**Лабораторная работа №7**

**Большое Домашнее Задание (БДЗ)**

**Динамические структуры данных**

**Цель работы:** получить практические навыки программирования динамических структур данных в виде однонаправленного списка.

**Теоретические сведения.**

Для того, чтобы в процессе выполнения программы произвольно добавлять и удалять данные, необходимо организовать данные не в виде массива, а как-то иначе. Если к элементу данных добавить указатель, в котором будет храниться адрес следующего элемента, то это и будет кардинальным решением проблемы. Такая организация представления и хранения данных называется **динамической структурой данных**.

Каждый элемент динамических структур данных состоит из собственно данных и одного или нескольких указателей, ссылающихся на аналогичные элементы. Это позволяет добавлять в динамическую структуру новые данные или удалять какие-то из имеющихся, не затрагивая при этом другие элементы структуры.

Динамические структуры данных бывают линейные и нелинейные.

**В линейной динамической структуре** данные связываются в цепочку. К линейным структурам относятся : списки (односвязные, двухсвязные, кольцевые), стеки, очереди (односторонние, двухсторонние, очереди с приоритетами). Организация нелинейных структур более сложная. **Нелинейные структуры** представляются, как правило, в виде дерева (каждый элемент имеет некоторое количество связей, например, в бинарном дереве каждый элемент (узел) имеет ссылку на левый и правый элемент).

Во многих задачах требуется использовать данные, у которых конфигурация, размеры и состав могут меняться в процессе выполнения программы. Для их представления используют динамические информационные структуры. К таким структурам относят:

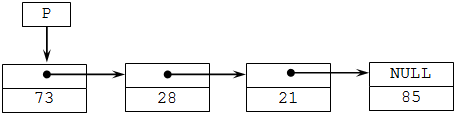
* однонаправленные (односвязные) списки;
* двунаправленные (двусвязные) списки;
* циклические списки;
* стек;
* дек;
* очередь;
* бинарные деревья.

Они отличаются способом связи отдельных элементов и/или допустимыми операциями. В данной лабораторной работе рассматриваются наиболее простые Динамические структуры данных -однонаправленные (односвязные) списки.

**Однонаправленные списки**

В однонаправленном списке данные линейно упорядочены, но порядок определяется не номерами, как в массиве, а указателями, входящими в состав элементов списка, будем называть его узлом списка. Каждый узел списка состоит из элемента-данных и элемента-указателя на следующий узел.

Каждый список имеет особый элемент, называемый указателем начала списка или головой списка, который обычно по содержанию отличен от остальных элементов. В поле указателя последнего узла находится NULL - признак конца списка.



**Рис. 1**

На Рис. 1 показан однонаправленный список из 4-х элементов.

Голова списка (P) - это указатель на первый узел.

Каждый узел списка имеет указатель на следующий.

Последний элемент имеет нулевой указатель – это хвост списка.

**Основные операции со списками:**

* создание списка;
* печать (просмотр) списка;
* вставка элемента в список;
* удаление элемента из списка;
* поиск элемента в списке
* проверка пустоты списка;
* удаление списка.

Рассмотрим некоторые приемы работы с узлами списка, фактически – это элементарные действия из которых будет состоять любой алгоритм действия. Все примеры приведены для следующего узла

// узел списка

typedefstruct **List**

{ List \*Next; // указатель на следующий узел

int **Data;** // информационное поле

}List;

Во-первых отметим, что также как в динамическом массиве, в списке действия делятся на две группы:

1. Работа с данными

* **в массиве**

int\*PMas; // указатель на данные

cout<<\*PMas<<endl; // обращение к данным

* **в списке**

List\* W; // указатель на узел

cout<<W->Data; // обращение к данным

1. Работа с адресом (переход с следующему)

* **в массиве**

PMas= PMas +1; // к следующему элементу массива

* **в списке**

W = W->Next; // к следующему узлу списка

Рассмотрим **особенности однонаправленного списка**. Указатель, связанный с текущим узлом не может адресовать ни один предыдущий, но может адресовать любой последующий узел.

