**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: Динамические массивы, списки.

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Михеева Э. Д. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**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.**

## Динамические объекты

В C++ можно использовать различные типы объектов, которые различаются по использованию памяти. Так, глобальные объекты создаются при запуске программы и освобождаются при ее завершении. Локальные автоматические объекты создаются в блоке кода и удаляются, когда этот блок кода завершает работу. Локальные статические объекты создаются перед их первым использованием и освобождаются при завершении программы.

Глобальные, а также статические локальные объекты помещаются в статической памяти, а локальные автоматические объекты размещаются в стеке. Объекты в статической памяти и стеке создаются и удаляются компилятором. Статическая память очищается при завершении программы, а объекты из стека существуют, пока выполняется блок, в котором они определены.

В дополнение к этим типам в C++ можно создавать динамические объекты. Продолжительность их жизни не зависит от того, где они созданы. Динамические объекты существуют, пока не будут удалены явным образом. Динамические объекты размещаются в динамической памяти.

Для управления динамическими объектами в С++ применяются операторы **new** и **delete**.

Оператор **new** выделяет место в динамической памяти для объекта и возвращает указатель на этот объект.

Оператор **delete** получает указатель на динамический объект и удаляет его из памяти.

Создание динамического объекта:

int \*ptr = new int;

Оператор **new** создает новый объект типа int в динамической памяти и возвращает указатель на него. Значение такого объекта неопределенно.

Также можно инициализировать объект при создании:

int \*p1 = new int(); // значение по умолчанию - 0

std::cout << "p1: " << \*p1 << "\n"; // 0

int \*p2 = new int(12);

std::cout << "p2: " << \*p2 << "\n"; // 12

## Освобождение памяти

Динамические объекты будут существовать пока не будут явным образом удалены. И после завершения использования динамических объектов следует освободить их память с помощью оператора **delete**:

int \*p1 = new int(12);

std::cout << "p1: " << \*p1 << "\n"; // 0

delete p1;

Особенно это надо учитывать, если динамический объект создается в одной части кода, а используется в другой. Например:

#include <iostream>

int\* createPtr(int value)

{

int \*ptr = new int(value);

return ptr;

}

void usePtr()

{

int \*p1 = createPtr(10);

std::cout << \*p1 << "\n"; // 10

delete p1; // объект надо освободить

}

int main()

{

usePtr();

return 0;

}

 В функции **usePtr** получаем из функции **createPtr** указатель на динамический объект. Однако после выполнения функции **usePtr** этот объект автоматически не удаляется из памяти (как это происходит в случае с локальными автоматическими объектами). Поэтому его надо явным образом удалить, использовав оператор **delete**.

Использование объекта по указателю после его удаления или повторное применение оператора **delete** к указателю могут привести к непредсказуемым результатам:

int \*p1 = new int(12);

std::cout << \*p1 << std::"\n"; // 0

delete p1;

// ошибочные сценарии

std::cout << \*p1 << std::"\n"; // объект по указателю p1 уже удален!

delete p1; // объект по указателю p1 уже удален!

Поэтому следует удалять объект только один раз.

Также нередко имеет место ситуация, когда на один и тот же динамический объект указывают сразу несколько указателей. Если оператор **delete** применен к одному из указателей, то память объекта освобождается, и по второму указателю этот объект  использовать уже невозможно. Если же после этого ко второму указателю применить оператор **delete**, то динамическая память может быть нарушена.

В то же время недопустимость указателей после применения к ним оператора **delete** не означает, что эти указатели невозможно использовать. Их можно использовать, если присвоить им адрес другого объекта:

#include <iostream>

int main()

{

int \*p1 = new int(12);

int \*p2 = p1;

delete p1; // адреса в p1 и p2 недопустимы

p1 = new int(11); // p1 указывает на новый объект

std::cout << \*p1 << "\n"; // 11

delete p1;

return 0;

}

Здесь после удаления объекта, на который указывает **p1**, этому указателю передается адрес другого объекта в динамической памяти. Соответственно мы также можем использовать указатель **p1**. В то же время адрес в указателе **p2** по прежнему будет недействительным.

