Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

“Севастопольский государственный университет”

Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 09.03.02 – “Информационные системы и технологии”

Часть 3

Севастополь

2016

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 09.03.02 — “Информационные системы и технологии”, часть 3/ Сост. В.Н. Бондарев, Т.И. Сметанина. — Севастополь: Изд-во СевГУ 2016. — 60 с.

Методические указания предназначены для проведения лабораторных работ и обеспечения самостоятельной подготовки студентов по дисциплине “Алгоритмизация и программирование”. Целью методических указаний является обучение студентов основным принципам структурного программирования, навыкам разработки программ на языках Паскаль и C/C++.

Методические указания составлены в соответствии с требованиями программы дисциплины “Алгоритмизация и программирование” для студентов направления 09.03.02 — “Информационные системы и технологии” и утверждены на заседании кафедры информационных систем протокол № 1 от 12 сентября 2016 года.

Допущено научно-методической комиссией института информационных технологий и систем управления в технических системах в качестве методических указаний.

Рецензент: Кожаев Е.А., к.т.н., доцент кафедры информационных технологий и компьютерных систем

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ — получение практических навыков создания и отладки программ на языке C/C++, использующих сложные типы данных. В результате выполнения лабораторных работ студенты должны углубить знания основных теоретических положений дисциплины “Алгоритмизация и программирование”, решая практические задачи на ЭВМ.

Студенты должны получить практические навыки работы в интегрированной системе программирования Borland C/C++ (или Eclipse CDT), навыки разработки программ, использующих текстовые и типизированные файлы, структуры, динамические списковые и древовидные структуры данных, графическую библиотеку.

Выбор вариантов и график выполнения лабораторных работ

В лабораторных работах студент решает заданную индивидуальным вариантом задачу. Варианты заданий приведены к каждой лабораторной работе и уточняются преподавателем.

Лабораторная работа выполняется в два этапа. На первом этапе — этапе самостоятельной подготовки — студент должен выполнить следующее:

- изучить основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы, используя конспект лекций и рекомендованную литературу;
- разработать алгоритм решения задачи и составить его структурную схему;
- описать схему алгоритма;
- написать программу на языке C/C++ и описать ее;
- оформить результаты первого этапа в виде заготовки отчета по лабораторной работе (**студенты-заочники** первый этап оформляют в виде контрольной работы, которую сдают на кафедру до начала зачетно-экзаменационной сессии).

На втором этапе, выполняемом в лабораториях кафедры, студент должен:

- отладить программу;
- разработать и выполнить тестовый пример;
- подготовить и защитить отчёт по лабораторной работе.

Студенты дневного отделения должны выполнять и защищать работы **строго по графику**. График выполнения лабораторных работ:

- лабораторная работа 5 – 10-ая неделя весеннего семестра;
- лабораторная работа 6 – 12-ая неделя весеннего семестра;
- лабораторная работа 7 – 14-ая неделя весеннего семестра;
- лабораторная работа 8 – 16-ая неделя весеннего семестра.

График уточняется преподавателем, ведущим лабораторные занятия.

Требования к оформлению отчета

Отчёты по лабораторной работе оформляются каждым студентом индивидуально. Отчёт должен включать: название и номер лабораторной работы; цель работы; вариант задания и постановка задачи; разработку и описание алгоритма решения задачи; текст и описание программы; результаты выполнения программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

Постановка задачи представляет изложение содержания задачи, цели её решения. На этом этапе должны быть полностью определены исходные данные, перечислены задачи по преобразованию и обработке входных данных, дана характеристика результатов, описаны входные и выходные данные.

Разработка алгоритмов решения задачи предполагает математическую формулировку задачи и описание решения задачи на уровне схем алгоритмов. Схемы алгоритмов должны быть выполнены в соответствии с требованиями ГОСТ и сопровождаться описанием.

Описание программы включает описание вычислительного процесса на языке C/C++. Кроме текста программы, в отчёте приводится её пооператорное описание.

В приложении приводится текст программы, распечатки, полученные во всех отладочных прогонах программы с анализом ошибок, результаты решений тестового примера.

1 ЛАБОРАТОРНАЯ РАБОТА №5

ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРОКАМИ И ФАЙЛАМИ

1.1 Цель работы

Изучение основных операций над строками и файлами, программирование операций обработки строк текстовых файлов, исследование свойств файловых указателей.

1.2 Краткие теоретические сведения

1.2.1 Строки

В языке С *строка* представляет собой массив символов, завершающийся литерой ‘\0’. При объявлении строкового массива необходимо принимать во внимание наличие терминатора (‘\0’) в конце строки, отводя под строку на один байт больше. Удобно задавать длину строки именованной константой:

```
const int len_str = 80;
char stroka[len_str];
```

В переменной **stroka** можно хранить не 80 символов, а только 79.

