Лабораторная работа №7

Работа с файлами.

***Цель работы*:** получить практические навыки программирования задач ввода-вывода с использованием файлов.

Файл – это именованная область внешней памяти, в которой хранится логически завершенный объем данных. Файл имеет следующие характерные особенности:

- имеет имя на диске, что дает возможность программам идентифицировать и работать с несколькими файлами;
- длина файла ограничивается только емкостью диска.

Часто бывает необходимо ввести некоторые данные из файла или вывести результаты в файл. Например, бывает необходимо обрабатывать массивы, которые слишком велики, чтобы полностью разместиться в памяти.

Файлы бывают текстовые и двоичные.

Текстовый файл – это файл, в котором каждый символ из используемого набора символов хранится в виде одного байта (кода символа). Текстовые файлы разбиваются на несколько строк с помощью специального символа "конец строки". Текстовый файл заканчивается специальным символом "конец файла".

Двоичный файл – файл, данные которого представлены в бинарном виде. При записи в двоичный файл символы и числа записываются в виде последовательности байт (в двоичном представлении).

Особенностью языка С++ является отсутствие в нем структурированных файлов. Все файлы рассматриваются как неструктурированная последовательность байтов. При таком подходе понятие файла распространяется и на различные устройства.

В С++ существуют специальные средства ввода-вывода данных. Все операции ввода-вывода реализуются с помощью функций, которые находятся в библиотеке С++. Библиотека С++ поддерживает три уровня ввода-вывода:

- потоковый ввод-вывод;
- ввод-вывод нижнего уровня;
- ввод-вывод для консоли и портов (зависит от ОС).

Поток – это абстрактное понятие, относящееся к любому переносу данных от источника к приемнику.

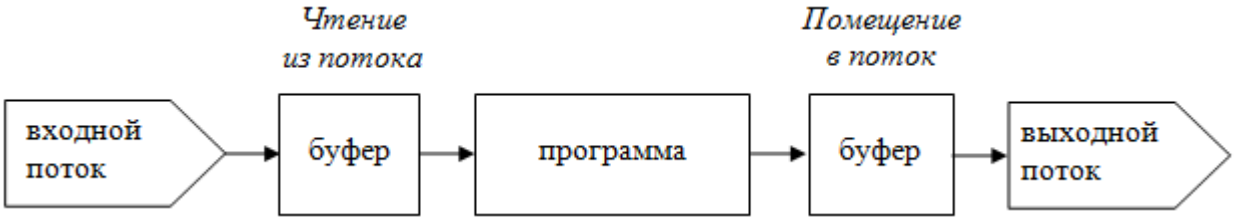
Функции библиотеки ввода-вывода языка С++, поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод. Таким образом, поток представляет собой файл вместе с предоставленными средствами буферизации.

Чтение данных из потока называется извлечением, вывод в поток – помещением (включением).

Поток определяется как последовательность байтов и не зависит от конкретного устройства, с которым производится обмен (оперативная память, файл на диске, клавиатура или принтер). Обмен с потоком для увеличения скорости передачи данных производится, как правило, через специальную область оперативной памяти – буфер. Буфер накапливает байты, и

фактическая передача данных выполняется после заполнения буфера. При вводе это дает возможность исправить ошибки, если данные из буфера еще не отправлены в программу.

Буферизация данных при работе с потоками



При работе с потоком можно:

- открывать и закрывать потоки (связывать указатели на поток с конкретными файлами);
- вводить и выводить строку, символ, форматированные данные, порцию данных произвольной длины;
- анализировать ошибки ввода-вывода и достижения конца файла;
- управлять буферизацией потока и размером буфера;
- получать и устанавливать указатель текущей позиции в файле.

Когда программа начинает выполняться, автоматически открываются пять потоков, из которых основными являются:

- стандартный поток ввода (stdin);
- стандартный поток вывода (stdout);
- стандартный поток вывода сообщений об ошибках (stderr).