## Одномерные динамические массивы

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 используется следующий синтаксис инструкции new (стиль С++):

int \*Arr = new int [100];

Причем в этом случае оператор new также возвращает указатель на объект типа int - первый элемент в созданном массиве.

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

delete [] Arr;

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

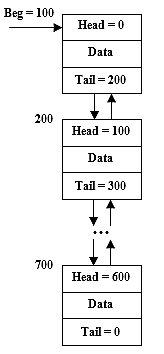
{

int data;

list \*head;

list \*tail;

};



 Поле  **Head** содержит адрес предыдущего элемента, поле **Tail** содержит адрес следующего элемента списка. Такая организация списка позволяет перемещаться по его элементам в двух направлениях.

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.**

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.

**Полный код программы:**

#include <iostream>  
#include <sstream>  
#include <chrono>  
#include <Windows.h>  
using namespace std;  
  
struct List{  
 int data;  
 List \*next;  
 List \*prev;  
};  
  
int checkInput(){  
 int input;  
 try{  
 cin >> input;  
 if (cin.fail()) throw 1;  
 } catch (int exeption){  
 cout << "! ерор ошибка";  
 exit(0);  
 }  
 cin.sync();  
 cout << "\n";  
 return input;  
}  
  
void printList(List \* cur){  
 cout << "список: ";  
 while(cur){  
 cout << cur->data << " ";  
 cur = cur->next;  
 }  
}  
  
void printArr(int \* arr, int size){  
 cout << "массив: ";  
 for (int \*i = arr; i != arr + size; i++) cout << \*i << " ";  
}  
  
void deleteList (List \* &list){  
 List \*Next;  
 while (list){  
 Next = list->next;  
 delete list;  
 list = Next;  
 }  
}  
  
List \*createListRand(unsigned size){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List \*head = nullptr, \*tail = nullptr;  
 srand(time(NULL));  
 for (int i = 0; i < size; i++){  
 head = new List;  
 head->data = rand() % 100;  
 head->next = tail;  
 if(tail) tail->prev = head;  
 tail = head;  
 }  
 if(size != 0) head->prev = nullptr;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "\nвремя, потраченное на создание\nсписка: " << end - start << " нс\n";  
 return head;  
}  
  
int \*createArrRand(unsigned size){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int \*arr = new int[size];  
 srand(time(NULL));  
 for (int i = size - 1; i >= 0; i--) arr[i] = rand() % 100;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "массива: " << end - start << " нс\n";  
 return arr;  
}  
  
List \*createListByHands(string str, int &size){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List \*head = nullptr, \*tail = nullptr;  
 istringstream iss(str);  
 int num, count = 0;  
 while (iss >> num) {  
 head = new List;  
 head->data = num;  
 head->prev = tail;  
 if(tail) tail->next = head;  
 tail = head;  
 count++;  
 }  
 size = count;  
 if(size > 0) {  
 head->next = nullptr;  
 while (head->prev) head = head->prev;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "\nвремя, потраченное на создание\nсписка: " << end - start << " нс\n";  
 return head;  
 }  
 else return nullptr;  
}  
  
int \*createArrByHands(string str, int size){  
 if (size == 0) return nullptr;  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int \*arr = new int[size];  
 istringstream iss(str);  
 int num, i = 0;  
 while (iss >> num){  
 arr[i] = num;  
 i++;  
 }  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "массива: " << end - start << " нс\n";  
 return arr;  
}  
  
