**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по практической работе №2**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2373 |  | Чекстер А. А. |
| 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.

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.

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

**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.

**Блок описания кода и использованных алгоритмов:**

1. Для создания такого списка можно использовать функцию
2. Удаление первого элемента и последнего элемента практически аналогично удалению элемента из односвязного списка. Нужно изменить значение указателя на первый элемент, обнулить значение указателя головы/хвоста, освободить динамическую память.

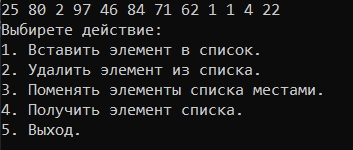
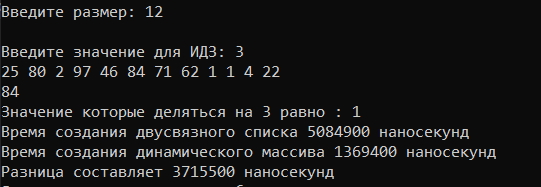
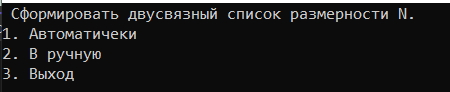
С удалением элемента из середины списка дело обстоит сложнее: необходимо проделать аналогичные операции по отношению к двум узлам, а не к одному.

1. Вставка нового узла в двусвязный линейный список проводится аналогично. Также присутствует три ситуации: вставка нового корня, вставка последним элементом и вставка узла в середину списка.
2. Больше всего трудностей может возникнуть именно с взаимообменом элементов двусвязного списка. Главная проблема – возможная путаница в указателях. Нужно правильно изменить указатели всех затрагиваемых узлов.

**Выводы.**

Мы научились изучению 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.

**Блок скриншотов работы программы**



Приложение А

рабочий код

#include <iostream>

#include <ctime>

#include <chrono>

using namespace std;

using namespace chrono;

class Node {

public:

int data;

int index;

Node\* prev, \* next;

public:

Node(int data) {

this->data = data;

this->prev = this->next = NULL;

}

};

class LinkedList {

public:

Node\* head, \* tail;

public:

LinkedList() {

head = tail = NULL;

}

~LinkedList() {

while (head != NULL)

pop\_front();

}

Node\* push\_front(int data) {

Node\* ptr = new Node(data);

ptr->next = head;

if (head != NULL)

head->prev = ptr;

if (tail == NULL)

tail = ptr;

head = ptr;

return ptr;

}

Node\* push\_back(int data) {

Node\* ptr = new Node(data);

ptr->prev = tail;

if (tail != NULL)

tail->next = ptr;

if (head == NULL)

head = ptr;

tail = ptr;

return ptr;

}

void pop\_front() {

if (head == NULL) return;

Node\* ptr = head->next;

if (ptr != NULL)

ptr->prev = NULL;

else

tail = NULL;

delete head;

head = ptr;

}

void pop\_back() {

if (tail == NULL) return;

Node\* ptr = tail->prev;

if (ptr != NULL)

ptr->next = NULL;

else

head = NULL;

delete tail;

tail = ptr;

}

Node\* getAt(int index) {

Node\* ptr = head;

int n = 0;

while (n != index) {

if (ptr == NULL)

return ptr;

ptr = ptr->next;

n++;

}

return ptr;

}

Node\* operator [] (int index) {

return getAt(index);

}

Node\* insert(int index, int data) {

Node\* right = getAt(index);

if (right == NULL)

return push\_back(data);

Node\* left = right->prev;

if (left == NULL)

return push\_front(data);

Node\* ptr = new Node(data);

ptr->prev = left;

ptr->next = right;

left->next = ptr;

right->prev = ptr;

return ptr;

}

void earase(int index) {

Node\* ptr = getAt(index);

if (ptr == NULL)

return;

if (ptr->prev == NULL) {

pop\_front();

return;

}

if (ptr->next == NULL) {

pop\_back();

return;

}

Node\* left = ptr->prev;

Node\* right = ptr->next;

left->next = right;

right->prev = left;

}

};

int main()

{

setlocale(0, "");

int N;

int choose;

LinkedList lst;

Menu: {

cout << " Сформировать двусвязный список размерности N.\n" <<

"1. Автоматичеки\n" <<

"2. В ручную\n" <<

"3. Выход\n";

}

cin >> choose;

switch (choose) {

case 1: {

system("cls");

srand(time(0));

int N, Count = 0, k;

cout << "\nВведите размер: ";

cin >> N;

cout << "\nВведите значение для ИДЗ: ";

cin >> k;

auto time1 = steady\_clock::now();

for (int i = 1; i <= N; i++) {

lst.push\_back(rand() % 100);

