**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

Тема: Линейные структуры данных.

|  |  |  |
| --- | --- | --- |
| Студентка гр. 2373 |  | Басова К.В. |
| 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.**

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

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 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.

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.

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.

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.

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.

**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.

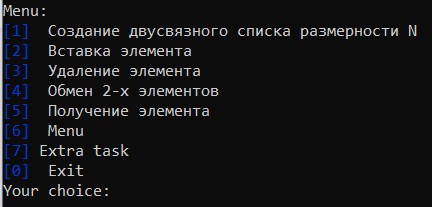
4.  Определение скорости вставки, удаление и получения элемента двусвязного списка п. 3

**Выполнение работы.**

Код программы представлен в приложении А.

Блок описания кода и использованных алгоритмов

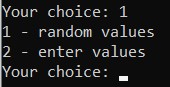
1. При запуске программы появляется окно, в котором выводится меню допустимых операций под номерами. Пользователю предлагается ввести любой номер операции от 0 до 7:



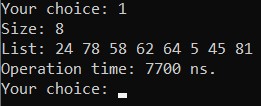
2. При попытке ввода символа, не входящего в диапазон выбора, пользователю предлагается ввести номер ещё раз:

Скриншот 31-05-2023 202023

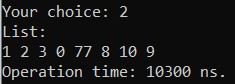
3. Операция 1 – создание двусвязного списка размерности N. Пользователю предлагается выбрать способ создания списка.



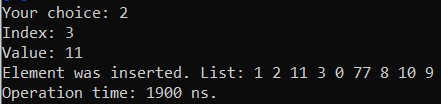
Для создания списка из случайных чисел небходимо ввести его размер. Полученный список выводится на экран.



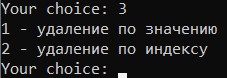
Другой способ - ввод значений элементов с клавиатуры. Размер списка определяется автоматически по количеству введённых чисел.



1. Операция 2 – вставка элемента. Пользователю необходимо внести индекс и значение вставляемого элемента. Полученный список выводится на экран.

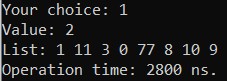


1. Операция 3 – удаление элемента. Пользователю предлагается выбрать способ удаления - по индексу или по значению.

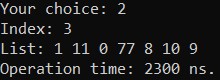


После удаления элемента полученный список выводится на экран.

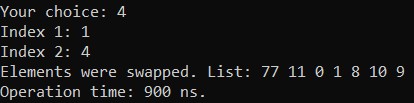
Удаление по значению:



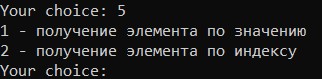
Удаление по индексу:



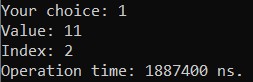
1. Операция 4 - обмен двух элементов. Пользователю необходимо ввести их индексы. Изменённый список выводится на экран.



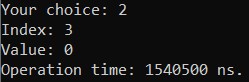
1. Операция 5 - получение элемента. Пользователю предлагается выбрать способ получения - по значению или по индексу.



Получение элемента по значению:



Получение элемента по индексу:



1. Операция 6 – повторный вывод меню на экран, 0 – выход из программы.

**Выводы.**

Связный список — базовая динамическая структура данных, состоящая из узлов, содержащих данные и ссылки («связки») на следующий и/или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями.

Приложение А

рабочий код

#include <iostream>

#include <algorithm>

#include <string>

#include <cstring>

#include <Windows.h>

#include <chrono>

using namespace std;

using namespace chrono;

struct Node {

int x;

Node\* Next, \* Prev;

};

class List {

Node\* Head, \* Tail; //Указатели, начало списка и последний элемент списка (голова и хвост)

int count;

public:

List() :Head(NULL), Tail(NULL), count(0) {}

~List();

void add(int); //Прототип функции добавления элемента в список, добавляется целое int

int deleteX(int, int); // Функция удаления элемента

void deleteAll(); // Функция удаления всех элементов

void insert(int, int); // Вставка элемента

void getX(int, int); // Получение элемента

void swapX(int, int); // Обмен элементами

void swapValue(int, int);

void show(); //Вывод каждого элемента на экран

void sort(); // Сортировка

void reverse(); // Обратный порядок

};

void List::add(int value) {

Node\* temp = new Node; //Выделяем память для звена

temp->x = value; //Записываем значение в звено списка

temp->Next = NULL; //Следующего элемента пока ещё нет

if (!Head) { //Если Head==NULL, то

temp->Prev = NULL; //Предыдущий элемент указывает в пустоту

Head = Tail = temp; //Голова тот же же элемент, что и хвост

}

else {

temp->Prev = Tail;

Tail->Next = temp;

Tail = temp; //Меняем адрес хвоста

}

count++;

}

int List::deleteX(int value, int choice) {

Node\* temp = Head;

int i = 1;

if (choice == 1) { // Если удаляем по значению

while (temp){

if (temp->x == value)

break;

temp = temp->Next;

i++;

}

value = i; // Переходим к его индексу

}

if ((value < 1) or (value > count)) {

cout << "Element wasn't found.\n";

return -1;

}

if ((value == 1) && (Head->Next)) { // Если нужный элемент - первый

temp = Head;

Head = Head->Next;

Head->Prev = NULL;

delete temp;

count--;

return 1;

}

else if ((value == 1) && (Head == Tail)) { // Если нужный элемент - единственный в списке

Head->Next = NULL;

Head = NULL;

delete Head;

count = 0;

return 1;

}

if ((value == count)) { // Если нужный элемент - последний

temp = Tail;

Tail = Tail->Prev;

Tail->Next = NULL;

delete temp;

count--;

return 1;

}

temp = Head;

Node \* temp2;

for (int k = 0; k < value - 1; k++)

temp = temp->Next;

temp2 = temp;

temp2->Prev->Next = temp->Next;

temp2->Next->Prev = temp->Prev;

delete temp;

count--; // Уменьшаем кол-во элементов на 1

return 1;

}

void List::deleteAll() { // Удаление всех элементов

Node\* temp;

while (count > 1) { // Пока не останется один элемент

temp = Tail;

Tail = Tail->Prev;

Tail->Next = NULL;

delete temp;

count--;

}

Head = Tail = NULL;

count = 0;

}

void List::insert(int insertIndex, int insertValue) { // Вставка элемента

Node\* temp = new Node;

if (insertIndex == 1) { // Если вставляем в начало списка

temp->Prev = NULL;

temp->x = insertValue;

temp->Next = Head;

if (Head != NULL) // Если элемент не единственный

Head->Prev = temp;

if (count == 0) // Если элемент единственный

Head = Tail = temp;

else

Head = temp;

count++;

return;

}

if (insertIndex == count + 1) { // Если вставляем в конец списка

temp->Next = NULL;

temp->x = insertValue;

temp->Prev = Tail;

if (Tail != NULL) // Если элемент не единственный

Tail->Next = temp;

if (count == 0) // Если элемент единственный

Tail = Head = temp;

else

Tail = temp;

count++; // Увеличиваем кол-во элементов на 1

return;

}

if (insertIndex < 1 || insertIndex > count + 1) { // Введено недопустимое значение индекса

cout << "Invalid index.\n";

return;

}

int i = 1;

Node\* ins = Head;

while (i < insertIndex) { // Находим нужный индекс

ins = ins->Next;

i++;

}

Node\* prevIns = ins->Prev;

temp->x = insertValue;

if (prevIns != NULL && count != 1)

prevIns->Next = temp;

temp->Next = ins;

temp->Prev = prevIns;

ins->Prev = temp;

count++;