По умолчанию стандартному потоку ввода stdin ставится в соответствие клавиатура, а потокам stdout и stderr соответствует экран монитора.
В C++ операции с файлами можно осуществлять в двух режимах: форматированном и потоковом. Рассмотрим основные функции для работы с файлами в форматированном режиме.

Функция открытия файла

Для работы с файлом в языке C++ необходима ссылка на файл. Для определения такой ссылки существует структура FILE, описанная в файле stdio.h. Данная структура содержит все необходимые поля для управления файлами, например: текущий указатель буфера, текущий счетчик байтов, базовый адрес буфера ввода-вывода, номер файла.

При открытии файла (потока) в программу
возвращается указатель на поток (файловый указатель), являющийся указателем на объект структурного типа FILE. Этот указатель идентифицирует поток во всех последующих операциях с файлом.

Например:

```
#include<stdio.h>
.............

FILE *fp;
```

Для открытия файла существует функция fopen, которая инициализирует файл.

Например:

p=fopen("t.txt","r");
Устанавливается связь между именем файла и указателем на файл, также определяется режим доступа к файлу.

Режимы доступа к файлу

режим	**описание**	**Начинается с …**
r	**открыть** файл для чтения	**начала**
w	**переписать или создать** новый файл для записи;	**начала**
a	**дополнить или создать** файл для записи;	**конца**
r+	**открыть** файл для чтения и записи	**начала**
w+	**создать** новый файл **для чтения и записи**	**начала**
a+	**дополнить или создать** файл для чтения и записи	**конца**

В файле stdio.h определена константа EOF, которая сообщает об окончании файла (отрицательное целое число).

При открытии файла могут возникать следующие ошибки:

- файл, связанный с потоком не найден (при чтении из файла);
- диск заполнен (при записи);
- диск защищен от записи (при записи) и т. п.

В этих случаях указатель на поток приобретет значение NULL (0). Указатель на поток, отличный от аварийного, не бывает равен NULL.

Для вывода сообщения об ошибке при открытии файла используется стандартная библиотечная функция из файла <stdio.h>:

void **perror (**const char*s);

Функция perror() выводит строку символов, адресуемую указателем s, за которой размещаются: двоеточие, пробел и сообщение об ошибке. Содержимое и формат сообщения определяются реализацией системы программирования.

Пример 1. Обнаружение и вывод ошибки при открытии файла.

```
#include "stdafx.h"
#include<stdlib.h>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
 FILE *fp;
 if ((fp=fopen("t.txt", "r"))==NULL)
   // выводит строку символов с сообщением об ошибке
   perror("ошибка при открытии файла");
```

```
  else
cout<<"open  OK!"<<endl;
system ("pause");
        return 0;

}
```

Перед началом выполнения операций с файлами целесообразно получить подтверждение, что функция fopen() выполнилась успешно

Функции работы с файлами

Функция закрытия файла

Открытые на диске файлы после окончания работы с ними рекомендуется закрыть явно. Это является хорошим тоном в программировании.
Синтаксис:
int **fclose(УказательНаПоток);**
Возвращает 0 при успешном закрытии файла и -1 в противном случае.
Открытый файл можно открыть повторно (например, для изменения режима работы с ним) только после того, как файл будет закрыт с помощью функции fclose().

Функция удаления файла

Синтаксис:
int **remove(**const char ***filename);**
Эта функция удаляет с диска файл, указатель на который хранится в файловой переменной filename. Функция возвращает ненулевое значение, если файл невозможно удалить.

Функция переименования файла

Синтаксис:
int **rename(**const char ***oldfilename,** const char ***newfilename);**
Функция переименовывает файл; первый параметр – старое имя файла, второй – новое.
Возвращает 0 при неудачном выполнении.

Функция контроля конца файла

Для контроля достижения конца файла есть функция feof.
int **feof(FILE * filename);**
Функция возвращает ненулевое значение, если достигнут конец файла.

