if (size <= 0) { cout << "Ошибка!\n"; return; }
    for (int i = 0; i < size; i++) { cout << arr[i] << ' '; }
    cout << '\n';
}

// Создание массива размера N. Данные - случайные числа от 0 до 99
int* createArr(int N)
{
    if (N <= 0) { cout << "Ошибка!\n"; return NULL; }
    int* arr = new int[N]();
    for (int i = 0; i < N; i++)
    {
        arr[i] = rand() % 100; //заполнение случайным числом от 0 до 99
    }
    return arr;
}

// == ФУНКЦИИ ПО ПУНКТАМ ==
//пункт 1.а по массивам: N случайных чисел
int taskArr1a(int*& arr, bool time, int repeat)
{
    clock_t start, end; //таймер
    int n;
    cout << "Введите количество элементов нового массива: ";
    while (!(cin >> n) || (n <= 0)) //проверка на корректность ввода
    {
        cout << "Ошибка! Новый массив не был создан!\n";
        cin.clear();
        cin.sync();
        while (cin.get() != '\n');
        cout << "Введите количество элементов нового массива: ";
    }

    if (time) { start = clock(); } //начало таймера
```

```

        for (int r = 0; r < repeat; r++){ if (arr) { delete[] arr; } arr =
createArr(n); }
        if (time) { end = clock(); } //конец таймера
        cout << "Создан новый массив: ";
        outputArr(arr, n);
        if (time) { cout << "Время выполнения (только создание): " << ((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
        return n;
    }

//пункт 1.b по массивам: произвольный размер
int taskArr1b(int*& arr, bool time)
{
    clock_t start, end;
    int x, size = 1;
    cout << "Введите элементы нового массива: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Новый массив не был создан\n"; }
    else
    {
        if (time) { start = clock(); } //начало таймера
        if (arr) { delete[] arr; }
        arr = createArr(1);
        arr[0] = x;
        while (cin && cin.get() != '\n') //считываются все значения, пока
не встречен мусор
        {
            int* tmp;
            cin >> x;
            if (cin)
            {
                size++;
                tmp = (int*)realloc(arr, size * sizeof(int));
                if (tmp != NULL) { arr = tmp; arr[size - 1] = x; }
            }
        }
        if (time) { end = clock(); } //конец таймера
        cout << "Создан новый массив: ";
        outputArr(arr, size);
        cout << "Длина массива – " << size << " элементов.\n";
        if (time) { cout << "Время выполнения (только создание): " <<
((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
        return size;
    }
    return 0;
}

//пункт 1.c по массивам: произвольный размер с файла
int taskArr1c(int*& arr, bool time, int repeat)
{
    clock_t start, end;

```

```

    int x, size = 1;
    ifstream fin("numbers.txt");
    if (!fin.is_open()) { cout << "Файл не был найден.\n"; } // если файл
не открыт, сообщить об этом
    else
    {

        fin >> x;
        if (!fin) { cout << "Ошибка! Новый массив не был создан\n"; }
        else
        {
            if (time) { start = clock(); } //начало таймера
            if (arr) { delete[] arr; }
            arr = createArr(1);
            arr[0] = x;
            while (fin && fin.get() != '\n') //считываются все значения,
пока не встречен мусор
            {
                int* tmp;
                fin >> x;
                if (fin)
                {
                    size++;
                    tmp = (int*)realloc(arr, size * sizeof(int));
                    if (tmp != NULL) { arr = tmp; arr[size - 1] = x; }
                }
            }
            if (time) { end = clock(); } //конец таймера
            cout << "Создан новый массив: ";
            outputArr(arr, size);
            cout << "Длина массива – " << size << " элементов.\n";
            if (time) { cout << "Время выполнения (только создание): " <<
((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
            return size;
        }
    }
    return 0;
}

//3.a - вставка нового элемента
void taskArr3a(int*& arr, bool time, int& size, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Массив не был изменён!\n"; return; }
    if (size < 1) { cout << "Ошибка! Массив не был найден!\n"; return; }
    int x;
    cout << "Введите значение нового элемента: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Массив не был изменён!\n"; cin.clear();
while (cin.get() != '\n'); }

```