Пусть **List\* Work** связан с некоторым узлом (Текущий узел), тогда:

cout<<"Текущий узел (данные) :"<<Work->Data<<endl;

cout<<"Первый следующий узел:"<<Work->Next->Data<<endl;

cout<<"Второй следующий узел :" <<Work->Next->Next->Data<<endl;

cout<<"Третий следующий узел:" <<Work->Next->Next->Next->Data<<endl;

**Задача №1**

**Преобразовать массив данных с однонаправленный список**

Решая конкретную задачу нужно стремиться реализовывать универсальные функции, которые могут впоследствии использоваться в других задачах.

#include "stdafx.h"

#include <iostream>

#include <clocale>

#define Msize 15 // размер исходного массива

using namespace std;

//структура для узла списка

typedef struct List

{ List\* Next; // указатель на следующий узел

int Data; // информационное поле

}List;

//============== прототипы функций =============

void init\_Mas (int\* B, int n); // инициализация массива

void show\_Mas (int\* B, int n,int col); // печать массива

List\* make\_List (int data); // создать список (из одного элемента)

void add\_last (int data, List\* H); //добавить узел в конец списка

void show\_List (List\* H); // печать списка

//=====================================

int \_tmain(int argc, \_TCHAR\* argv[])

{// область определений переменных

int Mas[Msize]; // исходный массив

//голова списка - указатель на начало

List\* Head=NULL; // списка нет (пустой список)

List\* Work; // рабочий указатель

setlocale (LC\_ALL,"Russian");

cout<<"---------- исходный массив -----------"<<endl;

init\_Mas (Mas, Msize);

show\_Mas (Mas, Msize,10);

cout<<"---------- создание списка -----------"<<endl;

// первый элемент массива (Mas[0]) стал первым узлом (головой) списка

Head = make\_List (Mas[0]);

show\_List (Head);

// преобразовать массив в список (со второго элемента Mas[1])

for (int i=1; i<Msize; i++) add\_last(Mas[i],Head);

cout<<"------------ весь список -----------"<<endl;

show\_List (H);

system ("pause");

return 0;

} //----------------------- конец main()-----------------------

//==================================

// ОБЛАСТЬ ОПРЕДЕЛЕНИЯ ФУНКЦИЙ

//----------------------------------------------------------------------

// СОЗДАТЬ СПИСОК (из одного элемента)

// Приём : данные

// Возврат : указатель на голову списка

//----------------------------------------------------------------------

List\* make\_List (int data)

{ List\* tH;

//выделить память под новый узел

tH= (List\*) malloc (sizeof (List) );

//запись данных в информационное поле

tH->Data=data;

// новый узел будет последним

tH->Next=NULL;

return tH;

}

//----------------------------------------------------------------------

// ПЕЧАТЬ СПИСКА

// Приём : указатель на начало печати

//----------------------------------------------------------------------

void show\_List (List\* H)

{ List\* Cur; // указатель на текущий узел

for (Cur=H; Cur->Next != NULL; Cur= Cur->Next)

cout << Cur->Data << "\t";

cout <<Cur->Data << "\n"; // печать данных последнего узла

}

//----------------------------------------------------------------------

//ДОБАВИТЬ УЗЕЛ В КОНЕЦ СПИСКА

// Приём : данные нового узла

// указатель на голову списка

//----------------------------------------------------------------------

void add\_last (int data, List\* H)

{ List\* New; // указатель для нового узла

List\* Cur; // рабочий указатель

// создаем новый узел не связанный со списком

New =(List\*) malloc (sizeof(List));// память для нового узла

New->Data = data; // запись данных

New->Next = NULL;// узел будет последним

// поиск последнего узла в списке

Cur = H; // рабочий указатель в начало

while (Cur->Next !=NULL) Cur=Cur->Next;// перемещение на следующий узел

// после выхода из цикла, Cur установлен на последний узел

Cur->Next=New; // включаем новый узел в список

}

//---------------------------------------------

void init\_Mas (int\* B, int n)

{int i;

for (i=0; i<n; i++)

B[i]=rand()%100;

}

//---------------------------------------------

void show\_Mas (int\* B, int n,int col)

{int i;

for (i=0; i<n; i++)

{cout<<B[i];

if ((i+1)%col) cout<<"\t";

else cout<<"\n";

}

cout<<"\n";

}

else cout << "\n";

}

//---------------------------------------------

**Комментарии к программе (Задача №1)**

Указатель на голову списка ( List\* Head ) расположен в функции main(), работа же с узлами списка идет в функциях. Отметим, что некоторые операции со списками будут изменять адрес головного узла, в этом случае адрес в Head функции main() также должен меняться. В Задаче №1 только одна функция меняет голову списка : List\* make\_List (int data);

