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 №11.2**

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 двунаправленными списками;

**Постановка задачи**

1. Сформировать двунаправленный список. Тип информационного поля указан в варианте. (Тип информационного поля int. Удалить из списка все элементы с четными номерами.)
2. Распечатать полученную структуру.
3. Выполнить обработку структуры в соответствии с заданием.
4. Распечатать полученный результат.

**Анализ задачи**

1. Используемые типы данных.
   1. Структура List используется для хранения одного информационного поля типа. В реализации через структуры мы так же создаем для объекта структуры еще 2 дополнительных поля, а именно указатель на предыдущий и указатель на следующий объект.

struct LIST // структура для двунаправленного списка

{

    int inf = 0; // информационное поле типа int

    LIST\* ptrPrev = NULL; // поле хранящее адрес предыдущего элемента

    LIST\* ptrNext = NULL; // поле хранящее адрес следующего элемента

};

* 1. Тип int используется для хранения одного поля типа int, а также для хранения размера size, для счетчика count и для временной переменной el.

int count = 1;

int size;

int el;

* 1. Тип list<int> используется для хранения элементов с помощью двусвязного списка при реализации через STL библиотеки.

1. Действия над используемыми данными.
   1. С данными типа int:
      1. Инкремент в счетчике.

count++;

* + 1. Инкремент в цикле.

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

* + 1. Используется для хранения данных в информационном поле.
  1. С данными List:
     1. Создание двусвязного списка.
     2. Удаление четных элементов.
     3. Вывод всех элементов на экран.
  2. Данные типа list<int> :
     1. Создание двусвязного списка.
     2. Удаление элементов списка.
     3. Копирование данных из одного списка в другой.
     4. Вывод всех элементов списка на экран.

1. Вид данных.
   1. Двусвязный список реализован с помощью структуры LIST и библиотеки list в STL.
2. Структура.
   1. В программе создана структура LIST, которая реализует двусвязный список. Поле inf – информационное поле списка. Поле ptrPrev и ptrNext хранят адреса предыдущего и следующего элемента соответсвенно.

struct LIST // структура для двунаправленного списка

{

    int inf = 0; // информационное поле типа int

    LIST\* ptrPrev = NULL; // поле хранящее адрес предыдущего элемента

    LIST\* ptrNext = NULL; // поле хранящее адрес следующего элемента

};

1. Ввод и вывод.
   1. Ввод и вывод всех данных осуществляются с помощью операторов ввода и вывода cin и cout.

cout << "Введите количество элементов списка: \t";

cin >> (\*size);

1. Действия для решения задачи.
   1. Функция, которая просит пользователя ввести размер двусвязного списка, и в случае некорректного ввода сообщает ему об этом. Эта функция одинакова для обеих реализаций.

void FillS(int\* size) // функция для ввода размера двунаправленного списка пользователем

{

    cout << "Введите количество элементов списка: \t";

    cin >> (\*size);

    while (\*size < 1)

    {

        cout << "Введен некорректный размер списка!" << endl;

        cout << "Введите количество элементов списка: \t";

        cin >> (\*size);

    }

}

* + 1. Функция создания двунаправленного списка через структуры. Данная функция сначала повторно проверяет размер списка, если размер некорректен, функция сообщим ему об этом. Далее мы проходимся с помощью цикла for по всем элементам списка, выстраивая связи между ними и задавая значение информационного поля. Подробнее в комментариях к коду.

LIST\* Init(int size) // функция создания двунаправленного списка

{

    if (size < 1) // проверка на то, чтобы размер списка не бы меньше единицы

    {

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

        return 0;

    }

    LIST\* ptrFirst = NULL; //  создается указатель на первый элемент, пока он никуда не указывает

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

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

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

    ptrFirst = ptr; // указателю на первый элемент присваивается созданный элемент

    for (int i = 2; i <= size; i++) //  повторяем процедуру до тех пор, пока количество элементов не достигнет необходимого количества

    {

        LIST\* NewOb = new LIST; // выделяется динамическая память для нового элемента списка

        LIST\* ptr2 = ptr; //  создается временная переменная которая будет хранить текущее положение

        ptr->ptrNext = NewOb; // полю со следующим элементом присваивается указатель на новый объект

        ptr = ptr->ptrNext; //  переходим на следующий элелмент

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

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

        ptr->ptrPrev = ptr2; // полю с предыдущим элементом текущего элемента присваивается адрес, который мы ранее запомнили

        ptr->ptrNext = NULL; //  полю со следующим элементом присваиваем NULL

    }

    return ptrFirst; // возвращаем указатель на первый элемент списка

}

* + 1. Заполнение же списка, реализованного с помощью библиотеки STL, происходит сильно проще, нам лишь необходимо пройтись с помощью цикла for от 0 до размера списка, и класть, с помощью уже заранее заготовленного метода элемент в конец списка. Подробнее в комментариях к коду.