void getLine(string &stringLine){getline(cin, stringLine);}  
  
void createAll(List \* &list, int \* &arr, int &size){  
 cout << "способ заполнения массива:\n1) случайными значениями\n2) вручную\n\nвыберите: ";  
 int inputType = checkInput();  
 int sizeList;  
 string stringList;  
 if (inputType == 1){  
 cout << "введите размер: ";  
 sizeList = checkInput();  
 if (sizeList <= 0){  
 cout << "некорректное значение, попробуйте снова\n";  
 createAll(list, arr, size);  
 }  
 size = sizeList;  
 list = createListRand(size);  
 arr = createArrRand(size);  
 }  
 else if(inputType == 2){  
 cout << "введите значения (через пробел): ";  
 getLine(stringList);  
 list = createListByHands(stringList, size);  
 arr = createArrByHands(stringList, size);  
 }  
 else{  
 cout << "что-то не то ввели, попробуйте снова\n";  
 createAll(list, arr, size);  
 }  
 cout << "\n";  
 printList(list);  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\nдлина списка и массива: " << size;  
}  
  
List \*listItem(List \* Beg, unsigned int index){  
 int p = 0;  
 while(p != index){  
 Beg = Beg->next;  
 p++;  
 }  
 return Beg;  
}  
  
void addValue(List \* &list, int \* &arr, int &size){  
 cout << "\nвведите, на место какого индекса вставить значение: ";  
 int index = checkInput();  
 if (!(index >= 1 && index <= size + 1)){  
 cout << "\nнет такого индекса, попробуйте снова: ";  
 index = checkInput();  
 }  
 cout << "\nвведите значение, которое хотите добавить: ";  
 int value = checkInput();  
 List \*addItem = new List;  
 addItem->data = value;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int \*arr1 = new int[size+1];  
 for(int i = 0, j = 0; i < size + 1; i++, j++){  
 if(i == index - 1){  
 arr1[i] = value;  
 j--;  
 }  
 else arr1[i] = arr[j];  
 }  
 delete []arr;  
 arr = arr1;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на добавление элемента\nв массив: " << end - start << " нс\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if(index > size){  
 List \*item = listItem(list, size-1);  
 item->next = addItem;  
 addItem->prev = item;  
 addItem->next = nullptr;  
 }  
 else if(index == 1){  
 list->prev = addItem;  
 addItem->next = list;  
 addItem->prev = nullptr;  
 list = addItem;  
 }  
 else{  
 List \*item = listItem(list, index-1);  
 addItem->next = item;  
 addItem->prev = item->prev;  
 item->prev->next = addItem;  
 item->prev = addItem;  
 }  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "в список: " << end - start << " нс\n";  
 size++;  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\n";  
 printList(list);  
 cout << "\nдлина списка и массива: " << size;  
}  
  
void changeValues(List \* &list, int \* &arr, int size){  
 cout << "\nвведите индексы, которые нужно поменять местами";  
 cout << "\nиндекс 1: ";  
 int index1 = checkInput();  
 cout << "индекс 2: ";  
 int index2 = checkInput();  
 if (!((index1 >= 1 && index1 <= size) && (index2 >= 1 && index2 <= size))){  
 cout << "\nнет такого индекса, попробуйте снова\n";  
 return;  
 }  
 if (index1 == index2){  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\n";  
 printList(list);  
 return;  
 }  
 index1--; index2--;  
 if(index1 > index2) swap(index1, index2);  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 swap(arr[index2], arr[index1]);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на обмен элементов\nв массиве: " << end - start << " нс\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List \*item1 = listItem(list, index1);  
 List \*item2 = listItem(list, index2);  
 if (item1->prev != nullptr) item1->prev->next = item2;  
 else list = item2;  
 if (item1->next != item2) item1->next->prev = item2;  
 if (item2->prev != item1) item2->prev->next = item1;  
 if (item2->next != nullptr) item2->next->prev = item1;  
 if(item1->next == item2){  
 item2->prev = item1->prev;  
 item1->prev = item2;  
 item1->next = item2->next;  
 item2->next = item1;  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "в списке: " << end - start << " нс\n";  
 return;  
 }  
 List \*help = item1->prev;  
 item1->prev = item2->prev;  
 item2->prev = help;  
 help = item1->next;  
 item1->next = item2->next;  
 item2->next = help;  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "в списке: " << end - start << " нс\n";  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\n";  
 printList(list);  
}  
  