Строки можно при описании *инициализировать* строковой константой. При этом нуль-символ (‘\0’) добавляется в строку автоматически:

```
char a[10] = “язык си”;
```

Если строка при определении инициализируется, её длину можно не указывать:

```
char a[] = “язык си ”;
```

В этом случае компилятор сам выделит память достаточную для размещения всех символов и нуль-литеры.

Для размещения строки в *динамической памяти* необходимо описать указатель на **char**, а затем выделить память с помощью функции **malloc()** или **new**:

```
char *s1 = (char*) malloc(m*sizeof(char));
char *s2 =  new char[m];
```

Здесь **m** – переменная, определяющая длину строки.

Для ввода-вывода строк можно использовать как *объекты* **cin** и **cout** языка С++, так и функции языка С. Рассмотрим сначала первый способ:

```
#include <iostream.h>
int main() {
  const int n = 80;
  char s[n];
  cin>>s;   cout<<s<<endl;  }
```

Таким образом, строка вводится аналогично числовым переменным. Однако, если ввести строку, состоящую из нескольких слов, разделенных пробелами, то будет введено только первое слово. Это связано с тем, что ввод выполняется до первого пробельного символа (**‘ ’, ‘\t’, ‘\n’**). Поэтому для ввода строки лучше использовать функции **getline** или **get**:

```
#include <iostream.h>
int main() {
  const int n = 80;
  char s[n];
  cin.getline(s,n);  cout<<s<<endl;
  cin.get(s,n);  cout<<s<<endl;
}
```

Функция **getline** считывает из входного потока **n**-1 символов или менее (если символ **'\n'** встретится раньше) и записывает их в строковую переменную **s**. Сам символ **'\n'** тоже считывается, но не записывается в **s**, вместо него записывается нуль-литера (**'\0'**). Если в потоке более **n**-1 символа, то следующий ввод будет выполняться, начиная с первого несчитанного символа.

Функция **get** работает аналогично, но оставляет в потоке символ '**\n**'. Поэтому перед повторным вызовом **get** необходимо удалять символ '**\n**' из входного потока. Для этого нужно вызвать **get** без параметров – **cin.get()**.

В языке С для ввода и вывода строк используются функции **scanf** и **printf** со спецификацией **%s**:

```
#include <stdio.h>
int main() {
  const int n = 40;
  char s[n];
  scanf("%s",s); printf("%s",s);
}
```

В функции **scanf** перед **s** операция взятия адреса (**&**) опущена, потому что имя массива является указателем на его начало. Функция **scanf** выполняет ввод до первого пробельного символа. Чтобы ввести строку, состоящую из нескольких слов, используйте спецификацию **%c**:

```
scanf("%20c",s);
```

В этом случае количество считываемых символов может быть только константой, и короткие строки придется при вводе дополнять пробелами до требуемой длины строки.

Библиотека языка С содержит функции, специально предназначенные для ввода (**gets**) и вывода (**puts**) строк.

Функция **gets(s)** читает символы с клавиатуры до появления символа '**\n**' и помещает их в строку **s** (сам символ '**\n**' в строку не включается, вместо него вносится '**\0**'). Функция возвращает указатель на **s**, а случае ошибки или конца файла – **NULL**. Однако функция **gets** не контролирует количество введенных символов в **s**, что может приводить к выходу за границы **s.**

Функция **puts(s)** выводит строку **s** на стандартное устройство вывода, заменяя завершающий символ '**\0**' на символ новой строки.

Большинство функций работы со строками описаны в головном файле **string.h.** Некоторые из этих функций указаны в таблице 1.1. Здесь аргументы ***s*** и ***t*** принадлежат типу **char***, ***cs*** и ***ct*** – типу **const char***, ***n*** – типу **size_t**, ***c*** – типу **int**, приведенному к **char**.

Например, чтобы строке **s** присвоить значение строки **t** можно использовать функции **strcpy** или **strncpy**:

```
char t[] = " Язык Си ";
char *s = new char[m];
strcpy(s,t);
strncpy(s,t,strlen(t)+1);
```

Функция **strcpy(s,t)** копирует все символы из строки **t**, включая и нуль-литеру, в строку, на которую указывает **s**. Функция **strncpy(s,t,n)** выполняет то же самое, но не более **n** символов. При этом если нуль-символ встретится раньше, копирование прекращается, и оставшиеся символы строки **s** заполняются нуль-символами. Если **n** меньше или равно длине строки **t**, завершающая нуль-литера не добавляется. Обе функции возвращают указатель на результирующую строку.

