**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: “Двусвязные списки, динамические массивы”

|  |  |  |
| --- | --- | --- |
| Студент гр. 2372 |  | Симаков Г.О. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение внутреннего строения двусвязного списка и строения динамического массива, понимания, как они хранятся в памяти.

**Основные теоретические положения.**

*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;

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

*Двусвязные списки*

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.

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.

*Сравнение*

Сравним производительность различных операций для списков с массивами. Основное преимущество массивов – возможность обратиться к произвольному элементу по его индексу. Для списка подобная операция потребует просмотра всех предшествующих элементов. Однако добавление в начало и середину массива, а также удаление из начала и середины требует сдвига всех элементов массива, расположенных после вставляемого (удаляемого). Для списка же производительность операций добавления и удаления не зависит ни от длины списка, ни от места вставки.

Таким образом списки работают быстрее при удалении и вставки 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.

Также, при выборе структуры следует учитывать, что для организации односвязного списка на каждый элемент требуется одно поле связи, занимающее 4 байта

**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.

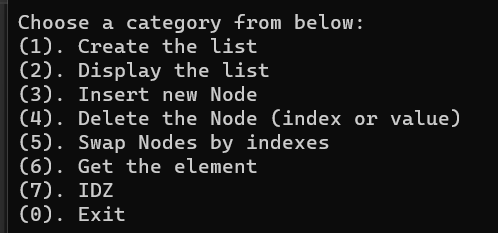
4. Определение скорости вставки, удаление и получения элемента двусвязного списка п. 3.

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 должны принимать одинаковые значения.

**Выполнение работы.**

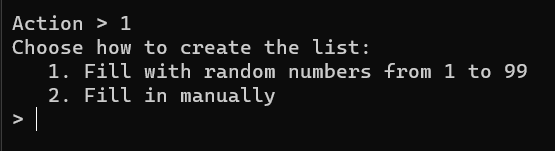
Код программы представлен в приложении А.

1. При запуске программы пользователю выводится меню доступных функций. Ожидается ввод команды с клавиатуры.

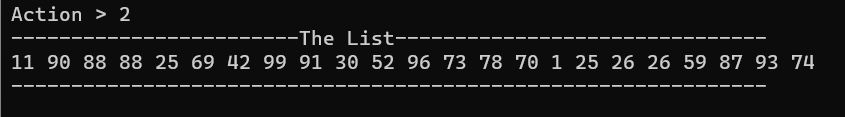


1. Далее, если пользователь ввёл:

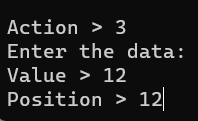
“1”, - открывается меню создания списка.



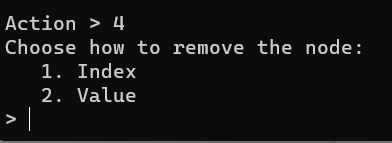
“2”, - выводится список на экран.



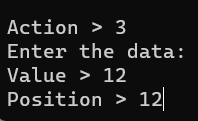
“3”, - открывается меню вставки элемента в двусвязный список.



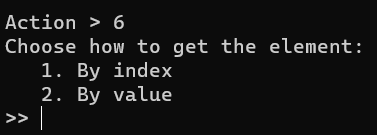
“4”, - открывается меню удаления элемента по (индексу/значению).



