# ЛАБОРАТОРНАЯ РАБОТА №1

# ИССЛЕДОВАНИЕ ОСОБЕННОСТЕЙ ОБЪЕКТНОГО ПОДХОДА ПРИ ПРОГРАММИРОВАНИИ СЛОЖНЫХ СТРУКТУР ДАННЫХ С ДИНАМИЧЕСКИМИ ПОЛЯМИ

Цель работы

Исследование основных средств определения класса, создания объектов класса, приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей использования конструкторов копирования.

Задания

1. Изучить основные средства языка C++ для определения класса, создания объектов класса, работы с подобными объектами и их уничтожения после использования;
2. Разработать программу на языке C++ согласно варианту 11 методических указаний (Рисунок 1). Работу программы реализовать в виде меню;

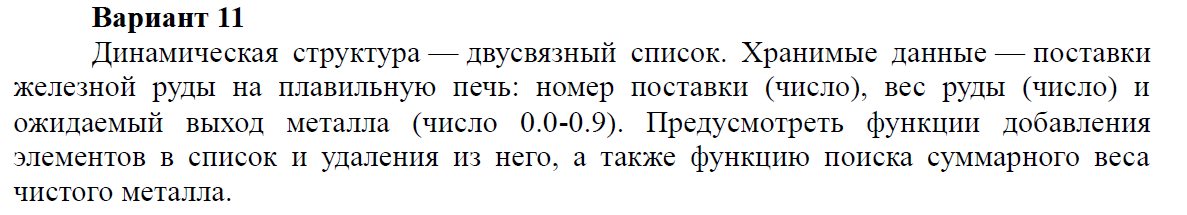


Рисунок 1 – Вариант задания

1. Разработать тестовые примеры и выполнить тестирование и отладку написанной программы;
2. Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы, проанализировав преимущества и недостатки структурного и объектного подхода в применении к этой задаче;

Текст программы

#include <iostream>

#include <conio.h>

#include <Windows.h>

using namespace std;

// Данные о поставке руды (инф. поле элемента списка)

class Data {

public:

int num; // номер поставки

float weight; // вес руды

float result; // ожидаемый выход металла

Data(int \_n = 0, float \_w = 0, float \_r = 0) { // конструктор

num = \_n;

weight = \_w;

result = \_r;

}

};

// Структура элемента списка

struct Node {

Node \*next, \*prev; // указатели на след. и пред. элементы

Data inf; // инф. поле элемента списка

};

// Список элементов с данными о поставках руды

class List {

Node \*head, \*tail;

public:

List() : head(NULL), tail(NULL) {}; // конструктор

void Add(); // добавление элементов

void Show(); // просмотр списка

void Delete(); // удаление элемента

float ClearWeight(); // поиск сумм. веса

List(const List& obj); // конструктор копирования

~List(); // деструктор

};

// Конструктор копирования

List::List(const List& obj) {

if (!obj.head)

cout << "\n\n ОШИБКА: Список отсутствует\n";

else {

head = tail = new Node; // копирование первого элемента

head->inf = obj.head->inf;

head->next = head->prev = NULL;

Node\* curr = obj.head->next; // копирование остальных элементов

while (curr) {

Node\* newNode = new Node;

newNode->inf = curr->inf;

newNode->prev = tail;

newNode->next = NULL;

tail->next = newNode;

tail = newNode;

curr = curr->next;

}

cout << "\n\n Список успешно скопирован \n";

}

return;

}

// Деструктор

List::~List() {

while (head) {

Node \*curr = head;

head = head->next;

delete curr;

}

}

// Добавление элементов в список

void List::Add() {

system("cls");

cout << "\n Добавление записей \n\n";

while (1) {

Node\* node = new Node; // заполнение нового элемента

cout << " Номер поставки - ";

cin >> node->inf.num;

cout << " Вес руды - ";

cin >> node->inf.weight;

cout << " Выход металла - ";

cin >> node->inf.result;

node->next = NULL;

if (!head) { // вставка эл-та в голову списка

node->prev = NULL;

head = tail = node;

}

else { // вставка эл-та в конец списка

node->prev = tail;

tail->next = node;

tail = node;

}

// Выбор продолжить или завершить ввод данных

cout << "\n [0] Завершить ввод данных\n";

cout << " [1] Продолжить ввод данных\n\n";

cout << " Ожидание комманды... \n\n";

while (1) {

char c = \_getch();

if (c == '1') break; // продолжить

if (c == '0') return; // завершить

}

}

}

// Просмотр списка

void List::Show() {

if (!head)

cout << "\n\n ОШИБКА: Список отсутствует\n";

else {

system("cls");

cout << "\n Вывод списка: \n\n";

Node\* curr = head;

while (curr) {

cout << " Номер поставки - " << curr->inf.num << '\n';

cout << " Вес руды - " << curr->inf.weight << '\n';

cout << " Выход металла - " << curr->inf.result << "\n\n";

curr = curr->next;

}

}

return;

}

// Удаление элемента из списка

void List::Delete() {

if (!head)

cout << "\n\n ОШИБКА: Список отсутствует\n";

else {

Node\* last = tail;

tail = tail->prev;

tail->next = NULL;

delete last;

cout << "\n\n Последний элемент списка удалён \n";

}

return;

}

// Поиск суммарного веса чистого металла (ожидаемого выхода)

float List::ClearWeight() {

float sumResult = 0;

Node\* curr = head;

while (curr) {

sumResult += curr->inf.result;

curr = curr->next;