```

else
{
    if (time) { start = clock(); } //начало таймера

    for (int i = 0; i < repeat; i++)
    {
        int* tmp;
        size++;
        tmp = (int*)realloc(arr, size * sizeof(int));
        if (tmp != NULL) { arr = tmp; arr[size - 1] = x; }
    }

    if (time) { end = clock(); } //конец таймера
    cout << "Создан новый массив: ";
    outputArr(arr, size);
    cout << "Длина массива – " << size << " элементов.\n";
    if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    return;
}
return;
}

//3ItemInd - получить элемент по индексу
void taskArr3ItemInd(int*& arr, bool time, int& size, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (size < 1) { cout << "Ошибка! Массив не был найден!\n"; return; }
    int x;
    cout << "Введите индекс элемента, который хотите получить (нумерация
элементов с нуля): ";
    cin >> x;
    if (!cin || x >= size || x < 0) { cout << "Ошибка! Элемент не был
найден.\n"; cin.clear(); while (cin.get() != '\n'); }
    else
    {
        if (time) { start = clock(); } //начало таймера

        int* item = nullptr;
        for (int i = 0; i < repeat; i++) { item = &arr[x]; }

        if (time) { end = clock(); } //конец таймера
        if (item != nullptr) { cout << "Получен элемент " << *item << " с
адресом " << item << '\n'; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
        return;
    }
    return;
}

```



```

}

//3Item - получить элемент по значению
void taskArr3Item(int*& arr, bool time, int& size, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (size < 1) { cout << "Ошибка! Массив не был найден!\n"; return; }
    int x;
    cout << "Введите значение элемента, который хотите получить: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear();
while (cin.get() != '\n'); }
    else
    {
        if (time) { start = clock(); } //начало таймера

        int* item = nullptr;
        for (int r = 0; r < repeat; r++)
        {
            for (int i = 0; i < size; i++)
            {
                if (arr[i] == x) { item = &arr[i]; break; }
            }
            if (item == nullptr) { cout << "Не найдено\n"; break; }
        }

        if (time) { end = clock(); } //конец таймера
        if (item != nullptr) { cout << "Получен элемент " << *item << " с
адресом " << item << '\n'; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
    return;
}

//3DelInd - удаление элемента по индексу
void taskArr3DelInd(int*& arr, bool time, int& size, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (size < 1) { cout << "Ошибка! Массив не был найден!\n"; return; }
    int x;
    cout << "Введите индекс, элемент которого хотите удалить (нумерация
элементов с нуля): ";
    cin >> x;
    if (!cin || x >= size || x < 0) { cout << "Ошибка! Элемент не был
найден.\n"; cin.clear(); while (cin.get() != '\n'); }
    else

```

```

{
    if (time) { start = clock(); } //начало таймера

    for (int r = 0; r < repeat; r++)
    {
        if (!cin || x >= size || x < 0) { cout << "Ошибка! Повторение
не удалось, так как такой элемент не был найден.\n"; r = repeat; break; }
        for (int j = x; j < size; j++) { arr[j] = arr[j + 1]; }
        //сдвиг влево на место удаляемого элемента
        int* tmp;
        size--;
        tmp = (int*)realloc(arr, size * sizeof(int));
        if (tmp != NULL) { arr = tmp; }
    }
    if (time) { end = clock(); } //конец таймера

    cout << "Новый массив: ";
    if (size) { outputArr(arr, size); }
    else { cout << "Массив пуст.\n"; }
    cout << "Длина массива – " << size << " элементов.\n";

    if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    return;
}
return;
}

```

//3Del - удаление элемента по значению

//Удаляется только 1 элемент, так как доступна функция повтора

void taskArr3Del(int*& arr, bool time, int& size, int repeat)

```

{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (size < 1) { cout << "Ошибка! Массив не был найден!\n"; return; }
    int x;
    cout << "Введите значение элемента, который хотите удалить: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear(); }
    while (cin.get() != '\n'); }
    else
    {
        bool check; //найден ли элемент?
        if (time) { start = clock(); } //начало таймера

        for (int r = 0; r < repeat; r++)
        {
            check = false;
            for (int i = 0; i < size; i++)
            {

```