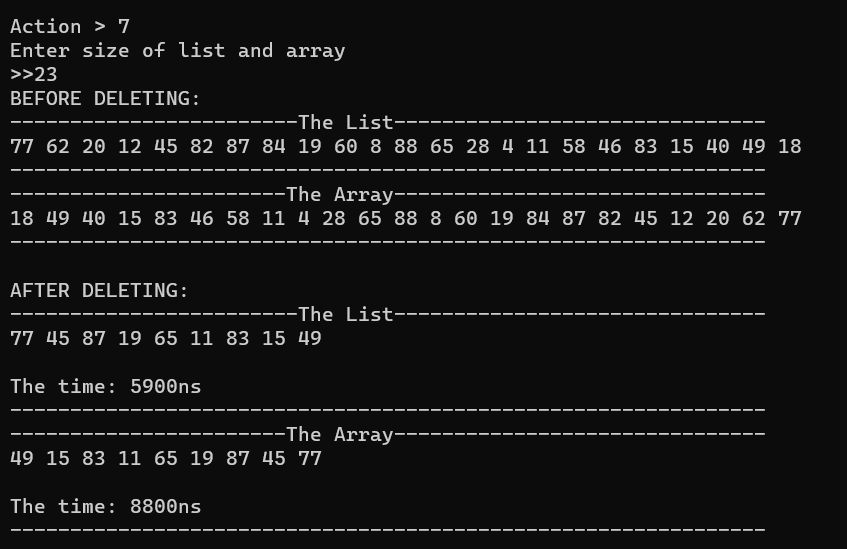
“5”, - открывается меню обмена элементов местами.



“6”, - открывается меню получения элемента.



“7”, - открывается меню ИДЗ(№8).



“0”, - программа завершается.

Приложение А

рабочий код

#include <iostream>  
#include <cstdlib>  
#include <iomanip>  
#include <chrono>  
#include <vector>  
#include <algorithm>  
#include <iterator>  
  
using namespace std;  
struct List //описание узла списка  
{  
 int data; //информационное поле  
 List\* next; //указатель на следующий элемент  
 List\* prev; //указатель на предыдущий элемент  
};  
  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CREATE A LIST WITH N RANDOM ELEMENTS(0, 99)\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
struct List\* createList(unsigned lenght) {  
 if (lenght == 0) return nullptr; // if length == 0 -> the list is not created  
  
 struct List\* Current = nullptr; // Адрес очередного элемента списка  
 struct List\* Next = nullptr; // Адрес следующего элемента списка  
 srand(time(nullptr));  
 // Creates DLL starting from the LAST node  
 for (int i = 0; i < lenght; ++i) {  
 Current = new List;  
 Current->data = rand() % 100;  
  
 Current->next = Next;  
 if (Next) Next->prev = Current;  
 Next = Current;  
 }  
 Current->prev = nullptr;  
 // Возвращаем адрес последнего созданного элемента,  
 // как адрес первого элемента списка  
 return Current; // Return head  
  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FREE THE MEMORY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
void DeleteList(List\*& head) {  
 List\* temp;  
 while (head)  
 {  
 temp = head->prev;  
 delete head;  
 head = temp;  
 }  
}  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*CREATE A LIST FROM USER INPUT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
struct List\* createListFromInput()  
{  
 List\* Head = nullptr;  
 List\* Tail = nullptr;  
  
 int userValue = 1;  
  
 while (userValue != 0) {  
 // Create NEW Node  
 List\* Current = new List;  
 cin >> userValue;  
 if (userValue != 0) {  
 Current->data = userValue;  
 Current->next = nullptr;  
  
 if (Tail) { // If node isn't the head, then update the pointers  
 Tail->next = Current;  
 Current->prev = Tail;  
 }  
 else { // If tail is null, then it is the first element; update the head  
 Current->prev = nullptr;  
 Head = Current;  
 }  
 Tail = Current;  
 }  
 }  
 return Head;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIND NODE BY INDEX\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
struct List\* getItemByIndex(struct List\* head, unsigned position) {  
 if (!head) { //  
 cout << "List is empty\n";  
 return nullptr;  
 }  
  
  
 for (unsigned i = 0; i < position; ++i) {  
 if (!head->next) {  
 cout << "Index out of range\n";  
 return nullptr;  
 }  
 head = head->next;  
 }  
  
 return head;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*FIND NODE BY VALUE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
struct List\* getItemByValue(struct List\* head, int value) {  
 if (!head) { // Empty list check  
 cout << "List is empty\n";  
 return nullptr;  
 }  
  
  
 while (head) {  
 if (head->data == value) break;  
  
 if (!head->next) {  
 cout << "Value not found\n";  
 return nullptr;  
 }  
 head = head->next;  
 }  
  
 return head;  
}  
  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*INSERT NODE AT THE SPECIFIED POSITION\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
bool AddList(List\*& head, int value, unsigned position) {  
  
 struct List\* temp = getItemByIndex(head, position);  
 if (!temp) {  
 return false;  
 }  
  