}

return sumResult;

}

// Приостановка программы до нажатия Enter

void Pause() {

while (1) {

int c2 = \_getch();

if (c2 == 13) break;

}

return;

}

// Главная функция - меню программы

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

List myList, copyList, empty;

while (1) {

// Список команд меню

system("cls");

cout << "\n МЕНЮ ПРОГРАММЫ \n\n";

cout << " [1] Создать / дополнить список \n";

cout << " [2] Просмотр списка \n";

cout << " [3] Удалить элемент из списка \n";

cout << " [4] Суммарный вес чистого металла \n";

cout << " [5] Создать копию списка \n";

cout << " [6] Просмотр копии списка \n";

cout << " [0] Завершение программы, очистка памяти \n\n";

cout << " Ожидание комманды... ";

// Ввод команды и выбор соотв. действия программы

switch (\_getch()) {

case '1': myList.Add(); break; // заполнение списка

case '2': myList.Show(); Pause(); break; // просмотр списка

case '3': myList.Delete(); Pause(); break; // удаление элемента

case '4': // сумм. вес

cout << " \n\n Чистый вес металла - ";

cout << myList.ClearWeight() << " у.е.\n";

Pause(); break;

case '5': { // копирование списка

copyList.~List();

List newList = myList;

copyList = newList;

newList = empty;

Pause(); break;

}

case '6': copyList.Show(); Pause(); break; // просмотр копии списка

case '0': // выход из программы

cout << "\n\n Программа успешно завершена \n";

return 0;

}

}

}

Тестовые примеры

Написанная программа была запущена в формате .exe. В результате в консоли был выведен список команд, составляющий меню программы (Рисунок 2).

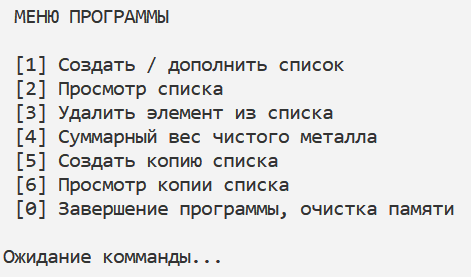


Рисунок 2 – Меню программы

Прежде чем создать новый список были запущены команды, осуществляющие операции над ним. В результате было выведено сообщение об ошибке в связи с отсутствием списка для обработки (Рисунок 3).

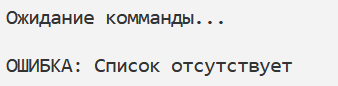


Рисунок 3

Была запущена команда заполнения списка (метод Add), в список myList были введены данные двух элементов (Рисунок 4).

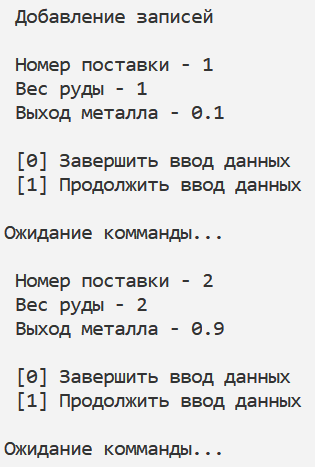


Рисунок 4 – Заполнение списка

Затем была запущена команда просмотра списка myList (метод Show). В результате были выведены данные, соответствующие введённым ранее элементам (Рисунок 5).

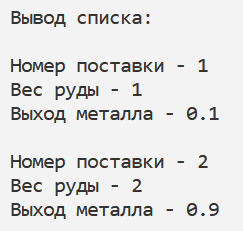


Рисунок 5 – Просмотр списка

Далее для myList был запущен метод ClearWeight, возвращающий суммарный вес чистого металла (поле result элемента списка). В результате программой была выведена верно подсчитанная сумма (Рисунок 6).

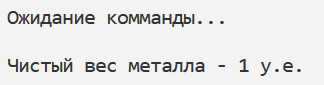


Рисунок 6 – Вывод суммы значений полей result

Была запущена команда копирования созданного списка (myList) в другой список copyList. При просмотре списка copyList были выведены те же данные, что и при выводе myList (Рисунок 5).

Затем с помощью метода Delete из myList был удалён последний элемент, что верно отобразилось при его повторном просмотре (Рисунок 7).

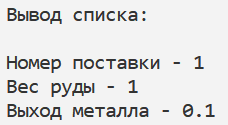


Рисунок 7 – Повторный просмотр списка myList

Был произведён повторный просмотр списка copyList. Просмотр показал, что его данные остались неизменны (Рисунок 5), что свидетельствует о верной работе конструктора копирования класса List.

Далее был осуществлён выход из программы (Рисунок 8). В процессе тестирования было подтверждено, что при окончании программы (а также при повторном копировании списка) все не используемые списки удаляются, утечки памяти не происходит. Это свидетельствует о верной работе деструктора класса List.

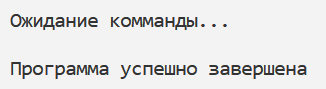


Рисунок 8 – Выход из программы

На основании проведённого тестирования было заключено, что программа верно справляется с поставленными перед ней задачами.

Вывод

В ходе работы были изучены основы классового программирования на языке C++. Исследованы особенности работы со сложными динамическими структурами данных при объектном подходе.

С помощью классов был реализован двусвязный список, структура его информационного поля, методы его создания, просмотра и обработки, а также конструктор копирования и деструктор. Результаты работы были зафиксированы в виде консольного приложения, через меню которого возможно осуществлять операции над объектами созданного для списков класса.