```

        if (arr[i] == x)
        {
            for (int j = i; j < size; j++) { arr[j] = arr[j + 1];
        } //сдвиг влево на место удаляемого элемента
            int* tmp;
            size--;
            tmp = (int*)realloc(arr, size * sizeof(int));
            if (tmp != NULL) { arr = tmp; }
            check = true;
            break;
        }
    }
    if (!check) { cout << "Элемент не найден\n"; r = repeat;
break; }
    }
    if (time) { end = clock(); } //конец таймера
    if (check)
    {
        cout << "Элемент удалён\n";
        cout << "Новый массив: ";
        if (size) { outputArr(arr, size); }
        else { cout << "Массив пуст.\n"; }
        cout << "Длина массива – " << size << " элементов.\n\n";
    }

    if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    return;
}
return;
}

// ДВУСВЯЗНЫЙ СПИСОК
struct list
{
    int data;
    list* tail;
    list* head;
};

// == Прочие действия ==
// Вывод списка, начиная с введённого узла
void outputList(list* beg)
{
    if (!beg) { cout << "Список не найден!"; return; }
    list* curr = beg;
    while (curr)
    {
        cout << curr->data << ' ';
        curr = curr->tail;
    }
}

```

```

        cout << '\n';
    }

    // Очистка списка, начиная с введенного узла
    void deleteList(list*& beg)
    {
        list* Next;
        while (beg)
        {
            Next = beg->tail;
            delete beg;
            beg = Next;
        }
    }

    // Длина списка
    int lenList(list* roster)
    {
        size_t len = 0;
        while (roster)
        {
            len++;
            roster = roster->tail;
        }
        return len;
    }

    // == Создание списка размера N. Данные - случайные числа от 0 до 99 ==
    list* createList(int N)
    {
        if (N <= 0) { cout << "Ошибка!\n"; return NULL; }
        list* Curr = 0, //текущий элемент
              * Next = 0; //следующий
        for (int i = 0; i < N; i++) //заполнение списка с конца
        {
            Curr = new list; //новый элемент

            Curr->data = rand() % 100; //заполнение случайным числом от 0 до
99

            Curr->tail = Next; //в адресной части - следующий элемент
            if (Next) //если существует следующий элемент
            {
                Next->head = Curr;
            } //закрепляем прошлый узел с текущим
            Next = Curr; //переходим к следующему элементу
        }

        Curr->head = 0; //его предыдущий адрес должен отсылаться на NULL
        return Curr; //адрес последнего элемента возвращается как адрес пер-
вого

```

```

}

// == Заполнение списка данными. N определяется автоматически ==
//Добавление нового элемента к списку после указанного узла. Не добавляет
элемент в начало списка
list* addItem(list* roster, int a)
{
    list* temp;
    temp = new list; //создание добавляемого узла
    temp->data = a; //заполняем значение

    temp->tail = roster->tail;
    temp->head = roster;
    if (roster->tail) { (roster->tail)->head = temp; }
    roster->tail = temp;

    return temp; //адрес добавленного узла
}

//доступ к заданному элементу списка (по индексу) для манипуляций с его
информационной частью
list* itemList(list* beg, int index)
{
    //index--; //если индексация не с нуля
    while (beg && (index--))
    {
        beg = beg->tail;
        if (!beg) { return 0; } //элемент не существует
    }
    return beg;
}

//удаление узла из списка;
list* deleteItem(list* delItem, list* beg)
{
    list* prev, * next;
    prev = delItem->head; //элемент, предшествующий удаляемому узлу
    next = delItem->tail; //следующий элемент после удаляемого

    //если НЕ первый
    if (prev) { prev->tail = delItem->tail; }
    else //если в начале
    {
        delete delItem;
        if (next) { next->head = prev; return next; } //если следующий
элемент существует - вернуть его как начало
        else { return NULL; } //если нет - список будет удалён
    }
    //если НЕ последний
    if (next) { next->head = delItem->head; }
}

```

```

        delete delItem;
        return beg;
    }

// == ФУНКЦИИ ПО ПУНКТАМ ==
//пункт 1.a по спискам: N случайных чисел
void taskList1a(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    int n;
    cout << "Введите длину нового списка: ";
    while (!(cin >> n) || (n <= 0))
    {
        cout << "Ошибка! Новый список не был создан!\n";
        cin.clear();
        cin.sync();
        while (cin.get() != '\n');
        cout << "Введите длину нового списка: ";
    }
    if (beg) { deleteList(beg); }
    if (time) { start = clock(); }
    for (int r = 0; r < repeat; r++){ beg = createList(n); }
    if (time) { end = clock(); }
    cout << "Создан новый список: ";
    outputList(beg);
    if (time) { cout << "Время выполнения (только создание): " << ((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
}