void deleteValue(List \* &list, int \* &arr, int &size){  
 cout << "\nспособ удаления элемента\n1) по значению\n2) по индексу\n\nвыберите: ";  
 int type, value, index, count = 0;  
 List \*cur = list;  
 type = checkInput();  
 switch (type) {  
 case 1:  
 cout << "введите значение: ";  
 value = checkInput();  
 break;  
 case 2:  
 cout << "введите индекс: ";  
 index = checkInput();  
 while (!(index >= 1 && index <= size)){  
 cout << "\nнет такого индекса, попробуйте снова: ";  
 index = checkInput();  
 }  
 while(cur){  
 if(count == index - 1){  
 value = cur->data;  
 break;  
 }  
 cur = cur->next;  
 count++;  
 }  
 break;  
 default:  
 cout << "\nчто-то не то ввели, попробуйте снова\n";  
 deleteValue(list, arr, size);  
 }  
 count = 0;  
 cur = list;  
 List \*help = nullptr;  
 bool flag = false;  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 while(cur){  
 if (cur->data == value){  
 flag = true;  
 if (cur->prev == nullptr){  
 list = cur->next;  
 if(cur->next != nullptr)cur->next->prev = nullptr;  
 delete cur;  
 cur = list;  
 }  
 else if(cur->next == nullptr){  
 help = cur->next;  
 if(cur->prev != nullptr) cur->prev->next = nullptr;  
 delete cur;  
 cur = help;  
 }  
 else{  
 help = cur->next;  
 cur->prev->next = cur->next;  
 cur->next->prev = cur->prev;  
 delete cur;  
 cur = help;  
 }  
 count++;  
 if(type == 2) break;  
 }  
 else cur = cur->next;  
 }  
 if(!flag) return;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на удаление элемента\nв списке: " << end - start << " нс\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int \*arr1 = new int[size-1];  
 flag = true;  
 for(int i = 0, j = 0; i < size; i++, j++){  
 if(arr[i] == value && flag){  
 j--;  
 if(type == 2) flag = false;  
 }  
 else arr1[j] = arr[i];  
 }  
 delete []arr;  
 arr = arr1;  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "в массиве: " << end - start << " нс\n";  
 size -= count;  
 cout << "\n";  
 printList(list);  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\nдлина списка и массива: " << size;  
}  
  
void getValue(List \* &list, int \* &arr, int size){  
 cout << "\nспособ получения элемента\n1) по значению\n2) по индексу\n\nвыберите: ";  
 int type, value, index, count = 0;  
 bool flag = false;  
 List \*cur = list;  
 type = checkInput();  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 switch (type) {  
 case 1: {  
 cout << "введите значение: ";  
 value = checkInput();  
 bool bebra = false;  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 for (int i = 0; i < size; i++){  
 if (arr[i] == value) {  
 bebra = true;  
 cout << "элемент " << value << " найден, его индекс: " << i + 1 << "\n";  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на нахождение элемента в массиве: " << end - start << " нс\n";  
 }  
 }  
 if (!bebra){  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "массив не содержит элемент " << value << "\n";  
 cout << "время, потраченное на нахождение элемента в массиве: " << end - start << " нс\n";  
 }  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 while (cur){  
 if (cur->data == value){  
 flag = true;  
 index = count;  
 cout << "\nэлемент " << value << " найден, его индекс: " << index + 1 << "\n";  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на нахождение элемента в списке: " << end - start << " нс\n";  
 }  
 cur = cur->next;  
 count++;  
 }  
 if (!flag) {  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "список не содержит элемент " << value << "\n";  
 cout << "время, потраченное на нахождение элемента в списке: " << end - start << " нс\n";  
 }  
 break;  
 }  
 case 2: {  
 cout << "введите индекс: ";  
 index = checkInput();  
 while (!(index >= 1 && index <= size)) {  
 cout << "нет такого индекса, попробуйте снова: ";  
 index = checkInput();  
 }  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(  
 std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "элемент с индексом " << index << " найден, его значение: " << arr[index - 1] << "\n";  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(  
 std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на поиск элемента в массиве: " << end - start << " нс\n";  
  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(  
 std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cur = listItem(list, index - 1);  
 cout << "\nэлемент с индексом " << index << " найден, его значение: " << cur->data << "\n";  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(  
 std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "время, потраченное на поиск элемента в списке: " << end - start << " нс\n";  
 break;  
 }  
 default: {  
 cout << "что-то не то ввели, попробуйте снова\n";  
 getValue(list, arr, size);  
 break;  
 }  
 }  
 cout << "\n";  
 printArr(arr, size);  
 cout << "\n";  
 printList(list);  
}  
  