Функции ввода-вывода данных файла

Библиотечные функции для работы с данными текстового файла
(все они описаны в файле **stdio.h**):

ФУНКЦИЯ	ОПЕРАЦИЯ
putc, fputc	Записывает символ в файл
getc , fgetc	Считывает символ из файла
fgets	Считывает строку из файла
fputs	Записывает строку в файл
fprintf	Форматированный вывод
fscanf	Форматированный ввод

Символьный ввод-вывод

Для символьного ввода-вывода используются функции:
int **fgetc(FILE *fp);**
где fp – указатель на поток, из которого выполняется считывание.

Функция возвращает очередной символ в формате int из потока fp. Если символ не может быть прочитан, то возвращается значение EOF.

int **fputc (**int **c, FILE*fp);**

где fp – указатель на поток, в который выполняется запись;

c – переменная типа int, в которой содержится записываемый в поток символ.

Функция возвращает записанный в поток fp символ в формате int. Если символ не может быть записан, то возвращается значение EOF.

Пример 2.

```
#include "stdafx.h"
//#include<stdio.h>
#include<stdlib.h>
#include <iostream>
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{
  FILE *f;
  int c;
  char *filename="t.txt";
  if ((f=fopen(filename,"r"))==0)
    perror(filename); // расшифровка и печать ошибки
  else
    while((c = fgetc(f)) != EOF)
      putchar(c);
      //вывод с на стандартное устройство вывода
  fclose(f);
system ("pause");
          return 0;
}
```

Строковый ввод-вывод

Для построчного ввода-вывода используются следующие функции:

char ***fgets(**char ***s,** int **n, FILE *f);**

где char *s – адрес, по которому размещаются считанные байты;

int n – количество считанных байтов;

FILE *f – указатель на файл, из которого производится считывание.

Прием байтов заканчивается после передачи n-1 байтов или при получении управляющего символа '\n'. Управляющий символ тоже передается в принимающую строку. Строка в любом случае заканчивается '\0'. При успешном завершении
считывания функция возвращает указатель на прочитанную строку, при неуспешном – 0.

int fputs(char *s, FILE *f);

где char *s – адрес, из которого берутся записываемые в файл байты;

FILE *f – указатель на файл, в который производится запись.

Символ конца строки ('\0') в файл не записывается. Функция возвращает EOF, если при записи в файл произошла ошибка, при успешной записи возвращает неотрицательное число.

Пример 3. Построчное копирование данных из файла f1.txt в файл f2.txt.

```
#include "stdafx.h"
#include<stdlib.h>
#include <iostream>
#define MAXLINE 255 //максимальная длина строки
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{//копирование файла in в файл out
  FILE *in, //исходный файл
       *out; //принимающий файл
  char buf[MAXLINE];
  //строка, с помощью которой выполняется копирование
  in=fopen("t1.txt","r");
  //открыть исходный файл для чтения
  out=fopen("t2.txt","w");
  //открыть принимающий файл для записи
  while(fgets(buf, MAXLINE, in)!=0)
  //прочитать байты из файла in в строку buf
    fputs(buf, out);
    //записать байты из строки buf в файл out
  fclose(in); //закрыть исходный файл
  fclose(out);//закрыть принимающий файл
system ("pause");
return 0;
}
```

Двоичный ввод-вывод

Для двоичного (блокового) ввода-вывода используются функции:

int fread(void *ptr, int size, int n, FILE *f);

где void *ptr – указатель на область памяти, в которой размещаются считанные из файла данные;
int size – размер одного считываемого элемента;
int n – количество считываемых элементов;
FILE *f – указатель на файл, из которого производится считывание.
В случае успешного считывания функция возвращает количество считанных элементов, иначе – EOF.

int fwrite(void *ptr ,int size, int n, FILE *f);

где void *ptr – указатель на область памяти, в которой размещаются считанные из файла данные;
int size – размер одного записываемого элемента;
int n – количество записываемых элементов;
FILE *f – указатель на файл, в который производится запись.
В случае успешной записи функция возвращает количество записанных элементов, иначе – EOF.