//пункт 1.b по спискам: произвольный размер
void taskList1b(list*& beg, bool time)
{
    clock_t start, end;
    int x;
    cout << "Введите элементы нового списка: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear(); }
    while (cin.get() != '\n'); }
    else
    {
        if (time) { start = clock(); }
        if (beg) { deleteList(beg); }
        beg = createList(1);
        beg->data = x;
        while (cin && cin.get() != '\n') //считываются все значения, пока
        не встречен мусор
        {
            cin >> x;
            if (cin)
            {
                addItem(beg, x);
            }
        }
    }
}

```

```

        beg = beg->tail;
    }
}
if (time) { end = clock(); }
while (beg->head) { beg = beg->head; }
cout << "Создан новый список: ";
outputList(beg);
cout << "Длина списка – " << lenList(beg) << " элементов.\n";
if (time) { cout << "Время выполнения (только создание): " <<
((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
}
}

//пункт 1.с по спискам: произвольный размер, считывание с файла
void taskList1c(list*& beg, bool time)
{
    clock_t start, end;
    int x;
    ifstream fin("numbers.txt");
    if (!fin.is_open()) { cout << "Файл не был найден.\n"; } // если файл
не открыт, сообщить об этом
    else
    {
        fin >> x;
        if (!fin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear();
while (cin.get() != '\n'); }
        else
        {
            if (time) { start = clock(); }
            if (beg) { deleteList(beg); }
            beg = createList(1);
            beg->data = x;
            while (fin && fin.get() != '\n') //считываются все значения,
пока не встречен мусор
            {
                fin >> x;
                if (fin)
                {
                    addItem(beg, x);
                    beg = beg->tail;
                }
            }
            if (time) { end = clock(); }
            while (beg->head) { beg = beg->head; }
            cout << "Создан новый список: ";
            outputList(beg);
            cout << "Длина списка – " << lenList(beg) << " элементов.\n";
            if (time) { cout << "Время выполнения (только создание): " <<
((double)end - start) / ((double)CLOCKS_PER_SEC) << '\n'; }
            fin.close();
        }
    }
}

```

```

    }

}

//3.a - вставка нового элемента !!после первого элемента
void taskList3a(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Список не был изменён!\n"; return; }
    if (!beg) { cout << "Ошибка! Список не был найден!\n"; return; }
    int x;
    cout << "Введите значение нового элемента: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear(); }
    while (cin.get() != '\n'); }
    else
    {
        if (time) { start = clock(); } //начало таймера

        for (int i = 0; i < repeat; i++)
        {
            addItem(beg, x);
        }

        if (time) { end = clock(); } //конец таймера
        cout << "Создан новый список: ";
        outputList(beg);
        cout << "Длина списка – " << lenList(beg) << " элементов.\n";
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
    return;
}

//3ItemInd - получить элемент по индексу
void taskList3ItemInd(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (!beg) { cout << "Ошибка! Список не был найден!\n"; return; }
    int x;
    cout << "Введите индекс элемента, который хотите получить (нумерация
элементов с нуля): ";
    cin >> x;
    if (!cin || x >= lenList(beg) || x < 0) { cout << "Ошибка! Элемент не
был найден.\n"; cin.clear(); while (cin.get() != '\n'); }
    else
    {

```



```

        if (time) { start = clock(); } //начало таймера

        list* item = nullptr;
        for (int i = 0; i < repeat; i++) { item = itemList(beg, x); }

        if (time) { end = clock(); } //конец таймера
        if (item != nullptr) { cout << "Получен элемент " << item->data
<< " с адресом " << item << '\n'; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
    return;
}

//3Item - получить элемент по значению
void taskList3Item(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (!beg) { cout << "Ошибка! Список не был найден!\n"; return; }
    int x;
    cout << "Введите значение элемента, который хотите получить: ";
    cin >> x;
    if (!cin || x >= lenList(beg) || x < 0) { cout << "Ошибка! Элемент не
был найден.\n"; cin.clear(); while (cin.get() != '\n'); }
    else
    {
        bool check = false; //найден ли элемент?
        if (time) { start = clock(); } //начало таймера

        list* tmp;
        list* item = nullptr;
        for (int r = 0; r < repeat; r++)
        {
            tmp = beg;
            size_t i = 0;
            do
            {
                if (tmp->data == x) { item = tmp; break; }
                tmp = tmp->tail;
                i++;
            } while (tmp);
            if (item == nullptr) { cout << "Не найдено\n"; break; }
        }

        if (time) { end = clock(); } //конец таймера
        if (item != nullptr) { cout << "Получен элемент " << item->data
<< " с адресом " << item << '\n'; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
}

```