 // Insert the new node  
 List\* newNode = new List;  
 newNode->data = value;  
 newNode->prev = temp->prev;  
 newNode->next = temp;  
 if (newNode->prev) newNode->prev->next = newNode;  
 if (newNode->next) newNode->next->prev = newNode;  
 if (position == 0) head = newNode;  
  
 return true;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DELETE THE NODE WITH A SPECIFIED VALUE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
bool deleteItemByValue(struct List\*& head, int value) {  
 if (!head) { //Empty list check  
 return false;  
 }  
  
 // Find the node  
 struct List\* temp = getItemByValue(head, value);  
 if (!temp) return false;  
  
 // Delete the node  
 if (temp->next != nullptr)  
 {  
 temp->next->prev = temp->prev;  
 }  
  
 if (temp->prev != nullptr)  
 {  
 temp->prev->next = temp->next;  
 }  
 else  
 {  
 head = temp->next;  
 }  
  
 delete temp;  
 return true;  
}  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DELETE THE NODE WITH A SPECIFIED INDEX\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
bool deleteItemByIdx(struct List\*& head, unsigned position) {  
 if (!head) { //Empty list check  
 return false;  
 }  
  
 // Find the node  
 struct List\* temp = getItemByIndex(head, position);  
 if (!temp) return false;  
  
 // Delete the node  
 if (temp->next != nullptr)  
 {  
 temp->next->prev = temp->prev;  
 }  
  
 if (temp->prev != nullptr)  
 {  
 temp->prev->next = temp->next;  
 }  
 else  
 {  
 head = temp->next;  
 }  
  
 delete temp;  
 return true;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SWAP NODES BY SPECIFIED INDEXES\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
bool swapByIdx(List\*& head, unsigned x, unsigned y)  
{  
  
 // Edge Cases  
 if (head == nullptr || head->next == nullptr  
 || x == y)  
 return false;  
 if (x > y) swap(x, y);  
  
 // Finding the Nodes  
  
 List\* Node1 = getItemByIndex(head, x);  
 List\* Node2 = getItemByIndex(head, y);  
  
 // Swapping Node1 and Node2  
  
 if (Node1 == head) head = Node2;  
  
 List\* temp;  
 temp = Node1->next;  
 Node1->next = Node2->next;  
 Node2->next = temp;  
  
 if (Node1->next != nullptr)  
 Node1->next->prev = Node1;  
 if (Node2->next != nullptr)  
 Node2->next->prev = Node2;  
  
 temp = Node1->prev;  
 Node1->prev = Node2->prev;  
 Node2->prev = temp;  
  
 if (Node1->prev != nullptr) {  
 Node1->prev->next = Node1;  
 }  
 if (Node2->prev != nullptr) {  
 Node2->prev->next = Node2;  
 }  
  
 return true;  
}  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*DISPLAY THE LIST\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
void printList(List\* head) {  
 if (!head) {  
 cout << "List is empty\n";  
 return;  
 }  
 List\* temp = head;  
  
 // Iterate until node is NOT NULL  
 while (temp) {  
 cout << temp->data << " ";  
 temp = temp->next;  
 }  
 cout << endl;  
}  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IDZ #8\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
struct List\* remove\_even(List\* head)  
{  
 while (head && (head->data % 2) == 0)  
 {  
 auto\* p = head;  
 head = head->next;  
 delete p;  
 }  
 if (!head)  
 return nullptr;  
  
 for (auto\* p = head; p->next; )  
 if ((p->next->data % 2) != 0)  
 p = p->next;  
 else  
 {  
 auto\* p2 = p->next;  
 p->next = p2->next;  
 delete p2;  
 }  
  
 return head;  
}  
  
  
  