Пример 4.

```
#include "stdafx.h"
//#include<stdio.h>
#include<stdlib.h>
#include <iostream>
//#define MAXLINE 255 //максимальная длина строки
using namespace std;
struct Employee {char name[30];
                 char title[30];
                 float rate;
                };
```

```
int _tmain(int argc, _TCHAR* argv[]){
  Employee e;
  FILE *f;
  if((f=fopen("text.dat","w"))==NULL) {
    printf("\nФайл не открыт для записи");
  }
  int n;
  //запись в файл
  printf("\nВведите количество записей N=");
  scanf("%d",&n);
  for(int i=0;i<n;i++) {
    //формируем структуру e
    printf("имя:");scanf("%s",&e.name);
    printf("наименование:");scanf("%s",&e.title);
    printf("налог:");scanf("%f",&e.rate);
    //записываем e в файл
    fwrite(&e,sizeof(Employee),1,f);
  }
  fclose(f);
  //чтение из файла
  if((f=fopen("text.dat","rb"))==NULL)
    printf("\nФайл не открыт для чтения");
  while(fread(&e,sizeof(Employee),1,f))
    printf("%s, %s, %f;\n", e.name, e.title, e.rate);
  fclose(f);
system ("pause");
        return 0;
}
```

Форматированный ввод-вывод

В некоторых случаях информацию удобно записывать в *файл* без преобразования, т.е. в символьном виде, пригодном для непосредственного отображения на экран. Для этого можно использовать функции форматированного ввода-вывода:

```
int fprintf(FILE *f, const char *fmt, ...);
```

где FILE*f – указатель на файл, в который производится запись;
const char *fmt – форматная строка;
par1, par2, ... – список переменных, в которые заносится информация из файла.
Функция возвращает число переменных, которым присвоено значение.

Пример 5.

```
int _tmain(int argc, _TCHAR* argv[]){
  FILE *f;
  int n, nn,m;
  if((f=fopen("int.dat","w"))==0)
    perror("int.dat");
  for(n=1;n<11;n++)
    fprintf(f, "\n%d %d", n, n*n);
  fclose(f);
  if ((f=fopen("int.dat","r"))==0)
    perror("int.dat");
  m=1;
  while(fscanf(f, "%d %d",&n, &nn)&& m++<11)
    printf("\n%d %d",n,nn);
  fclose(f);
system ("pause");
        return 0;}
```

Произвольный доступ к данным файла

Выше мы рассмотрели последовательный доступ к данным. Все функции работы с файлами после выполнения действия (чтения или записи) автоматически передвигают файловый указатель на следующий элемент данных, таким образом происходит последовательный «перебор» данных. Если требуется выборочная работа, необходимо использовать функции перемещения файлового указателя.

1. **rewind()** - **перемещает указатель (курсор) файла в начало.**

```
void rewind (FILE *fp);
```

2. **fseek()** - **устанавливает курсор файла в заданную позицию.**

```
int fseek (FILE *fp, long delta, int begin)
```

fp - указатель на файл
delta - на сколько байт переместить курсор от начала отсчета.
begin – начало отсчета, задается одним из макросов

Начало отсчета	Имя макроса
Начало файла	**SEEK_SET**
Текущая позиция	**SEEK_CUR**
Конец файла	**SEEK_END**

3. **ftell() определяет текущую позицию файлового указателя относительно начала**

```
long int ftell (FILE *fp)
```

4. **int feof (FILE *fp)** возвращает истинное значение, если обнаружен «конец файла»