Функция **strlen(t)** возвращает фактическую длину строки **t**, не включая нуль-литеру.

Таблица 1.1 — Функции для обработки строк

char* strcpy(s,ct)	копирует стринг ***ct*** в стринг ***s***; возвращает ***s***
char* strcat(s,ct)	соединяет строку ***s*** и ***ct***
char* strcmp(cs,ct)	сравнивает ***cs*** и ***ct***
char* strchr(cs,c)	возвращает указатель на первое ***c*** в ***cs***
char* strrchr(cs,c)	возвращает указатель на последнее ***c*** в ***cs***
size_t strspn(cs,ct)	возвращает число символов в начале ***cs***, ни один из которых не входит в строку ***ct***
size_t strcspn(cs,ct)	возвращает число символов в начале ***cs***, которые принадлежат множеству символов из строки ***ct***
char* strpbrk(cs,ct)	возвращает указатель в ***cs*** на первую литеру, которая совпала с одной из литер, входящих в ***ct***
char* strstr(cs,ct)	возвращает указатель на первое вхождение ***ct*** в ***cs***
size_t strlen(cs)	возвращает длину ***cs***
char* strok(s,ct)	ищет в ***s*** лексему, ограниченную литерами из ***ct***

1.2.2 Функции файлового ввода-вывода

Язык C++ унаследовал от языка C библиотеку стандартных функций ввода-вывода. Функции ввода-вывода объявлены в заголовочном файле **<stdio.h>**. Операции ввода-вывода осуществляются с файлами. Файл может быть *текстовым* или *бинарным* (двоичным). Различие между ними заключается в том, что текстовый файл разбит на строки. Признаком конца строки является пара символов **'\r'**(возврат каретки) и **'\n'**(перевод строки). При вводе или выводе значений из текстового файла они *требуют преобразований* во внутреннюю форму хранения этих значе ний в соответствии с их типами, объявленными в программе. Бинарный файл – это просто последовательность символов. При вводе-выводе информации в бинарные файлы никаких преобразований не выполняется. Поэтому работа с бинарными файлами выполняется быстрее.

В языке C реализован так называемый *потоковый ввод-вывод*. Термин поток происходит из представления процесса ввода-вывода информации в файл в

виде последовательности или потока байтов. Все потоковые функции ввода-вывода обеспечивают буферизированный, *форматированный* или *неформатированный ввод-вывод*.

Перед тем, как выполнять операции ввода или вывода в файловый поток его следует открыть с помощью функции **fopen()**. Эта функция имеет следующий прототип:

FILE *fopen(const char *filename, const char *mode);

Функция **fopen()** открывает файл с именем **filename** и возвращает указатель на файловый поток, используемый в последующих операциях с файлом. Параметр **mode** является строкой, задающей режим, в котором открывается файл. Он может принимать значения, указанные в таблице 2.2

Таблица 2.2 — Режимы открытия файлов

Режим	Описание режима
R	Файл открывается только для чтения
W	Файл создается только для записи. Если файл с этим именем уже существует, он будет перезаписан. Чтение из файла не разрешено.
A	Режим добавления записей (append). Файл открывается только для записи в конец или создается только для записи, если он еще не существует. Чтение не разрешено.
r+	Существующий файл открывается для обновления (считывания и записи)
w+	Создается новый файл для обновления. Перезаписывается любой существующий файл с тем же именем.
a+	Файл открывается для добавления в конец; если файл не существует, он создается, и любой существующий файл с тем же именем перезаписывается.

Чтобы указать, что данный файл открывается или создается как текстовый, добавьте символ **t** в строку режима (например, **“rt”**,”**w+t**” и т.п.). Аналогично можно сообщить, что файл открывается или создается как бинарный. Для этого добавьте в строку режима символ **b** (например, “**wb**”, “**a+b**”).

В случае успеха функция **fopen** возвращает указатель на открытый поток, в случае ошибки – **NULL**. Например:

FILE *fptr = fopen(“mytxt.txt”,”rt”);

Здесь объявляется переменная **fptr** – указатель на файловый поток и выполняется её инициализация с помощью функции **fopen()**.

Для завершения работы с потоком он должен быть закрыт с помощью функции **fclose()**:

fclose(fptr);

Рассмотрим функции, осуществляющие ввод-вывод в файловый поток.
Функция **fgetc()** имеет следующий прототип:

int fgetc(FILE *fptr);

Она осуществляет ввод символа из файлового потока **fptr**. В случае успеха функция возвращает код символа. Если делается попытка прочесть конец файла или произошла ошибка, то возвращается **EOF.**

Функция **fputc()** имеет следующий прототип:

int fputc(int c,FILE *fptr);

Она осуществляет вывод символа в поток.