}

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next) {

cout << ptr->data << " ";

}

cout << "\n";

auto time1\_1 = steady\_clock::now();

for (int j = 0; j < N; j++) {

if (lst[j]->data % k == 0) {

Count++;

cout << lst[j]->data << " ";

}

}

int\* Idz = new int[Count];

for (int p = 0; p < N; p++) {

if (lst[p]->data % k == 0) {

for (int c = 0; c < Count; c++) {

Idz[c] = lst[p]->data;

//cout << Idz[c] << " ";

}

}

}

cout << "\nЗначение которые деляться на " << k << " равно : " << Count << endl;

auto time2\_1 = steady\_clock::now();

auto time2 = steady\_clock::now();

cout << "Время создания двусвязного списка " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

cout << "Время создания динамического массива " << duration\_cast<nanoseconds>(time2\_1 - time1\_1).count() << " наносекунд\n";

cout << "Разница составляет " << (duration\_cast<nanoseconds>(time2 - time1).count()) - (duration\_cast<nanoseconds>(time2\_1 - time1\_1).count()) << " наносекунд\n";

system("pause");

break;

}

case 2: {

system("cls");

int N, Count = 0, k, count = 0;

bool flag = true;

cout << "\nВведите значение для ИДЗ: ";

cin >> k;

cout << "\nВведите список: \n";

cout << "\nДля завершения ввода, введите '0': ";

auto time1 = steady\_clock::now();

while (flag == true) {

cin >> N;

if (N != 0) {

count++;

lst.push\_back(N);

if (N % k == 0) {

Count++;

}

}

else

flag = false;

}

//cout << count;

auto time1\_1 = steady\_clock::now();

int\* Idz1 = new int[Count];

for (int p = 0; p < count; p++) {

if (lst[p]->data % k == 0) {

for (int c = 0; c < Count; c++) {

Idz1[c] = lst[p]->data;

cout << Idz1[c] << " ";

break;

}

}

}

cout << "\nЗначение которые деляться на " << k << " равно : " << Count << endl;

auto time2\_1 = steady\_clock::now();

cout << endl;

auto time2 = steady\_clock::now();

cout << "Время создания " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

cout << "Время создания динамического массива " << duration\_cast<nanoseconds>(time2\_1 - time1\_1).count() << " наносекунд\n";

cout << "Разница составляет " << (duration\_cast<nanoseconds>(time2 - time1).count()) - (duration\_cast<nanoseconds>(time2\_1 - time1\_1).count()) << " наносекунд\n";

system("pause");

break;

}

case 3: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

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

system("pause");

goto Menu;

}

}

Menu2: {

system("cls");

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next) {

cout << ptr->data << " ";

}

cout << "\nВыбирете действие:\n"

<< "1. Вставить элемент в список.\n"

<< "2. Удалить элемент из списка.\n"

<< "3. Поменять элементы списка местами.\n"

<< "4. Получить элемент списка.\n"

<< "5. Выход.\n";

}

cin >> choose;

switch (choose) {

case 1: {

system("cls");

int i;

float n;

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << "\nВведите индекс элемента, на место которого хотите вставить новый элемент: ";

cin >> i;

cout << "\nВведите значение элемента: ";

cin >> n;

auto time1 = steady\_clock::now();

lst.insert(i, n);

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << endl;

auto time2 = steady\_clock::now();

cout << "Время вставки " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

system("pause");

goto Menu2;

}

case 2: {

system("cls");

float i;

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << endl;

cout << "\nВведите индекс элемента, который хотите удалить: ";

cin >> i;

auto time1 = steady\_clock::now();

lst.earase(i);

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << endl;

auto time2 = steady\_clock::now();

cout << "Время вставки " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

system("pause");

goto Menu2;

}

case 3: {

system("cls");

int n, f;

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << "\nВведите индекс первого элемента: ";

cin >> n;

cout << "Введите индекс второго элемента: ";

cin >> f;

swap(lst[n]->data, lst[f]->data);

auto time1 = steady\_clock::now();

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

auto time2 = steady\_clock::now();

cout << "Время вставки " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

system("pause");

goto Menu2;

}

case 4: {

system("cls");

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

int N;

cout << "\nВведите индекс значения: ";

cin >> N;

auto time1 = steady\_clock::now();

cout << lst[N]->data << endl;

for (Node\* ptr = lst.head; ptr != NULL; ptr = ptr->next)

cout << ptr->data << " ";

cout << endl;

auto time2 = steady\_clock::now();

cout << "Время вставки " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

system("pause");

goto Menu2;

}

case 5: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

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

system("pause");

goto Menu2;

}

}

}