Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №11.1**

Дисциплина: Основы алгоритмизации и программирования

Тема: “Динамические структуры данных”

Вариант 15

Выполнил:

Студент гр. ИВТ-20-2б

Чувашев Максим Алексеевич

Проверил:

Доцент кафедры ИТАС  
Полякова О.А

Пермь, 2021

**Цель работы**

Получить практические навыки работы с однонаправленными списками;

**Постановка задачи**

1. Сформировать однонаправленный список. Тип информационного поля указан в варианте. (Тип информационного поля double. Добавить в список после каждого элемента с отрицательным информационным полем элемент с информационным полем равным 0)
2. Распечатать полученную структуру.
3. Выполнить обработку структуры в соответствии с заданием.
4. Распечатать полученный результат.

**Анализ задачи**

1. Используемые типы данных.
   1. В программе используется: структура List, которая используется для хранения информационного поля типа double и дополнительного поля, которое хранит адрес следующего элемента списка.

struct LIST

{

double inf; // информационное поле

LIST\* ptrNext = NULL; // указатель на следующий элемент

};

* 1. Тип double используется, для хранения информационных данных в структуре.
  2. Тип int используется в цинке со счетчиком.

1. Действия над используемыми данными.
   1. Данные типа int используется для создания списка в цикле со счетчиком

for (int i = 2; i <= size; i++)

* 1. Тип double используется для хранения данных информационного поля списка.

double inf; // информационное поле

* 1. С данными типа List производятся следующие действия: создание односвязного списка, добавление элементов, удаление элементов, вывод всех элементов на экран на экран. (Код представлен в пункте )

1. Вид данных.
   1. Односвязный список в программе реализован с помощью структуры, одно из полей которых хранит указатель на следующий элемент, а другое поле содержит информацию типа double. (Код в пункте)
2. Структура.
   1. В программе используется структура, реализующая односвязный список. Поле inf типа double хранит значение информационного поля списка. А поле ptrNext хранит указатель на следующий элемент списка.

struct LIST

{

double inf; // информационное поле

LIST\* ptrNext = NULL; // указатель на следующий элемент списка

};

1. Ввод и вывод.
   1. Ввод и вывод реализованы через консоль с помощью операторов ввода и вывода cin cout.

cout << "Введите количество элементов в списке: \t";

cin >> (\*size);

1. Действия для решения задачи.
   1. Функция, которая просит пользователя ввести размер будущего списка. В случае, если указанный пользователь меньше единицы, программа сообщает об этом, и просит ввести корректный размер списка.

void Fill(int\* size) // функция ввода количества элементов списка

{

cout << "Введите количество элементов в списке: \t";

cin >> (\*size);

while ((\*size) < 1)

{

cout << "Список такого размера не может быть создан!" << endl;

cout << "Введите количество элементов в списке: \t";

cin >> (\*size);

}

}

* 1. Функция, которая создает сам список, получая в качестве параметра размер, указанный пользователем. Программа дополнительно проверяет размер. Далее программа, проходится с помощью цикла и создает столько элементов структуры сколько указал пользователь. Подробное действие функции описано в коментариях к коду.

LIST\* Init(int size) // заполниение списка

{

if (size < 1) // проверка на возможность существования списка

{

cout << "Список не может быть создан!";

return 0;

}

LIST\* ptrFirst = NULL; // изначально указатель на первый элемент ни на что не указывает

LIST\* ptr = new LIST; // выделяется динамическая память для элемента структуры

cout << "Введите 1 элемент списка: \t";

cin >> ptr->inf; // пользователь заполняет информационное поле первого элемента

ptrFirst = ptr; // указатель на первый элемент теперь указывает на первый элемент

for (int i = 2; i <= size; i++) // цикл для создания остальных элементов списка

{

LIST\* NewOb = new LIST; // выделение динамической памяти для объекта структуры

ptr->ptrNext = NewOb; // поле ptrNext теперь указывает на следующий элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

cout << "Введите " << i << " элемент списка: \t";

cin >> ptr->inf; // пользователь заполняет информационное поле текущего элемента списка