```

    }
    return;
}

//3DelInd - удаление элемента по индексу
void taskList3DelInd(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (!beg) { cout << "Ошибка! Список не был найден!\n"; return; }
    int x;
    cout << "Введите индекс элемента, который хотите удалить (нумерация
элементов с нуля): ";
    cin >> x;
    if (!cin || x >= lenList(beg) || x < 0) { cout << "Ошибка! Элемент не
был найден.\n"; cin.clear(); while (cin.get() != '\n'); }
    else
    {
        if (time) { start = clock(); } //начало таймера

        for (int i = 0; i < repeat; i++)
        {
            if (itemList(beg, x)) { deleteItem(itemList(beg, x), beg); }
            else { cout << "Ошибка! Такой элемент не был найден.\n";
break; }
        }

        if (time) { end = clock(); } //конец таймера
        if (beg)
        {
            cout << "Создан новый список: ";
            outputList(beg);
            cout << "Длина списка – " << lenList(beg) << " элементов.\n";
        }
        else { cout << "Список был удалён\n"; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
    return;
}

//3Del - удаление элемента по значению
void taskList3Del(list*& beg, bool time, int repeat)
{
    clock_t start, end;
    bool check = false; //удалён ли элемент
    if (repeat < 1) { cout << "Ошибка! Количество повторений не может
быть меньше 1!\n"; return; }
    if (!beg) { cout << "Ошибка! Список не был найден!\n"; return; }
    int x;

```

```

    cout << "Введите значение элемента, который хотите удалить: ";
    cin >> x;
    if (!cin) { cout << "Ошибка! Некорректный ввод.\n"; cin.clear();
while (cin.get() != '\n'); }
    else
    {
        list* delItem;
        if (time) { start = clock(); } //начало таймера

        for (int i = 0; i < repeat; i++)
        {
            delItem = beg;
            do {
                if (delItem->data == x)
                {
                    beg = deleteItem(delItem, beg);
                    check = true;
                    break;
                }
                delItem = delItem->tail;
            } while (delItem);
            if (!check) { break; }
        }

        if (time) { end = clock(); } //конец таймера
        if (beg)
        {
            if (check)
            {
                cout << "Создан новый список: ";
                outputList(beg);
                cout << "Длина списка – " << lenList(beg) << " элемен-
тов.\n";
            }
            else { cout << "Элемент не был найден\n"; }
        }
        else { cout << "Список был удалён\n"; }
        if (time) { cout << "Время выполнения: " << ((double)end - start)
/ ((double)CLOCKS_PER_SEC) << '\n'; }
    }
    return;
}

int main()
{
    setlocale(0, "");
    bool time = true;
    int repeat = 1; //количество повторений действия

    // ДИНАМИЧЕСКИЙ МАССИВ
    int size = 0;

```

```

int* arr = new int[size]();

// ДВУСВЯЗНЫЙ СПИСОК
list* beg = NULL;