Пример 6

Создать текстовый файл "new1.txt" записав в него строку из 50 символов. Все символы, отличные от пробела, переписать в новый файл "new2.txt":

```
#include "stdafx.h"
#include <stdio.h>
#include <conio.h>
#include <string.h>
#include<stdlib.h>

int _tmain(int argc, _TCHAR* argv[]){
  char ch, sl[50];
  char text[]="one two three four";
  FILE *pf, *pr;                  // Указатели на файлы
  pf=fopen("new1.txt","w");  // Создание нового файла new1.txt
 // clrscr();
  fprintf(pf,"%s\n",text);        // Запись в файл строки text
  fclose(pf);                     // Закрытие файла pf
  pf=fopen("new1.txt","r");    // Открытие файла pf для чтения
  pr=fopen("new2.txt","w");   // Создание нового файла new2.txt
  while (!feof(pf))               // Пока не конец файла
  { ch=getc(pf);                  // Чтение символа ch из файла pf
    if (ch != ' ')
       putc(ch,pr);                 // Запись в файл pr символа ch
  }
  fclose(pr);                     // Закрытие файла pr
  rewind(pf);                     // Возврат указателя на начало файла pf
  fgets(sl,50,pf);                // Чтение из файла pf строки в переменную sl
```

```
  printf("%s\n",sl);                // Вывод строки sl на дисплей
  pr=fopen("new2.txt","r");   // Открытие файла pr для чтения
  while (!feof(pr))                // Пока не конец файла pr
  {  ch=getc(pr);                  // Чтение символа из файла pr
     putchar(ch);                  // Вывод символа ch на дисплей
   }
  fclose(pf);                       // Закрытие файлов
  fclose(pr);
system ("pause");
return 0;
}
```

При чтении текстовых файлов лучше использовать функции getc или fgetc, так как при использовании fscanf (pr, “%s”, sl) читается только очередное слово до пробела или символа табуляции и требуется повторение этой функции многократно для других слов.

Пример 7.

Написать программу, реализующую подсчет количества символов в заданном тексте и файловый ввод-вывод данных. Работа программы должна включать ввод пользователем с клавиатуры имен входного и выходного файлов. Входной файл создается с помощью редактора текста. Результат работы программы сохраняется в выходном файле, а также выводится на экран.

```
#include "stdafx.h"
#include <stdio.h>
#include<stdlib.h>
int _tmain(int argc, _TCHAR* argv[]){
  int sum=0;
  char c,file1[10],file2[10];
  FILE *t,*g;
  printf("Введите имя входного файла : ");
  scanf("%s",file1);
  printf("Введите имя выходного файла  ");
  scanf("%s",file2);
  t=fopen(file1,"r");
  g=fopen(file2,"w");
  printf("\nСодержимое входного файла %s:\n",file1);
  while(!feof(t)) {
     c=getc(t);
           if (c!=10&& c!=-1)
    //символы конца строки и конца файла не считаются
    sum++;
    printf("%c", c);
  }
  fprintf(g,"%d",sum);
  printf("\nКоличество символов в тексте входного файла равно %d",sum);
  fclose(t);
  fclose(g);
  system("pause");
 return 0;
}
```

Общие требования к проекту

1. Во всех вариантах создать новые типы
   FIO
2. Требования к оформлению информации на экране:
   - на экран выводить заголовки и комментарии
   - отражать на экране исходные данные, промежуточный и окончательный результаты;
   - выделять области ввода и вывода информации с помощью строк-разделителей.
3. Наличие **диалогового интерфейса и табличного вывода данных обязательно**
4. Первичная инициализация (когда на диске отсутствуют файлы с данными) проводится с клавиатуры.
5. При повторном входе данные загружаются с жесткого диска
6. При выходе из программы измененные данные сохраняются на жестком диске.

Пример диалогового интерфейса (База данных «Склад товаров»)

Добавить новый элемент …… 1
Распечатать базу товаров …… 2
Поиск товара по названию …… 3
Фильтр по цене …….. 4
Выход из программы …….. 5
……..
Введите номер функции

Пример распечатки данных в табличном виде (База данных «Склад товаров»)

Название Товара	**Цена (руб)**	**Количество (кг)**	**Общая сумма (руб)**
Сыр «Российский»	560	26.5	14840.00
Масло сливочное	380.5	100.25	38145.12
Рис длинный	68	25.0	1700.00
Рис круглый	62	56.75	3518.50
Всего товаров на сумму			58203.62
Количество записей в базе	4		

Пример реализации проекта (продолжение работы)

В данном примере показана простая реализация проекта для хранения разнотипной информации и взаимодействия (чтения и записи) с жестким диском. Также показан форматированный вывод информации на терминал.