Функция **fgets()** имеет следующий прототип:

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

Она осуществляет чтение строки символов из файлового потока в строку **s.** Функция прекращает чтение, если прочитано **n**-1 символов или встретится символ перехода на новую строку **‘\n’**. Если этот символ встретился, то он сохраняется в переменной **s**. В обоих случаях в переменную **s** добавляется символ **‘\0’**. В случае успеха функция возвращает указатель на считанную строку **s**. Если произошла ошибка или считана метка **EOF**, то она возвращает **NULL.**

Для записи строки в файл можно использовать функцию **fputs**. При этом символ перехода на новую строку не добавляется, и завершающая нуль-литера в файл не копируется. Функция **fputs()** имеет следующий прототип:

int fputs(const *char, FILE *fptr);

В случае успеха функция **fputs()** возвращает неотрицательное значение. В противном случае она возвращает **EOF.**

Форматированный ввод-вывод текстовых файлов организуется в языке С с помощью функций **fscanf()** и **fprintf()**. Эти функции аналогичны функциям **scanf()** и **printf()** соответственно, с той лишь разницей, что их первым аргументом является указатель на файл, открытый в соответствующем режиме.

Функция **feof()** осуществляет проверку достижения метки конца файла. Она имеет прототип:

int feof(FILE *fptr);

Функция возвращает **0**, если конец файла не достигнут.

Рассмотрим пример файлового ввода и вывода с помощью функций **fscanf()** и **fprintf()**:

```
#include <stdio.h>
int main() {
  int i,x;
  FILE *in,*out; // описание указателей на файлы
  if ((in = fopen(“c:\\old.dat”,”rt”))== NULL)
    { fprintf(stderr,” Не могу открыть входной файл \n”);
          return 1;
    }
  if ((out = fopen(“c:\\new.dat”,”wt”))== NULL)
   { fprintf(stderr,”Не могу открыть выходной файл \n”);
          return 1;
   }
  i = 0;
```

```
    while (fscanf(in,"%d",&x)!=EOF)
          { i++;
           fprintf(out,"%d\t%d\n",i,x);
          }
    fclose(in);
    fclose(out);
    return 0;
  }
```

Программа копирует целые числа из входного файла **old.dat** в выходной файл **new.dat**. При этом в выходной файл записываются также порядковые номера чисел. Копирование выполняется до тех пор, пока не будет встречена метка конец файла **EOF**. Если входной или выходной файл нельзя открыть, то программа выводит соответствующее сообщение в стандартный поток **stderr**, который по умолчанию связан с пользовательским терминалом и предназначен для вывода сообщений об ошибках.

Для осуществления *неформатированного ввода-вывода* (без преобразований) применяются функции **fread()** и **fwrite()**. Эти функции имеют следующие прототипы:

size_t fread(void *ptr, size_t size, size_t n, FILE *fptr);
size_t fwrite(const void *ptr, size_t size, size_t n, FILE *fptr);

Функция **fread()** считывает блоки данных из файлового потока, на который указывает **fptr**, в буфер, доступ к которому выполняется через указатель **ptr**, а функция **fwrite()** выполняет обратную операцию, то есть переписывает блоки данных из буфера в файловый поток. При этом копируется **n** блоков данных, каждый из которых содержит **size** байтов. В случае успеха функции возвращают число скопированных блоков. В случае ошибки возвращается **0** или число полностью скопированных блоков. Если **n** больше значения, которое вернула функция **fread**, то это говорит о том, что встретилась метка конец файла.

Когда файл открывается для чтения или записи, с ним на самом деле связывается структура **FILE**, определенная в заголовочном файле **<stdio.h>**. Эта структура для каждого открытого файла содержит *счетчик положения текущей записи*. Сразу после открытия файла его значение равно **0**. Каждая операция ввода-вывода вызывает приращение этого счетчика на число записанных или считанных байтов из файла. Функции позиционирования – **fseek()**, **ftell()** и **rewind()** позволяют изменять или получать значение счетчика, связанного с файлом.

Функция

long int ftell(FILE *fptr);

возвращает текущее значение счетчика связанного с файлом. В случае ошибки она возвращает **-1L.**

Функция

int fseek(FILE *fptr, long offset, int from);

изменяет позиционирование файлового потока **fptr** на **offset** байтов относительно позиции, определяемой параметром **from**. Параметр **from** может принимать значения:

SEEK_SET — начало файла, 0;

SEEK_CURR — текущая позиция в файле, 1;

SEEK_END — конец файла, 2.