Именно поэтому она возвращает указатель, а вызов функции имеет вид:

Head = make\_List (Mas[0]);

Функция make\_List () возвращает новый адрес головного узла, который запоминается в Head

**Комментарии к List\* make\_List (int data)** // создать список (из одного элемента)

Метод добавления нового узла в списка :

1. Создается независимый, не связанный со списком узел (динамическая структура данных)

List\* tH; // создать указатель для работы с динамической структурой

tH= (List\*) malloc (sizeof (List) ); //выделить память под новый узел

tH->Data=data; //запись данных в информационное поле

tH->Next=NULL; // новый узел будет последним

1. Вновь созданный независимый узел должен быть связан со списком. Каким именно образом выполняется эта связь, зависит от операции, которую реализует программист. В операции создания списка указатель на вновь созданный узел возвращается в функцию main() и становится головой списка Head = make\_List (Mas[0]);

**Комментарии к** **void show\_List (List\* H)** // печать списка

Функция используетList\* Cur; // указатель на текущий узел

в цикле распечатываются данные каждого узла :

for (Cur=H; Cur->Next != NULL; Cur= Cur->Next)

cout << Cur->Data << "\t";

Условие продолжения цикла Cur->Next != NULL – текущий узел не последний

Переход к следующему узлу Cur= Cur->Next

**Комментарии к** **void add\_last (int data, List\* H)** //добавить узел в конец списка

Потредуется два рабочих указателя – для нового узла (List\* New;) и для поиска места вставки с список

List\* New; // указатель для нового узла

List\* Cur; // рабочий указатель

1. // создаем новый узел не связанный со списком

New =(List\*) malloc (sizeof(List));// память для нового узла

New->Data = data; // запись данных

New->Next = NULL;// узел будет последним

1. Поиск места для вставки нового узла

// поиск последнего узла в списке

Cur = H; // рабочий указатель в начало

while (Cur->Next !=NULL) Cur=Cur->Next;// перемещение на следующий узел

// после выхода из цикла, Cur установлен на последний узел

1. Связь нового узла со списком.

Предыдущее значение Cur->Next=NULL, так как это последний узел, теперь он будет связан с новым узлом

Cur->Next=New; // включаем новый узел в список

Обращаю Ваше внимание, что при любых изменениях список должен иметь голову и хвост, то есть актуальное значение указателя Head и NULL вместо адреса в последнем узле (завершение списка). Функция **add\_last ()** не меняет голову списка, а о завершении списка мы позаботились заранее, на 1-ом этапе

**Задача №2**

**Удалить узел из списка после поиска.**

**Ищем указанные данные, после чего найденный узел удаляем.**

Работаем над текстом программы Задачи №1.

1. Добавим функцию поиска и удаления

//============== прототипы функций =============

bool del\_find (int etalon, List\*\* H); // удалить узел после поиска

1. Изменениявфункции main()

int \_tmain(int argc, \_TCHAR\* argv[])

{// область определений переменных

. . .

int k; // значение для поиска

. . .

k=61; // значение для поиска (конец списка)

cout<<"--------------------- удалить "<<k<<endl;

if (del\_find (k, &Head)) show\_List (Head);

else cout<<"Элемент "<<k<<" не найден!!!"<<endl;

k=41; // значение для поиска (начало списка)

cout<<"--------------------- удалить "<<k<<endl;

if (del\_find (k, &Head)) show\_List (Head);

else cout<<"Элемент "<<k<<" не найден!!!"<<endl;

k=69; // значение для поиска (середина списка)

cout<<"--------------------- удалить "<<k<<endl;

if (del\_find (k, &Head)) show\_List (Head);

else cout<<"Элемент "<<k<<" не найден!!!"<<endl;

k=999; // значение для поиска (нет в списке)

cout<<"--------------------- удалить "<<k<<endl;

if (del\_find (k, &Head)) show\_List (Head);

else cout<<"Элемент "<<k<<" не найден!!!"<<endl;

. . .