}

return ptrFirst; // возвращаем указатель на первый элемент списка

}

* 1. Функция, которая выводит все элементы списка в консоль. Сначала она проверяет список на существование, если список существует, то она проходится по всем элемента списка с помощью цикла while и выводит все встретившееся ей элементы в консоль. Подробное действие данной функции описано в комментариях к коду.

void Print(LIST\* ptrFirst) // функция вывода списка в консоль

{

cout << endl;

if (ptrFirst == NULL) // проверка на существование списка

{

cout << "Список пуст!" << endl;

}

else

{

LIST\* ptr = ptrFirst; // создание элемента структуры

while (ptr != NULL) // проверка на окончание списка

{

cout << ptr->inf << "\t"; // вывод информационного поля списка в консоль

ptr = ptr->ptrNext; // переход к следующему элементу списка

}

cout << endl << endl;

}

}

* 1. Функция, которая будет изменять список. Она будет добавлять 0 после каждого встретившегося отрицательного числа. Она проходится с помощью цикла while по всех элементам списка, проверяет их на отрицательность, и в случае положительного исхода, создает новый элемент списка с информационным полем равным 0. Подробное действие функции описано в комментариях к приложенному коду.

void Edit(LIST\* ptrFirst) // функция изменения списка

{

LIST\* ptr = ptrFirst; // создание элемента списка

while (ptr != NULL) // цинкл пока элементы существуют

{

if (ptr->inf < 0 ) // проверка информационных полей на отрицательность

{

LIST\* old = ptr->ptrNext; // запоминает старый элемент

LIST\* NewOb = new LIST; // создаем новый элемент

ptr->ptrNext = NewOb; // указываем на новый элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

ptr->inf = 0; // заполняем информационное поле данного элемента 0

ptr->ptrNext = old; // указываем на запомненный нами ранее элемент

}

else

{

ptr = ptr->ptrNext; // переходим к следующему элементу

}

}

}

* 1. Функция, удаляющая все элементы списка из динамической памяти. Она проходится циклом while по всем элементам списка и удаляет их. Подробное действие функции описано в комментариях к приложенному коду.

void Del(LIST\* ptrFirst) // функция удаления списка

{

LIST\* ptr = ptrFirst; // создаем элемент структуры

while (ptr != NULL) // пока не кончатся элементы проходимся по списку

{

LIST\* old = ptr; // запоминаем текущий элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

delete old; // удаляем запомненный нами элемент

}

}

* 1. Функция, main() которая последовательно вызывает все вышеописанные функции.

int main()

{

system("chcp 1251>nul");

int size;

Fill(&size);

LIST\* ptrFirst = Init(size);

Print(ptrFirst);

Edit(ptrFirst);

Print(ptrFirst);

Del(ptrFirst);

}

**Полный код**

#include <iostream>

#include <string>

using namespace std;

struct LIST

{

double inf; // информационное поле

LIST\* ptrNext = NULL; // указатель на следующий элемент списка

};

void Fill(int\* size) // функция ввода количества элементов списка

{

cout << "Введите количество элементов в списке: \t";

cin >> (\*size);

while ((\*size) < 1)

{

cout << "Список такого размера не может быть создан!" << endl;

cout << "Введите количество элементов в списке: \t";

cin >> (\*size);

}

}

LIST\* Init(int size) // заполниение списка

{

if (size < 1) // проверка на возможность существования списка

{

cout << "Список не может быть создан!";

return 0;

}

LIST\* ptrFirst = NULL; // изначально указатель на первый элемент ни на что не указывает