}

void List::getX(int value, int choice) { // Получение элемента

Node\* temp = Head;

if (choice == 1) { // Получение по значению

int i = 1, found = 0;

while (temp) {

if (temp->x == value) { // Находим элемент с нужным значением

found++;

if (found == 1)

cout << "Index: "; // Выводим индекс элемента

cout << i << " ";

}

temp = temp->Next;

i++;

}

if (found == 0) // Если элемент не найден

cout << "Element wasn't found.";

cout << "\n";

}

if (choice == 2) { // Получение по индексу

int i = 1;

if ((value < 1) || (value > count)) { // Введено недопустимое значение индекса

cout << "Incorrect index.\n";

}

while (i < value) { // Пока не дойдём до элемента с нужным индексом

temp = temp->Next;

i++;

}

cout << "Value: " << temp->x << "\n"; // Выводим значение элемента

}

}

void List::swapX (int index1, int index2) { // Обмен элементами

Node\* prev1, \* prev2, \* next1, \* next2;

int i = 1;

Node\* elem1 = Head, \* elem2 = Head;

while (i < index1) { // Пока не найдём элемент с первым индексом

elem1 = elem1->Next;

i++;

}

while (i < index2) { // Пока не найдём элемент со вторым индексом

elem2 = elem2->Next;

i++;

}

prev1 = elem1->Prev;

prev2 = elem2->Prev;

next2 = elem2->Next;

next1 = elem1->Next;

if (elem2 == next1) // Обмениваются соседние узлы

{

elem2->Next = elem1;

elem2->Prev = prev1;

elem1->Next = next2;

elem1->Prev = elem2;

if (next2 != NULL)

next2->Prev = elem1;

if (elem1 != Head)

prev1->Next = elem2;

}

else // Обмениваются отстоящие узлы

{

if (elem1 != Head)

prev1->Next = elem2;

elem2->Next = next1;

elem1->Next = next2;

elem2->Prev = prev1;

if (next2 != NULL)

next2->Prev = elem1;

elem1->Prev = prev2;

if (next1 != NULL)

next1->Prev = elem2;

}

}

void List::swapValue(int index1, int index2) { // Обмен значений элементов

int i = 1, swapi;

Node\* elem1 = Head, \* elem2 = Head;

while (i < index1) { // Находим элемент с первым индексом

elem1 = elem1->Next;

i++;

}

while (i < index2) { // Находим элемент со вторым индексом

elem2 = elem2->Next;

i++;

}

swapi = elem1->x;

elem1->x = elem2->x;

elem2->x = swapi;

}

void List::show() { // Вывод элементов списка

Node\* temp = Head; //Направляем указатель на начало списка

while (temp) { //Обходим список

cout << temp->x << ' '; //Выводим элементы

temp = temp->Next;

}

cout << "\n";

}

void List::sort() { // Сортировка

Node\* left = Head; //Первый элемент — это голова

Node\* right = Head->Next; //Второй элемент — это следующий за головой элемент

Node\* temp = new Node;

while (left->Next) { //Обходим по всем звеньям, за исключением крайнего правого

while (right) { //Обходим по всем звеньям, включая крайний правый

if ((left->x) > (right->x)) { //Проверяем необходимость перестановки

temp->x = left->x; //И переставляем элементы, за исключением указателей связи, местами

left->x = right->x;

right->x = temp->x;

}

right = right->Next;

}

left = left->Next;

right = left->Next; //Возвращаем второй указатель назад

}

}

List::~List() {

while (Head) //Пока по адресу на начало списка что-то есть

{

Tail = Head->Next; //Резервная копия адреса следующего звена списка

delete Head; //Очистка памяти от первого звена

Head = Tail; //Смена адреса начала на адрес следующего элемента

}

}

void Menu();

int main()

{

SetConsoleCP(1251); // Установка русского языка

SetConsoleOutputCP(1251);

List lst;

int size, sizeMas = 0;

string menu\_n;

int\* mas = new int[100];

int created = 0;

time\_point<steady\_clock, duration<\_\_int64, ratio<1, 1000000000>>> start, end;

nanoseconds result;