bool check = true; //выход из меню
bool check1 = false; //выход из под меню
bool outp = false;
//false - заканчивает цикл, приводя непосредственно к выходу
do {
    //system("cls");
    char sw = ' '; //переключатель главного меню
    char sw1 = ' '; //переключатель саб-меню
    cout << "\nВыберите нужный раздел: \n";
    cout << "\x1b[32m[1]\x1b[0m Создание целочисленного одномерного
массива\n";
    cout << "\x1b[32m[2]\x1b[0m Работа с элементами массива \n";
    cout << "\x1b[32m[3]\x1b[0m Создание двусвязного списка\n";
    cout << "\x1b[32m[4]\x1b[0m Работа с узлами двусвязного спис-
ка\n";
    cout << "\x1b[32m[5]\x1b[0m Включить/выключить счётчик времени
при выполнении какого-либо действия. Состояние: ";
    if (time) { cout << "\x1b[32mВключено\x1b[0m\n"; }
    else { cout << "\x1b[33mВыключено\x1b[0m\n"; }
    cout << "\x1b[32m[6]\x1b[0m Установить другое число повторов дей-
ствия. Количество повторений сейчас: " << repeat << '\n';
    cout << "\x1b[32m[7]\x1b[0m Очистить экран консоли\n";
    cout << "\x1b[32m[8]\x1b[0m Включить/выключить отображение списка
и массива. Состояние: ";
    if (outp) { cout << "\x1b[32mВключено\x1b[0m\n"; }
    else { cout << "\x1b[33mВыключено\x1b[0m\n"; }
    cout << "\x1b[32m[0]\x1b[0m Закрыть программу\n";
    if (outp)
    {
        cout << "Массив: ";
        if (size) { outputArr(arr, size); }
        else { cout << "Не создан.\n"; }
        cout << "Список: ";
        if (beg) { outputList(beg); }
        else { cout << "Не создан.\n"; }
    }
    cout << "Пожалуйста, введите число, чтобы выполнить нужное дей-
ствие: ";

    cin >> sw;
    while (cin.get() != '\n') { sw = ' '; }; //если строка содержит
более одного символа, возвращается ошибка

    switch (sw)
    {

```

```

case '1': //[1] Создание целочисленного одномерного массива
do {
    check1 = false;
    sw1 = ' ';
    cout << "\n\x1b[32m[1]\x1b[0m Ввести количество элементов
в массиве, чтобы получить массив со случайными числами (от 0 до 99)\n";
    cout << "\x1b[32m[2]\x1b[0m Ввести значение элементов
массива. Его размер определяется автоматически\n";
    cout << "\x1b[32m[3]\x1b[0m Считать массив с файла\n";
    cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
    cout << "Пожалуйста, введите число, чтобы выполнить нуж-
ное действие: ";

    cin >> sw1;
    while (cin.get() != '\n') { sw1 = ' '; };

    switch (sw1)
    {
    case '1': //[1] Случайные числа
        size = taskArr1a(arr, time, repeat);
        break;
    case '2': //[2] Заполнение массива
        size = taskArr1b(arr, time);
        break;
    case '3': //[3] Заполнение массива с файла
        size = taskArr1c(arr, time, repeat);
        break;
    case '0': //[0] Назад
        break;
    default:
        cout << "Ошибка! Пожалуйста, попробуйте снова\n";
        check1 = true; //цикл пойдёт заново
        break;
    }
} while (check1);
break;

case '2': //[2] Работа с элементами массива
do {
    check1 = false;
    sw1 = ' ';
    cout << "\nВыберите, что хотите сделать: \n";
    cout << "\x1b[32m[1]\x1b[0m Вставить новый элемент в мас-
сив\n";
    cout << "\x1b[32m[2]\x1b[0m Удалить элемент массива (по
индексу)\n";
    cout << "\x1b[32m[3]\x1b[0m Удалить элемент массива (по
значению)\n";
    cout << "\x1b[32m[4]\x1b[0m Получить элемент массива по
индексу\n";

```

```

        cout << "\x1b[32m[5]\x1b[0m Получить элемента массива по
значению\n";
        cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
        cout << "Пожалуйста, введите число, чтобы выполнить нуж-
ное действие: ";

        cin >> sw1;
        while (cin.get() != '\n') { sw1 = ' '; };

        switch (sw1)
        {
        case '1': //[1] Вставить новый элемент в массив
            taskArr3a(arr, time, size, repeat);
            break;
        case '2': //[2] Удалить элемент массива (по индексу)
            taskArr3DelInd(arr, time, size, repeat);
            break;
        case '3': //[3] Удалить элемент массива (по значению)
            taskArr3Del(arr, time, size, repeat);
            break;
        case '4': //[4] Получить элемент массива по индексу
            taskArr3ItemInd(arr, time, size, repeat);
            break;
        case '5': //[5] Получить элемента массива по значению
            taskArr3Item(arr, true, size, repeat);
            break;
        case '0': //[0] Назад
            break;
        default:
            cout << "Ошибка! Пожалуйста, попробуйте снова\n";
            check1 = true; //цикл пойдёт заново
            break;
        }
    } while (check1);
    break;

    case '3': //[3] Создание двусвязного списка
        do {
            check1 = false;
            sw1 = ' ';
            cout << "\nВыберите, что хотите сделать: \n";
            cout << "\x1b[32m[1]\x1b[0m Ввести количество элементов в
списке, чтобы получить список со случайными числами (от 0 до 99)\n";
            cout << "\x1b[32m[2]\x1b[0m Ввести значение узлов списка.
Его размер определяется автоматически\n";
            cout << "\x1b[32m[3]\x1b[0m Считать список с файла\n";
            cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
            cout << "Пожалуйста, введите число, чтобы выполнить нуж-
ное действие: ";

```