void idz(List \* &list, int \* &arr, int &size){  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 int count = 0;  
 for(int i = 0; i < size; i++) if (arr[i] % 2 != 0) count++;  
 int \*arr1 = new int[size - count];  
 for(int i = 0, j = 0; i < size; i++, j++){  
 if(arr[i] % 2 != 0) j--;  
 else arr1[j] = arr[i];  
 }  
 size -= count;  
 delete []arr;  
 arr = arr1;  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "\nвремя, потраченное на удаление нечетных элементов в массиве: " << end - start << " нс\n";  
 printArr(arr, size);  
 start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List \*cur = list, \*help = nullptr;  
 while(cur){  
 if (cur->data % 2 != 0){  
 if (cur->prev == nullptr){  
 list = cur->next;  
 if(cur->next != nullptr)cur->next->prev = nullptr;  
 delete cur;  
 cur = list;  
 }  
 else if(cur->next == nullptr){  
 help = cur->next;  
 if(cur->prev != nullptr) cur->prev->next = nullptr;  
 delete cur;  
 cur = help;  
 }  
 else{  
 help = cur->next;  
 cur->prev->next = cur->next;  
 cur->next->prev = cur->prev;  
 delete cur;  
 cur = help;  
 }  
 }  
 else cur = cur->next;  
 }  
 end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << "\nвремя, потраченное на удаление нечетных элементов в списке: " << end - start << " нс\n";  
 printList(list);  
}  
  
int main() {  
 SetConsoleOutputCP(CP\_UTF8);  
 int kek(1), command;  
 int size;  
 int \*arr = nullptr;  
 List \*list = nullptr;  
 createAll(list, arr, size);  
 cout << "\n\n\tменю:\n1. идз\n2. вставить элемент в список\n3. поменять элементы местами\n4. удалить элемент списка\n5. найти элемент в списке\n";  
 while (kek) {  
 cout << "\nвведите номер пункта: ";  
 cin >> command;  
 switch(command) {  
 case 1:{  
 idz(list, arr, size);  
 }break;  
  
 case 2:{  
 addValue(list, arr, size);  
 }break;  
  
 case 3:{  
 changeValues(list, arr, size);  
 }break;  
  
 case 4:{  
 deleteValue(list, arr, size);  
 }break;  
  
 case 5:{  
 getValue(list, arr, size);  
 }break;  
 }  
  
 cout << "\n\nпродолжить выполнение? (1 - да, 0 - нет)" << endl;  
 cin >> kek;  
 }  
 delete []arr;  
 deleteList(list);  
 return 0;  
}

**Вывод:**

Проведя тесты на большом количестве элементов, я выяснила, что списки работают быстрее при удалении и вставке элемента, а динамический массив лучше справляется с нахождением элемента по индексу и обмену элементов. Это обуславливается разной работай с памятью каждого из типов хранения данных. Поэтому при работе с данными, где постоянно необходимо удалять и добавлять элементы лучше подойдут списки, например, очередь или стек. Динамические массивы лучше подойдут статическим данным, где необходимо в основном брать элементы из массива, например, статическая база данных.