Menu(); // Вывод меню

cout << "Your choice: ";

cin >> menu\_n;

while (menu\_n != "0"){

if (menu\_n == "1") {

int create\_n;

cout << "1 - random values\n2 - enter values\nYour choice: ";

cin >> create\_n;

if (create\_n == 1) {

if (created == 1) {

lst.deleteAll();

for (int i = sizeMas-1; i >= 0; i--)

mas[i] = NULL;

}// Очищаем список и массив

while (true) {

cout << "Size: ";

if (cin >> size) // Вводим размер списка

{

if (size < 0) { // Недопустимое значение

cout << "Invalid number. Try again\nSize: ";

cin >> size;

}

else if (size == 0) // Пустой список

{

cout << "The list is empty";

break;

}

else {

sizeMas = size;

start = steady\_clock::now();

for (int k = 0; k < size; k++)

mas[k] = (rand() % 100);

for (int k = 0; k < size; k++)

lst.add(mas[k]); // Добавление в список

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "List: ";

lst.show();

cout << "Operation time: " << result.count() << " ns.\n";

created = 1;

break;

}

}

else // Недопустимое значение

{

cout << "Invalid input. Try again.";

}

}

}

if (create\_n == 2) {

if (created == 1) {

lst.deleteAll();

for (int i = sizeMas - 1; i >= 0; i--)

mas[i] = NULL;

}// Очищение списка

char s[256];

int number = 0;

bool found = false;

int ind = 0;

cin.get();

cout << "List: \n";

cin.getline(s, 256); // Читаем введённую строку

start = steady\_clock::now();

for (int i = 0; i < strlen(s); i++) { // Обработка строки

char ch = s[i];

if (ch >= '0' && ch <= '9') {

int digit = ch - '0';

number = number \* 10 + digit;

found = true;

}

else {

if (found) { // Запись элементов в массив

mas[ind] = number;

number = 0;

found = false;

ind++;

}

}

if (found)

mas[ind] = number;

}

created = 1;

sizeMas = ind + 1;

for (int k = 0; k < sizeMas; k++) // Добавление элементов в список

lst.add(mas[k]);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Operation time: " << result.count() << " ns.\n";

}

}

else if (created == 0)

cout << "Create the list first.\n";

else if (menu\_n == "2") { // Вставка элемента

int insertIndex, insertValue;

cout << "Index: ";

cin >> insertIndex;

cout << "Value: ";

cin >> insertValue;

if ((insertIndex <= sizeMas + 1) && (insertIndex >= 1)) {

for (int k = sizeMas; k >= insertIndex; k--) {

mas[k] = mas[k - 1];

}

mas[insertIndex - 1] = insertValue;

}

sizeMas++;

start = steady\_clock::now();

lst.insert(insertIndex, insertValue);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Element was inserted. List: ";

lst.show(); // Вывод списка

cout << "Operation time: " << result.count() << " ns.\n";

}

else if ((sizeMas == 0) && (menu\_n == "3" || menu\_n == "4" || menu\_n == "5")) {

cout << "The list is empty.\n ";

}

else if (menu\_n == "3") { // Удаление элемента

int delete\_n, deleteIndex, deleteValue;

int delTime = 0;

cout << "1 - удаление по значению \n2 - удаление по индексу\n";

cout << "Your choice: ";

cin >> delete\_n;

if (delete\_n == 1) { // Удаление по значению

cout << "Value: ";

cin >> deleteValue;

int k = 0, need = -1;

while (k != sizeMas) {

if (mas[k] == deleteValue)

need = k;

k++;

}

if (need != -1) {

for (int l = need; l < sizeMas; l++) {

mas[l] = mas[l + 1];

}

}

sizeMas--;

start = steady\_clock::now();

lst.deleteX(deleteValue, 1);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

delTime = result.count();