  
  
  
  
//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*MAIN MENU\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
int main()  
{  
 system("chcp 65001");  
  
 struct List\* head = nullptr;  
 List\* ptr = head;  
  
 int x, length;  
 unsigned position;  
 while (true)  
 {  
  
 cout << "\nChoose a category from below:\n";  
 cout << "(1). Create the list" << endl;  
 cout << "(2). Display the list" << endl;  
 cout << "(3). Insert new Node" << endl;  
 cout << "(4). Delete the Node (index or value)" << endl;  
 cout << "(5). Swap Nodes by indexes" << endl;  
 cout << "(6). Get the element" << endl;  
 cout << "(7). IDZ" << endl; //IDZ #9  
 cout << "(0). Exit" << endl;  
 cout << "\nAction > "; cin >> x;  
  
 if (cin.fail() || cin.get() != '\n') {  
 cout << "Bad entry. Enter a NUMBER:\n";  
 cin.clear();  
 cin.ignore(100000, '\n');  
 continue;  
 }  
  
 switch (x) {  
 case 0: {  
 if (head) {  
 DeleteList(head);  
 }  
 return 0;  
 }  
 case 1: {  
 cout << "Choose how to create the list:\n";  
 cout << " 1. Fill with random numbers from 1 to 99\n";  
 cout << " 2. Fill in manually\n> ";  
 int key;  
 cin >> key;  
 if (key != 2 && key != 1) {  
 cout << "Invalid input\n";  
 continue;  
 }  
 if (key == 1) {  
 cout << "Enter size of list\n>>";  
 cin >> length;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 head = createList(length);  
 if (!head) {  
 cout << "The list is empty.\n";  
 continue;  
 }  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of creating list: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 }  
 else {  
 cout << "Enter value (To stop enter 0):\n";  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 head = createListFromInput();  
  
 if (!head) {  
 cout << "The list is empty.\n";  
 continue;  
 }  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of creating list: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
  
 }  
 break;  
 }  
 case 2: {  
 cout << setw(32) << setfill('-') << "The List" << setw(32) << setfill('-') << '\n';  
 printList(head);  
 cout << setw(64) << setfill('-') << '\n';  
 break;  
 }  
 case 3: {  
 if (!head) {  
 cout << "The list is empty.\n";  
 continue;  
 }  
 cout << "Enter the data: \n";  
 int value;  
 cout << "Value > ";  
 cin >> value;  
 cout << "Position > ";  
 cin >> position;  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (!AddList(head, value, position)) {  
 continue;  
 };  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of inserting node: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 break;  
 }  
 case 4: {  
 if (!head) {  
 cout << "List is empty\n";  
 continue;  
 }  
  
 cout << "Choose how to remove the node:\n";  
 cout << " 1. Index\n";  
 cout << " 2. Value\n> ";  
 int input;  
 cin >> input;  
 if (input != 1 && input != 2) {  
 cout << "Invalid input\n";  
 continue;  
 }  
  
 if (input == 1) {  
 int index;  
 cout << "Enter index:\n> ";  
 cin >> index;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (!deleteItemByIdx(head, index)) {  
 continue;  
 };  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of deleting node: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
  