void fillLIST(list<int> &LIST, int size) // заполнение списка

{

int el; // переменная которая будет временно хранить данные которые мы будем заность в список

for (int i = 0; i < size; i++) // проходимся от 0 до размера списка и заполняем список

{

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

cin >> el; // ввод эмемента списка

LIST.push\_back(el); // введенный элемент кладется в конец списка

}

}

* + 1. Функция, которая печатает содержимое массива в консоль. Через структуры, для начала мы создаем еще один указатель на первый элемент массива. Далее с помощью цикла while пока наш указатель не начнет указывать на NULL проходимся по всему списку и выводим элемент, на который указывает указатель, далее сдвигаем указатель.

void Print(LIST\* ptrFirst) // Функция которая выводит все элементы списка на экран

{

    cout << endl << endl;

    LIST\* ptr = ptrFirst; // выделение динамической памяти под переменную

    while (ptr != NULL) // проходимся по всем элементам пока указатель ну станет равен NULL

    {

        cout << ptr->inf << "\t"; // выводим информационное поле в консоль

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

    }

}

* + 1. Функция, которая выводит на экран все элементы списка, реализованного через библиотеки STL, реализована по-другому. Мы просто проходимся по всему списку с помощью цикла for и оператора, который является началом списка, передвигая оператор, пока он не станет равен последнему элементу.

void PrintLIST(list<int> &LIST) // вывод всех элементов списка

{

for (list<int>::const\_iterator it = LIST.cbegin(); it != LIST.cend(); it++) // проходимся с помощью итератора списка по всему списку

{

cout << \*it << "\t"; // выводим элементы списка в консоль

}

cout << endl;

}

* 1. Функция удаления элементов из списка.
     1. В реализации через структуры, проходимся по всем элементам списка, и если count является четным, то мы запоминаем старый элемент, обрываем с ним связи и ставим указатели не включая этот элемент, а в конце мы удаляем данный элемент. Подробнее в комментариях к коду.

void Edit(LIST\* ptrFirst) // функция которая будет удалять каждый четный элемент списка

{

    int count = 1; //  добавляем счетчик

    LIST\* ptr = ptrFirst; //  запоминаем первый элемент списка

    while (ptr != NULL) // пока элементы существуют то мы проходимся по списку

    {

        if (count % 2 == 0) // если элемент четный заходим в цикл

        {

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

            LIST\* prev = ptr->ptrPrev; //  запоминаем адрес предыдущего элемента

            LIST\* next = ptr->ptrNext; //  запоминаем адрес следующего элемента

            prev->ptrNext = next; // у предыдущего элемента в поле которое отвечает за следующий элемент, указываем запомненный ранее элемент

            next->ptrPrev = prev; // у следующего элемента в поле, которое отвечает за предыдущий элемент, указываем запомненный ранее элемент

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

            delete old; //  удаляем элемент который мы избалили от связей

        }

        else {

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

        }

        count++; // инкрементируем счетчик

    }

}

* + 1. В реализации через STL библиотеки, мы создаем итератор который является первым элементом списка, так же создаем вспомогательный список, в который мы будем записывать только нечетные объекты. Проходимся с помощью цикла for и итератора по всем элементам списка, и если элемент не четный мы копируем его во второй список. Далее с помощью заранее определенного метода assign мы копируем из новый список в старый. И потом стираем новый массив, с помощью заранее определенного метода clear.

void EditLIST(list<int> &LIST) // удаление четных элементов

{

list<int>::iterator it = LIST.begin(); // создание итератора, которому присваивается начальный элемент списка

list<int>LIST2; // создается вспомогательный список

int count = 1; // создается счетчик

for (list<int>::const\_iterator it = LIST.cbegin(); it != LIST.cend(); it++) // с помощью итератора проходимся по всем элементам списка

{

if (count % 2 != 0) // если элемент нечетный по индексу

{

LIST2.push\_back((\*it)); // кладем в конец вспомогательного списка данный элемент

}

count++; // инкрементируем счетчик

}

LIST.assign(LIST2.begin(), LIST2.end()); // копирование всех элементов из нового списка в старый

LIST2.clear(); // очищаем новый список

}

* 1. Функция main()
     1. В реализации через структуры.

int main()

{

    setlocale(LC\_ALL, "Ru");

    int size;

    FillS(&size);

    LIST\* ptrFirst = Init(size);

    Print(ptrFirst);

    Edit(ptrFirst);

    Print(ptrFirst);