}

if (delete\_n == 2) { // Удаление по индексу

cout << "Index: ";

cin >> deleteIndex;

int k = deleteIndex - 1;

if ((mas[k] != NULL) && (deleteIndex <= sizeMas) && (deleteIndex >= 1)) {

for (int l = k; l < sizeMas; l++) {

mas[l] = mas[l + 1];

}

}

sizeMas--;

start = steady\_clock::now();

lst.deleteX(deleteIndex, 2);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

delTime = result.count();

}

cout << "List: ";

lst.show();

cout << "Operation time: " << delTime << " ns.\n";

}

else if (menu\_n == "4") { // Обмен элементами

int index1, index2, temp;

cout << "Index 1: ";

cin >> index1;

cout << "Index 2: ";

cin >> index2;

if ((index1 <= sizeMas) && (index1 >= 1) && (mas[index1-1] != NULL)) {

if ((index2 <= sizeMas) && (index2 >= 1) && (mas[index2-1] != NULL)) {

swap(mas[index1 - 1], mas[index2 - 1]);

}

}

start = steady\_clock::now();

lst.swapValue(index1, index2);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Elements were swapped. List: ";

lst.show();

cout << "Operation time: " << result.count() << " ns.\n";

}

else if (menu\_n == "5") { // Получение элемента

int get\_n, getElem;

cout << "1 - получение элемента по значению \n2 - получение элемента по индексу\n";

cout << "Your choice: ";

cin >> get\_n;

if (get\_n == 1) { // Получение индекса элемента по его значению

cout << "Value: ";

cin >> getElem;

start = steady\_clock::now();

lst.getX(getElem, 1);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Operation time: " << result.count() << " ns.\n";

}

if (get\_n == 2) { // Получение значения элемента по его индексу

cout << "Index: ";

cin >> getElem;

start = steady\_clock::now();

lst.getX(getElem, 2);

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

cout << "Operation time: " << result.count() << " ns.\n";

}

}

else if (menu\_n == "6") { // Вывод меню

Menu();

}

else if (menu\_n == "7") { // Доп.задание

int timeMas, timeList;

cout << "Удалить чётные элементы.\n";

cout << "List: ";

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

cout << mas[i] << " ";

cout << "\n";

lst.show();

start = steady\_clock::now();

for (int ind = 1; ind < sizeMas; ind++) {

for (int l = ind; l < sizeMas - 1; l++){

mas[l] = mas[l + 1];

}

sizeMas--;

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

timeMas = result.count();

start = steady\_clock::now();

for (int ind = 2; ind <= sizeMas; ind++) {

lst.deleteX(ind, 2);

}

end = steady\_clock::now();

result = duration\_cast<nanoseconds>(end - start);

timeList = result.count();

cout << "New List: ";

lst.show();

cout << "Operation time (Array): " << timeMas << " ns.\n";

cout << "Operation time (List): " << timeList << " ns.\n";

}

else { // Недопустимое значение

cout << "Invalid input. Try again\n";

}

cout << "Your choice: ";

cin >> menu\_n;

}

return 0;

}

void Menu() { // Вывод меню

cout << "Menu:\n" << "\x1b[34m" << "[1] " << "\x1b[0m" << "Создание двусвязного списка размерности N" << "\n";

cout << "\x1b[34m" << "[2] " << "\x1b[0m" << "Вставка элемента\n";

cout << "\x1b[34m" << "[3] " << "\x1b[0m" << "Удаление элемента\n";

cout << "\x1b[34m" << "[4] " << "\x1b[0m" << "Обмен 2-х элементов\n";

cout << "\x1b[34m" << "[5] " << "\x1b[0m" << "Получение элемента\n";

cout << "\x1b[34m" << "[6] " << "\x1b[0m" << "Menu\n";

cout << "\x1b[34m" << "[7] " << "\x1b[0m" << "Extra task\n";

cout << "\x1b[34m" << "[0] " << "\x1b[0m" << "Exit\n";

}