1. Опереление функции del\_find ()

//==================================

// ОБЛАСТЬ ОПРЕДЕЛЕНИЯ ФУНКЦИЙ

//-----------------------------------------------------------------

// УДАЛИТЬ УЗЕЛ ПОСЛЕ ПОИСКА

// Приём : int etalon - значение для поиска

// List\*\* Head - указатель на тип List\* (указатель на указатель)

// Возврат : (true, false), результат поиска

//-----------------------------------------------------------------

bool del\_find (int etalon, List\*\* H)

{ List\* Cur, \*Work;

bool fl = false; // флаг поиска

if ((\*H)->Data == etalon) //\*\*\*\*\*\*\*\*\*\*\* УДАЛЕНИЕ ГОЛОВНОГО УЗЛА

{ Work = (\*H)->Next; // сохранение адреса следующего за головным

free (\*H); // удалить головной узел

\*H = Work; // изменение адреса головного узла

fl=true;

}

else // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* УДАЛЕНИЕ ВСЕХ ОСТАЛЬНЫХ \*\*\*\*\*\*\*\*\*\*\*\*\*

{Cur = \*H; // Cur–узел это текущий узел (устанавливаем в начало)

// цикл поиска совпадения по данным

while (Cur->Next !=NULL) // находимся в цикле, пока Cur-узел не последний

if ((Cur->Next)->Data != etalon) // проверка следующего за Cur

Cur=Cur->Next; // смена Cur-узла (перемещение далее)

else

{fl=true; // данные совпали : искомый узел - следующий за Cur

break;

}

if (fl)

{// Work - сохранение адреса второго, следующего за Cur-узлом

Work = Cur->Next->Next;

free (Cur->Next); // удаление первого, следующего за Cur-узлом

Cur->Next= Work; // Cur-узел связывается со вторым,следующим за ним за

}

} //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

return fl;

}

//---------------------------------------------

**Комментарии к программе (Задача №2)**

В задаче №1 функцияmake\_List()изменяет голову списка, поэтому возвращает указательList\*

Но что делать если функция, которая изменяет голову списка должна возвращать другую информацию? Очевидно, что нужно «научиться» обращаться к указателю, расположенному за пределами функции из тела самой функции.

Разберем теорию этого вопроса.

Поставим абстрактную задачу, где нам нужно удалённо (из функции) обращаться к данным функции main() следующих типов: int x , int\* ptr

//------------------------------------------------------------------------------------------

// Функция принимает параметры:

// int\* p - для доступа к внешней переменной типа int (простой указатель)

// int\*\* pp - для доступа к внешней переменной типа int\* (указатель на указатель)

//------------------------------------------------------------------------------------------

void f (int\* p, int\*\* pp) // вызов f (&x, &ptr);

{// ДОСТУП К ДАННЫМ ЗА ПРЕДЕЛАМИ ФУНКЦИИ f()

// запись числа 100 в переменную x

\*p=100; // разыменование указателя p (удалённый доступ к данным типа int)

// запись p (адрес x) в указатель ptr

\*pp = p; // разыменование указателя pp (удалённый доступ к данным типа int\*)

}

int \_tmain(int argc, \_TCHAR\* argv[])

{ //данные, к которым нужно обращаться из функции f()

int x =0,\*ptr= NULL;

setlocale (LC\_ALL,"Russian");

// значения x и ptr до вызова функции

cout<<"x="<<x<<" адрес х = "<<&x<<" ptr="<<ptr<<endl;

f (&x, &ptr); // при вызове передаем адреса переменной x и указателя ptr

cout<<"Вызов функции: f (&x, &ptr);"<<endl;

cout<<"x="<<x<<" адрес х = "<<&x<<" ptr="<<ptr<<endl;

system ("pause");

return 0;

}

Пример наглядно показывает, что функцияf () удалённо изменила переменную x и указатель ptr

Вернемся к задаче №2. Планируемая функция «удаление узла после поиска» может менять адрес головного узла (если найденный узел окажется головным), в то же время она должна возвращать значение «найдено/не найдено», для диагностики результата поиска.

Поэтому прототип функции имеет вид:

bool del\_find (int etalon, List\*\* H); // удалить узел после поиска

1-ый параметр – данные для поиска

2-ой параметр – указатель на указатель для удалённого изменения Head в функции main()

Действия функции: если функция находит узел, то она удаляет его из списка и возвращает true, если же искомый узел не найден, то функция возвращает false, при этом список остается неизменным.