1. Структуры, определяющие проект

```
include "stdafx.h"
#include<iostream>
#include <clocale>
#define size_work 5
#define size_max 200
using namespace std;
// Вспомогательный тип, для структуризации данных проекта
typedef struct _fio      // информация о человеке
{char* fam;              //Фамилия
char* name;              //Имя
char* father;            //отчество
} fio;

//-----------------------------------------------
Основной, определяющий тип данных проекта
typedef struct _work
{      fio worker;       // ФИО сотрудника
       int dept;                 // номер отдела
} work;
```

2. Прототипы функций проекта.

Правильно организацованный проект реализует каждое действие в виде вызова как минимум одной функции, а чаще в виде последовательности вызовов иерархически организованных функций. Такой подход структурирует программу и упрощает текст в вызывающей программе

```
// ввод с клавиатуры фамилии имени и отчества
void init_fio(work* pp);
// инициализация массива структур work
void init_work (work* b_work, int n) ;
// печать информации в табличном виде
void list_work (work* b_work, int n);
// сохранение данных на диске
void save_work (work* b_work, int n);
// загрузка данных с диска
void load_work (work* b_work);
```

3. Алгоритм действий в вызывающей программе

```
int _tmain(int argc, _TCHAR* argv[])
{// создание указателей для работы с памятью и жестким диском
work *pD;
FILE* pf;
setlocale (LC_ALL,"Russian");              // русификация вывода на экран
/* выделение памяти под массив структур, запрос делается с запасом (чтобы
хватило для данных, загружаемых с диска и на расширение массива при
текущей работе)*/
pD = new work[size_max];      // основной массив данных
/* режим rb+ позволяет читать и записывать данные, при отсутствии файла
генерируется ошибка*/
if((pf = fopen("data.bin","rb+"))==NULL)
// если файла нет, то первоначальный ввод с клавиатуры
{cout<<"Файла нет, инициализация с клавиатуры !"<<endl;
      init_work (pD,size_work);
}
else                  // если файл есть, то загрузка из файла*/
load_work (pD); // загрузка данных с диска в динамическую память
// печать информации в табличном виде
list_work (pD,size_work);
/* в этой части располагается пользовательский интерфейс (меню команд)
проекта*/
. . .
// сохранение данных на диске
save_work (pD,size_work);
delete []pD;
cout<<endl; system ("pause");
return 0;
}
```

Обратите внимания – в вызывающей программе нет никаких вспомогательных элементов (рабочих переменных и указателей), все что необходимое для работы находится в функциях. Поэтому алгоритм действий «легко читается», внимание не отвлекается на вспомогательные действия. Это хороший стиль программирования к которому нужно стремиться.

4. Определения функций

```
//------------------- Определения Функций--------------------------
void init_fio(work* pp)    // ввод с клавиатуры фамилии имени и отчества
{char buff[80]; // буферная строка для ввода с клавиатуры
int len;
cout<<"Введите\n Фамилию :";
fscanf (stdin,"%s",buff); // ввод строки в буфер
len=strlen(buff)+1; // определение длины
pp->worker.fam= new char(len) ; //выделение дин. памяти под строку
memcpy (pp->worker.fam, buff,len);// копирование из буфера в память
fflush(stdin);                          // очистка буфера входного потока
cout<<" Имя :";
fscanf(stdin,"%s",buff);
len=strlen(buff)+1;
pp->worker.name= new char(len) ;
memcpy (pp->worker.name, buff,len);
fflush(stdin);
cout<<" Отчество :";
fscanf(stdin,"%s",buff);
len=strlen(buff)+1;
pp->worker.father= new char(len) ;
memcpy (pp->worker.father, buff,len);
fflush(stdin);
};

//-----------------------------------------------
// инициализация с клавиатуры массива структур типа work
void init_work (work* b_work, int n)
{work *tw;// рабочий указатель для работы с дин. памятью
 for (tw=b_work; tw<b_work+n; tw++)
        {init_fio(tw); // ввод 3-х динамических строк
          cout<<" Номер отдела : ";
          cin>>tw->dept; // ввод числовой информации
          fflush(stdin);
        }
};
```

```
//---------------------------------------------
// печать информации в табличном виде
void list_work (work* b_work, int n)
{work *tw;
char  buff[80]; // буфер для соединения (упаковки) строк
// печать «шапки таблицы
cout<<'\t'<<"ФИО"<<'\t'<<'\t'<<"Номер отдела"<<endl;
cout<<"-----------------------------------------\n";
for (tw=b_work; tw<b_work+n; tw++)
{strcpy(buff,"");      //---------------------------- упаковка ФИО в одну строку
  strcat (buff,tw->worker.fam);strcat (buff," ");
  strcat (buff,tw->worker.name);strcat (buff," ");
  strcat (buff,tw->worker.father); //--------------------------
/*Какой бы длины не были строки ФИО, упакованный и отформатированный
буфер не будет превышать 25 знаков, что не позволит выйти за границы
столбца таблицы*/
  fprintf(stdout,"\n%-25s %5d",buff,tw->dept);
}
};
```