    LIST\* ptr = ptrFirst;

    while (ptr != NULL) //  пока указатель не равен NULL, проходимся циклом и удаляем весь список

    {

        ptrFirst = ptr;

        ptr = ptr->ptrNext;

        delete ptrFirst;

    }

}

* + 1. В реализации через библиотеки STL.

int main()

{

setlocale(LC\_ALL, "Ru");

int size;

EnterSize(&size);

list<int> LIST;

fillLIST(LIST, size);

PrintLIST(LIST);

EditLIST(LIST);

PrintLIST(LIST);

LIST.clear();

}

**Полный код реализация через структуры**

#include <iostream>

using namespace std;

struct LIST // структура для двунаправленного списка

{

int inf = 0; // информационное поле типа int

LIST\* ptrPrev = NULL; // поле хранящее адрес предыдущего элемента

LIST\* ptrNext = NULL; // поле хранящее адрес следующего элемента

};

void FillS(int\* size) // функция для ввода размера двунаправленного списка пользователем

{

cout << "Введите количество элементов списка: \t";

cin >> (\*size);

while (\*size < 1)

{

cout << "Введен некорректный размер списка!" << endl;

cout << "Введите количество элементов списка: \t";

cin >> (\*size);

}

}

LIST\* Init(int size) // функция создания двунаправленного списка

{

if (size < 1) // проверка на то, чтобы размер списка не бы меньше единицы

{

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

return 0;

}

LIST\* ptrFirst = NULL; // создается указатель на первый элемент, пока он никуда не указывает

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

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

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

ptrFirst = ptr; // указателю на первый элемент присваивается созданный элемент

for (int i = 2; i <= size; i++) // повторяем процедуру до тех пор, пока количество элементов не достигнет необходимого количества

{

LIST\* NewOb = new LIST; // выделяется динамическая память для нового элемента списка

LIST\* ptr2 = ptr; // создается временная переменная которая будет хранить текущее положение

ptr->ptrNext = NewOb; // полю со следующим элементом присваивается указатель на новый объект

ptr = ptr->ptrNext; // переходим на следующий элелмент

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

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

ptr->ptrPrev = ptr2; // полю с предыдущим элементом текущего элемента присваивается адрес, который мы ранее запомнили

ptr->ptrNext = NULL; // полю со следующим элементом присваиваем NULL

}

return ptrFirst; // возвращаем указатель на первый элемент списка

}

void Print(LIST\* ptrFirst) // Функция которая выводит все элементы списка на экран

{

cout << endl << endl;

LIST\* ptr = ptrFirst; // выделение динамической памяти под переменную

while (ptr != NULL) // проходимся по всем элементам пока указатель ну станет равен NULL

{

cout << ptr->inf << "\t"; // выводим информационное поле в консоль

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

}

}

void Edit(LIST\* ptrFirst) // функция которая будет удалять каждый четный элемент списка

{

int count = 1; // добавляем счетчик

LIST\* ptr = ptrFirst; // запоминаем первый элемент списка

while (ptr != NULL) // пока элементы существуют то мы проходимся по списку

{

if (count % 2 == 0) // если элемент четный заходим в цикл

{

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

LIST\* prev = ptr->ptrPrev; // запоминаем адрес предыдущего элемента

LIST\* next = ptr->ptrNext; // запоминаем адрес следующего элемента

prev->ptrNext = next; // у предыдущего элемента в поле которое отвечает за следующий элемент, указываем запомненный ранее элемент

if (next != NULL) // Проверка на обращение к NULL

{

next->ptrPrev = prev; // у следующего элемента в поле, которое отвечает за предыдущий элемент, указываем запомненный ранее элемент

}

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

delete old; // удаляем элемент который мы избалили от связей

}

else {

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

}

count++; // инкрементируем счетчик

}

}

int main()

{

setlocale(LC\_ALL, "Ru");

int size;

FillS(&size);

LIST\* ptrFirst = Init(size);

Print(ptrFirst);

Edit(ptrFirst);

Print(ptrFirst);

LIST\* ptr = ptrFirst;

while (ptr != NULL) // пока указатель не равен NULL, проходимся циклом и удаляем весь список

{

ptrFirst = ptr;

ptr = ptr->ptrNext;

delete ptrFirst;

}

}

**Полный код реализация через библиотеки STL**

#include <iostream>

#include <iterator>

#include <list>

using namespace std;

void EnterSize(int \*size) // вводим размер списка

{

cout << "Введите размер списка: \t";

cin >> (\*size);

while ((\*size) < 1) // проверяем правильнось введенных пользователем данных

{

cout << "Введен неправильный размер списка!\n";

cout << "Введите размер списка: \t";

cin >> (\*size);