При вызове del\_find () проверяем резуьтат поиска: если возвращается true, то печатаем измененный массив, иначе выводим сообщение об ошибке

if (del\_find (k, &Head)) show\_List (Head);

else cout<<"Элемент "<<k<<" не найден!!!"<<endl;

Проверка правильности работы программы завершает работу над программой. Необходимо не просто проверять работоспособность, нужно убедиться, что программа корректно работает при всех вариантах использования.

В нашем случае список состоит из узлов 3-х типов:

* головной узел
* хвостовой узел
* прочие узлы списка

Необходимо проверить все возможные варианты:

k=61; // значение для поиска (конец списка)

k=41; // значение для поиска (начало списка)

k=69; // значение для поиска (середина списка)

k=999; // значение для поиска (нет в списке)

**Комментарии к bool del\_find (int etalon, List\*\* H);** // удалить узел после поиска

Схема удаления узла списка

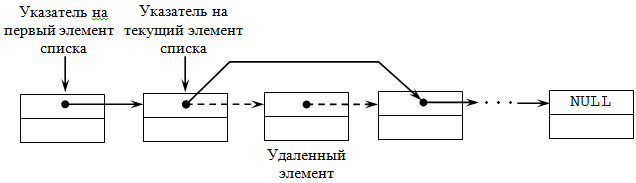


Рис 2

1. Переменные и указатели функции **del\_find()**

List\* Cur, \*Work; // рабочие указатели

bool fl = false; // флаг поиска

1. Удаление (для головного узла)

Работа с данными головного узла идет удалённо,через указатель H, поэтому каждый раз требуется разименование. Проверка головного узла на совпадение с искомым значением.

if ((\*H)->Data == etalon) //\*\*\*\*\*\*\*\*\*\*\* УДАЛЕНИЕ ГОЛОВНОГО УЗЛА

если совпадает, то перед удалением нужно сохранить адрес узла, следующего за головным, этот узел станет головным после удаления

{ Work = (\*H)->Next;

free (\*H); // удалить головной узел

\*H = Work; // изменение адреса головного узла

fl=true;

}

1. Удаление (для всех остальных узлов)

На Рис 2 видно, что в операции участвуют 3 узла: удаляемый и два его окружающих, Очевидно, что текущий указатель нужно устанавливать перед удалямым узлом (так как в однонаправленном списке нет доступа к предыдущим узлам – только к последующим).

Поиск ведем через указатель Cur функции **del\_find()**

Первоначально устанавливаем Cur на начало списка

else // \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* УДАЛЕНИЕ ВСЕХ ОСТАЛЬНЫХ \*\*\*\*\*\*\*\*\*\*\*\*\*

{Cur = \*H; // Cur–узел это текущий узел (устанавливаем в начало)

// цикл поиска совпадения по данным

while (Cur->Next !=NULL) // находимся в цикле, пока Cur-узел не последний

Проверяем узел следующий за Cur, так как если совпадет, нам нужно, чтобы указатель остался перед удаляемым узлом

if (Cur->Next->Data != etalon) // проверка следующего за Cur

Cur=Cur->Next; // смена Cur-узла (перемещаемся дальше)

else

{fl=true; // данные совпали : искомый узел - следующий за Cur

break;

}

if (fl)

Узел найден, действия по удалению:

1. сохранение адреса узла (в Work) с которым связан удаляемый узел. Если удаляемый узел – это первый следующий за Cur-узлом, то адрес в Work – это второй, следующий за Cur-узлом

{// Work - сохранение адреса второго, следующего за Cur-узлом

Work = Cur->Next->Next;

Удаление найденного узла - первого следующего за Cur-узлом

free (Cur->Next); // удаление первого, следующего за Cur-узлом

На данном этапе Cur-узел связан с удаленным узлом, нужно восстановить связность списка

Cur->Next= Work; // Cur-узел связывается со вторым,следующим за ним за

}

} //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

return fl;