```

cin >> sw1;
while (cin.get() != '\n') { sw1 = ' '; };

switch (sw1)
{
case '1': //[1] Случайные числа
    taskList1a(beg, time, repeat);
    break;
case '2': //[2] Заполнение списка
    taskList1b(beg, time);
    break;
case '3': //[3] Заполнение списка с файла
    taskList1c(beg, time);
    break;
case '0': //[0] Назад
    break;
default:
    cout << "Ошибка! Пожалуйста, попробуйте снова\n";
    check1 = true; //цикл пойдёт заново
    break;
}
} while (check1);
break;

case '4': //[4] Работа с узлами двусвязного списка
do {
    check1 = false;
    sw1 = ' ';
    cout << "\nВыберите, что хотите сделать: \n";
    cout << "\x1b[32m[1]\x1b[0m Вставить новый элемент в спи-
сок\n";
    cout << "\x1b[32m[2]\x1b[0m Удалить узел списка (по
индексу)\n";
    cout << "\x1b[32m[3]\x1b[0m Удалить узел списка (по
значению)\n";
    cout << "\x1b[32m[4]\x1b[0m Получить элемент списка по
индексу\n";
    cout << "\x1b[32m[5]\x1b[0m Получить элемент списка по
значению\n";
    cout << "\x1b[32m[0]\x1b[0m Вернуться назад\n";
    cout << "Пожалуйста, введите число, чтобы выполнить
нужное действие: ";

    cin >> sw1;
    while (cin.get() != '\n') { sw1 = ' '; };

    switch (sw1)
    {
    case '1': //[1] Вставить новый элемент в массив
        taskList3a(beg, time, repeat);

```

```

        break;
    case '2': //[2] Удалить элемент массива (по индексу)
        taskList3DelInd(beg, time, repeat);
        break;
    case '3': //[3] Удалить элемент массива (по значению)
        taskList3Del(beg, time, repeat);
        break;
    case '4': //[4] Получить элемент списка по индексу
        taskList3ItemInd(beg, time, repeat);
        break;
    case '5': //[5] Получить элемент списка по значению
        taskList3Item(beg, time, repeat);
        break;
    case '0': //[0] Назад
        break;
    default:
        cout << "Ошибка! Пожалуйста, попробуйте снова\n";
        check1 = true; //цикл пойдёт заново
        break;
    }
} while (check1);
break;

```

```

    case '5': //[5] Включить/отключить счётчик времени при выполнении
какого-либо действия
        time = !time;
        if (time) { cout << "Счётчик времени теперь включён.\n"; }
        else { cout << "Счётчик времени теперь отключён.\n"; }
        break;

```

```

    case '6': //[6] Установить другое число повторов действия. По
умолчанию равен 1
        do
        {
            int x = 0;
            check1 = false;
            cout << "Введите число повторений (выше нуля): ";

            cin >> x;
            while (cin.get() != '\n');

            if (!cin || x <= 0) { cout << "Ошибка! Количество
повторений не может быть меньше 1. Значение не было изменено.\n";
                check1 = true; }
            else { repeat = x; }
        } while (check1);
        break;

```

```

    case '7': //[7] Очистить экран
        system("cls");

```



```

        break;

        case '8': //[5] Включить/отключить счётчик времени при выполнении
какого-либо действия
            outp = !outp;
            if (outp) { cout << "Отображение списка и массива
включено.\n"; }
            else { cout << "Отображение списка и массива отключено.\n"; }
            break;

        case '0': //[0] Закрыть программу
            cout << "Выход из программы...\n";
            check = false; //выход из цикла
            break;
        default: //в случае, если введено что-то иное
            cout << "Ошибка! Пожалуйста, попробуйте снова\n";
            break;
    }

} while (check);

if (size) { delete[] arr; }
deleteList(beg);
system("Pause");
return 0;
}

```