}

}

void fillLIST(list<int> &LIST, int size) // заполнение списка

{

int el; // переменная которая будет временно хранить данные которые мы будем заность в список

for (int i = 0; i < size; i++) // проходимся от 0 до размера списка и заполняем список

{

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

cin >> el; // ввод эмемента списка

LIST.push\_back(el); // введенный элемент кладется в конец списка

}

}

void PrintLIST(list<int> &LIST) // вывод всех элементов списка

{

for (list<int>::const\_iterator it = LIST.cbegin(); it != LIST.cend(); it++) // проходимся с помощью итератора списка по всему списку

{

cout << \*it << "\t"; // выводим элементы списка в консоль

}

cout << endl;

}

void EditLIST(list<int> &LIST) // удаление четных элементов

{

list<int>::iterator it = LIST.begin(); // создание итератора, которому присваивается начальный элемент списка

list<int>LIST2; // создается вспомогательный список

int count = 1; // создается счетчик

for (list<int>::const\_iterator it = LIST.cbegin(); it != LIST.cend(); it++) // с помощью итератора проходимся по всем элементам списка

{

if (count % 2 != 0) // если элемент нечетный по индексу

{

LIST2.push\_back((\*it)); // кладем в конец вспомогательного списка данный элемент

}

count++; // инкрементируем счетчик

}

LIST.assign(LIST2.begin(), LIST2.end()); // копирование всех элементов из нового списка в старый

LIST2.clear(); // очищаем новый список

}

int main()

{

setlocale(LC\_ALL, "Ru");

int size;

EnterSize(&size);

list<int> LIST;

fillLIST(LIST, size);

PrintLIST(LIST);

EditLIST(LIST);

PrintLIST(LIST);

LIST.clear();

}

**Блок схема реализация через структуры**













**Блок схема реализация через STL**



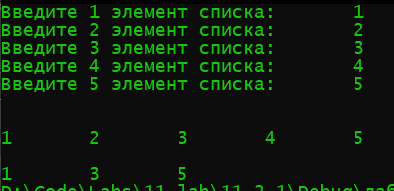
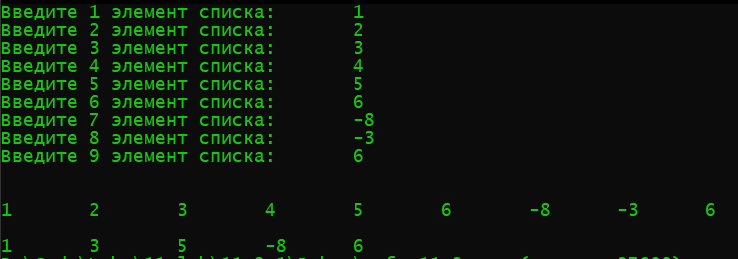
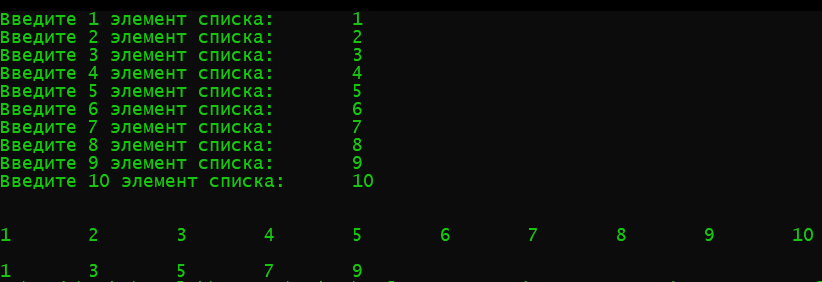
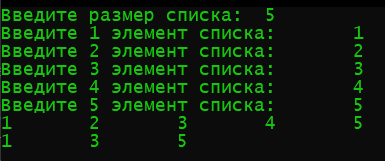
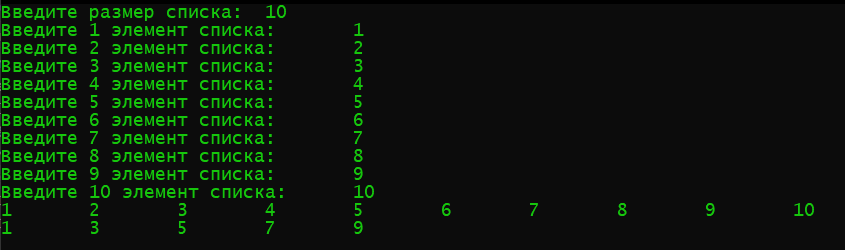








**Скриншоты результатов работы программы**

1. Нормальный ввод:
   1. Структуры
      1. 
      2. 
      3. 
   2. STL
      1. 
      2. 
      3. 