}

**Общие требования к проекту**

1. Продолжаем работу над проектом.
2. Необходимо выполнять все требования, сформулированные в лабораторных работах №5, №6, по оформлению информации на экране, диалоговому интерфейсу.
3. Информацию в памяти организовать в виде однонаправленного списка.
4. Все разработанные функции приспособить для работы со списком, включая функции взаимодействия с жестким диском.
5. Реализовать новые функции, указанные в задании
6. **Обратить особое внимание на тестирование программы. Ваш проект должен надежно работать при любых входных данных и действиях оператора.**

|  |  |
| --- | --- |
| **Номер компьютера** | **Задание для групп** |
| 1,13,25 | **Проект: ВУЗ (Студент), продолжение**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * определить всех студентов по заданному направлению подготовки; * определить всех студентов заданной группы (группа это направление подготовки+номер группы), результат запомнить в новом списке * при выходе из программы сохранить основной список на диске.   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить информацию о новом студенте (добавление в начало списка); * найти заданного студента и удалить его ; * отсортировать основной список по алфавиту; * при выходе из программы запомнить измененную базу на диске. |
| 2,14,26 | **Проект: Склад (товары)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * найти все товары заданной секции; * определить товары с количеством меньше заданного, результат вывести на экран; * при выходе из программы сохранить основной список на диске.   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить товар (товар – это название + цена). Если товар уже есть, то добавляем количество, иначе создаем новую позицию (добавлять в конец списка) ; * найти товар по названию и удалить его; * отсортировать список по возрастанию количества; * при выходе из программы запомнить измененную базу на диске. |
| 3,15,27 | **Проект: Транспорт (пассажир самолета)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде * найти все рейсы заданной компании, результат вывести на экран; * найти всех пассажиров заданного рейса; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить нового пассажира в заданную позицию (по номеру) списка; * найти и удалить всех пассажиров заданного рейса; * список сортировать по алфавиту; * при выходе из программы запомнить измененную базу на диске. |

|  |  |
| --- | --- |
| 4,16,28 | **Проект: Банк (депозит)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * определить все счета с суммой больше заданной, результат вывести на экран; * найти все счета заданного клиента (по имени и фамилии), результат; * запомнить в новом списке и вывести на экран; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить новый счет в начало списка * найти и удалить все счета с суммой меньше чем заданная * сортировать список по алфавиту * при выходе из программы запомнить измененную базу на диске |
| 5,17,29 | **Проект: Библиотека (статья в журнале)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка. Распечатать в табличном виде * найти все статьи заданного автора, результат вывести на экран; * найти всех авторов заданного журнала (журнал – это название + номер + год); * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить новую статью в заданную (по номеру) позицию; * удалить все статьи определенного года; * сортировать основной список по алфавиту (название журнала); * при выходе из программы запомнить измененную базу на диске. |
| 6,18,30 | **Проект: Почта (ценное письмо)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде * найти все письма заданного отправителя (отправитель – это фамилия + имя) результат вывести на экран; * найти все письма с оценкой большей заданного; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить новое письмо в конец списка; * удалить все письма по заданному индексу отправителя; * сортировать список по алфавиту (фамилия получателя); * при выходе из программы запомнить измененную базу на диске. |

|  |  |
| --- | --- |
| 7,19 | **Проект: Библиотека (книга)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * найти все книги заданного автора, результат вывести на экран; * найти всех авторов заданного издательства; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить нового авторав заданную (по номеру в списке) позицию; * найти и удалить книгу по заданному названию; * сортировать список по году издания; * при выходе из программы запомнить измененную базу на диске. |
| 8,20 | **Проект: Банк (кредит)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * найти все счета и общую сумму кредитов у заданного клиента (клиент – это фамилия + имя), результат вывести на экран; * определить всех клиентов с заданным процентом по кредиту; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить счет в начало списка; * удалить все счета с заданной датой; * сортировать список по увеличению суммы кредита; * при выходе из программы запомнить измененную базу на диске. |
| 9,21 | **Проект: Транспорт (машина)**  Данные хранить в памяти в виде однонаправленного списка.  Все ранее реализованные функции приспособить для работы со списком  **Тестовая программа для проверки ранее реализованных функций**   * загрузить данные с диска в виде списка, распечатать в табличном виде; * найти все машины заданной марки, результат вывести на экран; * найти всех владельцев с пробегом машины больше заданного; * при выходе из программы сохранить основной список на диске;   **Интерфейс программы дополнить новыми функциями работы со списком**   * добавить новую машину в конец списка; * найти и удалить все машины мощностью двигателя меньше заданной; * сортировать основной список по увеличению пробега машины; * при выходе из программы запомнить измененную базу на диске. |