 }  
 else {  
 int value;  
 cout << "Enter value:\n>> ";  
 cin >> value;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (!deleteItemByValue(head, value)) {  
 continue;  
 };  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of deleting node: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 }  
  
 break;  
 }  
 case 5: {  
 if (!head) {  
 cout << "List is empty\n";  
 continue;  
 }  
 cout << "Select nodes to swap:\n";  
 unsigned idx1, idx2;  
 cout << "Index 1 > "; cin >> idx1;  
 cout << "Index 2 > "; cin >> idx2;  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (!swapByIdx(head, idx1, idx2)) {  
 continue;  
 };  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of swapping nodes: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 break;  
 }  
 case 6: {  
 if (!head) {  
 cout << "List is empty\n";  
 continue;  
 }  
  
 cout << "Choose how to get the element:\n";  
 cout << " 1. By index\n";  
 cout << " 2. By value\n>> ";  
 int input;  
 cin >> input;  
 if (input != 2 && input != 1) {  
 cout << "Invalid input\n";  
 continue;  
 }  
  
 if (input == 1) {  
 int index;  
 cout << "<< Enter index:\n>> ";  
 cin >> index;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List\* temp = getItemByIndex(head, index);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (temp == 0) {  
 cout << "Item whit index ( " << index << " ) ... wasn't founded" << endl;  
 }  
 else {  
 cout << "Item whit index ( " << index << " ) = " << temp->data << endl;  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of getting item by index: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 }  
  
 }  
 else {  
 int value;  
 cout << "<< Enter value:\n>> ";  
 cin >> value;  
  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 List\* temp = getItemByValue(head, value);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 if (temp == 0) {  
 cout << "Item whit value ( " << value << " ) ... wasn't founded" << endl;  
 }  
 else {  
 cout << "The address of the element with value (" << value << " ) = " << temp << endl;  
 cout << setw(32) << setfill('-') << "Time" << setw(32) << setfill('-') << '\n';  
 cout << "Duration of getting item by value: " << end - start << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
 }  
 }  
 break;  
 }  
 case 7: {  
 cout << "Enter size of list and array\n>>";  
 cin >> length;  
 head = createList(length);  
 if (!head) {  
 cout << "The list is empty.\n";  
 continue;  
 }  
  
 cout << "BEFORE DELETING: \n";  
 cout << setw(32) << setfill('-') << "The List" << setw(32) << setfill('-') << '\n';  
 printList(head);  
 cout << setw(64) << setfill('-') << '\n';  
  
 srand(time(nullptr));  
 vector<int> arr(length);  
 for (int i = 0; i < length; i++)  
 {  
 arr[i] = (rand() % 100);  
 }  
  
 cout << setw(32) << setfill('-') << "The Array" << setw(32) << setfill('-') << '\n';  
 std::copy(arr.begin(), arr.end(), std::ostream\_iterator<int>(std::cout, " "));  
 cout << endl << setw(64) << setfill('-') << '\n';  
  
 cout << endl << "AFTER DELETING: \n";  
 auto start = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 remove\_even(head);  
 auto end = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 cout << setw(32) << setfill('-') << "The List" << setw(32) << setfill('-') << '\n';  
 printList(head);  
 cout << endl << "The time: " << end - start << "ns" << endl;  
 cout << setw(64) << setfill('-') << '\n';  
  
 auto start2 = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 arr.erase(std::remove\_if(arr.begin(), arr.end(), [](const int& x) {  
 return x % 2 == 0;  
 }), arr.end());  
 auto end2 = std::chrono::duration\_cast<std::chrono::nanoseconds>(std::chrono::system\_clock::now().time\_since\_epoch()).count();  
 // вывод содержимого вектора в std::cout  
 cout << setw(32) << setfill('-') << "The Array" << setw(32) << setfill('-') << '\n';  
 std::copy(arr.begin(), arr.end(), std::ostream\_iterator<int>(std::cout, " "));  
 cout << endl << endl << "The time: " << end2 - start2 << "ns";  
 cout << endl << setw(64) << setfill('-') << '\n';  
  
 DeleteList(head);  
 arr.clear();  
 break;  
  
 }  
  
 default:  
 cout << "\nCategory with number " << x << " does not exist." << endl;  
 break;  
 }  
  
 }  
}