LIST\* ptr = new LIST; // выделяется динамическая память для элемента структуры

cout << "Введите 1 элемент списка: \t";

cin >> ptr->inf; // пользователь заполняет информационное поле первого элемента

ptrFirst = ptr; // указатель на первый элемент теперь указывает на первый элемент

for (int i = 2; i <= size; i++) // цикл для создания остальных элементов списка

{

LIST\* NewOb = new LIST; // выделение динамической памяти для объекта структуры

ptr->ptrNext = NewOb; // поле ptrNext теперь указывает на следующий элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

cout << "Введите " << i << " элемент списка: \t";

cin >> ptr->inf; // пользователь заполняет информационное поле текущего элемента списка

}

return ptrFirst; // возвращаем указатель на первый элемент списка

}

void Print(LIST\* ptrFirst) // функция вывода списка в консоль

{

cout << endl;

if (ptrFirst == NULL) // проверка на существование списка

{

cout << "Список пуст!" << endl;

}

else

{

LIST\* ptr = ptrFirst; // создание элемента структуры

while (ptr != NULL) // проверка на окончание списка

{

cout << ptr->inf << "\t"; // вывод информационного поля списка в консоль

ptr = ptr->ptrNext; // переход к следующему элементу списка

}

cout << endl << endl;

}

}

void Edit(LIST\* ptrFirst) // функция изменения списка

{

LIST\* ptr = ptrFirst; // создание элемента списка

while (ptr != NULL) // цинкл пока элементы существуют

{

if (ptr->inf < 0 ) // проверка информационных полей на отрицательность

{

LIST\* old = ptr->ptrNext; // запоминает старый элемент

LIST\* NewOb = new LIST; // создаем новый элемент

ptr->ptrNext = NewOb; // указываем на новый элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

ptr->inf = 0; // заполняем информационное поле данного элемента 0

ptr->ptrNext = old; // указываем на запомненный нами ранее элемент

}

else

{

ptr = ptr->ptrNext; // переходим к следующему элементу

}

}

}

void Del(LIST\* ptrFirst) // функция удаления списка

{

LIST\* ptr = ptrFirst; // создаем элемент структуры

while (ptr != NULL) // пока не кончатся элементы проходимся по списку

{

LIST\* old = ptr; // запоминаем текущий элемент

ptr = ptr->ptrNext; // переходим к следующему элементу

delete old; // удаляем запомненный нами элемент

}

}

int main()

{

system("chcp 1251>nul");

int size;

Fill(&size);

LIST\* ptrFirst = Init(size);

Print(ptrFirst);

Edit(ptrFirst);

Print(ptrFirst);

Del(ptrFirst);

}

**Блок схема**







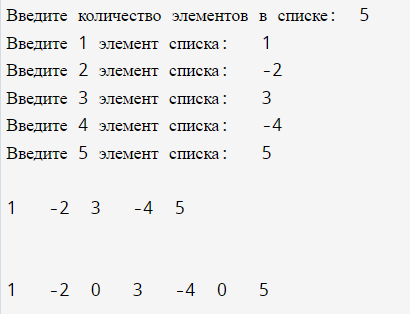
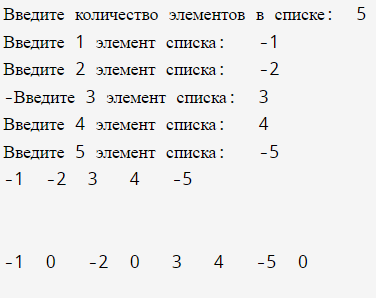
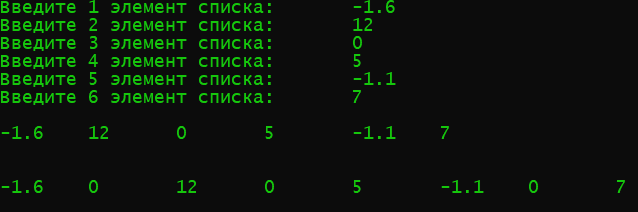








**Скриншоты результатов работы программы**

1. **Нормальный ввод:**
   1. 
   2. 
   3. 
2. **Неправильный ввод:**
   1. 