При взаимодействии **память->жесткий диск** и **жесткий диск->память** нужно различать данные 2-х типов :

- Данные хранятся монолитным массивом, в нашем случае динамический массив, адресуемый указателем **pD**
- Данные хаотически расположенные в куче (все динамические строки ФИО)

Кроме того отметим, что адреса динамических строк при записи на диск потеряют свою актуальность, так как при повторной загрузке данных с диска в память, из кучи будет выделена другая память с другими адресами.
Эти проблемы будем решать следующим образом:

- монолитный массив данных, адресуемый указателем **pD** храним на диске в бинарном файле
- хаотично расположенные строки ФИО храним в текстовом файле

Таким образом при хранении на диске данные распадаются на две не связанные друг с другом части, расположенные в разных файлах. Связность необходимая для работы будет восстановлена при загрузке данных в память.

```
// сохранение данных на диске
//---------------------------------------------
void save_work (work* b_work, int n)
{work *tw;
 FILE* pf;
// определение формата записи на диск
if((pf = fopen("data.bin","wb"))==NULL)
{perror("Ошибка открытия файла: режим save_bin");
}
// двоичная запись массива структур в файл data.bin
tw=b_work;
fwrite (tw, sizeof(work),size_work, pf);
fclose (pf); // закрыть бинарный файл
// запись динамических строк из Кучи в файл data.txt
if((pf = fopen("data.txt","w"))==NULL)
{perror("Ошибка открытия файла: режим save_txt");
}
// запись строк в текстовый файл
for (tw=b_work; tw<b_work+n; tw++)
{fprintf (pf, "%s\n", tw->worker.fam);
  fprintf (pf, "%s\n", tw->worker.name);
  fprintf (pf, "%s\n", tw->worker.father);
}
fclose (pf); // закрыть текстовый файл
};
```

```
// загрузка данных с диска
//---------------------------------------------
void load_work (work* b_work)
{work *tw;
 FILE* pf;
 char  buff[80]; // буфер для чтения строки с диска
// определение формата чтения с диска (бинарный файл)
if((pf = fopen("data.bin","rb"))==NULL)
{perror("Ошибка открытия файла: режим load_bin");
}
// продолжать цикл пока "не конец файла"
for (tw=b_work; !feof(pf); tw++)
fread (tw, sizeof(work),1, pf);       // чтение 1–ой структуры
fclose (pf); // закрыть бинарный файл
 // определение формата чтения с диска (текстовый файл)
if((pf = fopen("data.txt","r"))==NULL)
{perror("Ошибка открытия файла: режим load_txt");
}
for (tw=b_work; !feof(pf); tw++)
{ fscanf (pf,"%s\n", buff);                    // fam с диска в буфер
  //    выделение памяти под строку
  tw->worker.fam= new char(strlen(buff)+1) ;
  strcpy (tw->worker.fam, buff); // копирование из буфера в память
  fscanf (pf,"%s\n",buff);                     // name с диска
  tw->worker.name= new char(strlen(buff)+1) ;
  strcpy (tw->worker.name, buff);
  fscanf (pf,"%s\n",buff);                              // father с диска
  tw->worker.father= new char(strlen(buff)+1) ;
  strcpy (tw->worker.father, buff);
}
fclose (pf); // закрыть текстовый файл
};
```

Продолжаем работу над проектом. В каждом задании добавить поля с типами:
fio (вместо полей Имя,Фамилия) **и date** в виде число (int), месяц(char*), год (int)
Добавить новые функции. Реализовать взаимодействие с жестким диском.

Номер компьютера	**Задание для групп**
1,13,25	Проект: ВУЗ (Студент), продолжение Видоизменить структуру **student** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавляем поле типа **date** (дата рождения) (структура даты: число (int), месяц(char*), год (int)) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функциями поиска самого молодого и старшего студента (поиск по 3-м полям даты рождения) – При выходе из программы запомнить измененную базу на диске
2,14,26	Проект: Склад (товары) Видоизменить структуру **tovar** – добавить поле типа **date** (дата производства товара) (структура даты: число (int), месяц(char*), год (int)) – добавить поле типа int (срок годности в месяцах) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска просроченных товаров (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
3,15,27	Проект: Транспорт (пассажир самолета) Видоизменить структуру **plane** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата поездки) (структура даты: число (int), месяц(char*), год (int)) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска всех пассажиров

	между двумя заданными датами (поиск по 3-м полям даты) – При выходе из программы запомнить измененную базу на диске
4,16,28	**Проект: Банк (депозит)** Видоизменить структуру **deposid** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата открытия счета) (структура даты: число (int), месяц(char*), год (int)) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска клиентов со счетом, открытым позже заданной даты – При выходе из программы запомнить измененную базу на диске
5,17,29	Проект: Библиотека (статья в журнале) Видоизменить структуру **artical** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата выдачи журнала из библиотеки) (структура даты: число (int), месяц(char*), год (int)) – время, на которое выдается журнал (в месяцах) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска клиентов, которые вовремя не сдали журнал (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
6,18,30	Проект: Почта (ценное письмо) Видоизменить структуру **letter** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата отправки письма) (структура даты: число (int), месяц(char*), год (int)) – контрольное время вручения (в днях) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска клиентов, которые вовремя не получили письмо (используем понятие «текущая дата»)

	– При выходе из программы запомнить измененную базу на диске
7,19	Проект: Библиотека (книга) Видоизменить структуру **book** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата рождения автора) (структура даты: число (int), месяц(char*), год (int)) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска авторов пенсионного возраста (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
8,20	Проект: Банк (кредит) Видоизменить структуру **credit** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата получения кредита) (структура даты: число (int), месяц(char*), год (int)) – срок возврата кредита (в месяцах) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска клиентов с просроченными платежами (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
9,21	Проект: Транспорт (машина) Видоизменить структуру **car** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата последнего ТО) (структура даты: число (int), месяц(char*), год (int)) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска машин, у которых с даты последнего ТО прошло более 18 месяцев (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске

10,22	Проект: ВУЗ (преподаватель) Видоизменить структуру **prepod** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата инструктажа по технике безопасности, ИТБ), структура даты: число (int), месяц(char*), год (int) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска преподавателей, у которых с даты последнего ИТБ прошло более 15 месяцев (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
11,23	Проект: Склад (поставщики) Видоизменить структуру **supplier** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата поставки товара), структура даты: число (int), месяц(char*), год (int) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска товара у которого с даты поставки прошло более 12 месяцев (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске
12,24	Проект: Почта (посылка) Видоизменить структуру **parcel** – вместо полей «Имя, Фамилия» вводим поле типа **fio** – добавить поле типа **date** (дата отправки посылки), структура даты: число (int), месяц(char*), год (int) **Тестовая программа** – При запуске программы данные ввести с клавиатуры (первый запуск программы) или загрузить с диска (все последующие запуски) – Добавить несколько новых элементов массива данных(ввод с клавиатуры) – Все реализованные ранее функции нужно доработать в соответствии со сделанными изменениями, функции должны по-прежнему работать – Интерфейс программы дополнить функцией поиска посылки, отправленной более 6 месяцев назад (используем понятие «текущая дата») – При выходе из программы запомнить измененную базу на диске