Функция **fseek()** возвращает **0**, если счетчик текущей записи успешно изменен.

Функция

void rewind(FILE *fptr);

устанавливает счетчик текущей позиции на начало потока.

1.2.3 Файловый ввод-вывод с использованием классов языка C++

Для реализации ввода-вывода с использованием классов языка C++ в программу следует включить заголовочный файл **<fstream.h>**. Для осуществления операций с файлами библиотека ввода-вывода C++ предусматривает три класса:

ifstream – потоковый класс для ввода из файла;

ofstream – потоковый класс для вывода в файл;

fstream – потоковый класс для ввода-вывода в файл.

Для создания потока ввода, открытия файла и связывания его с потоком можно использовать следующую форму конструктора класса **ifstream**:

ifstream fin(“text.txt”, ios::in);

Здесь определяется объект **fin** класса входных потоков **ifstream**. С этим объектом можно работать так же, как со стандартными объектами **cin** и **cout**, то есть использовать операции помещения в поток << и извлечения из потока >>, а также рассмотренные выше функции **get, getline**. Например:

```
const len = 80;
char s[len];
fin.getline(s,len);
```

Предполагается, что файл с именем **text.txt** находится в том же каталоге, что и текст программы, иначе следует указать полный путь, дублируя символ обратной косой черты, так как иначе он будет иметь специальное значение, например:

ifstream fin(“c:\\work\\text.txt”, ios::in);

Второй параметр этого конструктора означает режимы открытия файла. В таблице 1.3 приведены возможные режимы открытия и их значение.

Например, для открытия файла вывода в режиме добавления можно использовать инструкцию:

ofstream fout(“out.txt”,ios::out | ios::app);

Файлы, которые открываются для вывода, создаются, если они не существуют.

Таблица 1.3 — Режимы открытия и их значения

Режим	Назначение
ios::in	Открыть для чтения
ios::out	Открыть для записи
ios::ate	Начало вывода устанавливается в конец файла (допускает изменение позиции вывода)
ios::app	Открыть для добавления в конец (безотносительно к операциям позиционирования)
ios::trunc	Открыть, удалив содержимое файла
ios::binary	Двоичный режим операций

Если открытие файла завершилось неудачей, объект, соответствующий потоку, будет возвращать значение **false**. Например, если предыдущая инструкция завершилась неудачей, то это можно проверить так:

```
if (!fout)
{ cout<<" Файл не открыт \n";}
```

Если при открытии файла не указан режим **ios::binary**, то файл открывается как текстовый. В этом случае можно использовать при работе с файлом функции ввода-вывода, принятые в языке С, такие, как **fprintf()** и **fscanf()**.

Для проверки достигнут ли конец файла или нет, можно использовать функцию **eof()** класса **ios**.

Завершив операции ввода-вывода, необходимо закрыть файл, вызвав функцию **close()**:

```
fout.close();
```

Закрытие файла происходит автоматически при выходе объекта потока из области видимости.

Произвольный доступ к записям файлов реализуется с помощью функций (методов) **seekg()** и **seekp()**, используемых, соответственно, для позиционирования входного и выходного потоков. Например:

```
fin.seekg(0, ios::end);
```

Функция **seekg(offset, org)** перемещает текущую позицию чтения в файле на **offset** байтов относительно **org**. Параметр **org** может принимать одно из трех значений:

ios::beg — от начала файла;

ios::cur — от текущей позиции;

ios::end — от конца файла.

Получить текущее значение позиции в потоке ввода и вывода можно с помощью функций **tellg()** и **tellp()**, соответственно. Например, вызов **fin.tellg()** вернет значение счетчика текущей позиции.

Для осуществления *неформатированного ввода-вывода* при работе с потоками в языке С++ применяются функции **read()** и **write()**. Эти функции характеризуются прототипами:

istream_type& read(char *s, streamsize n);
ostream_type& write(const char *s, streamsize n);

Здесь параметр **s** указывает на буфер, в который соответственно помещаются или из которого извлекаются символы, а **n** означает число читаемых или записываемых символов. Примеры вызова этих функций приведены ниже в тексте программы.

1.2.4 Пример программы обработки текстового файла

Написать программу, которая считывает текст из файла и выводит в выходной файл только вопросительные предложения из этого текста.

Исходные данные и результаты.

Исходные данные хранятся в текстовом файле неизвестного размера. Предложение может занимать несколько строк текстового файла, поэтому ограничиться буфером на одну строку в данной задаче нельзя. Для этого выделим буфер, в который поместится весь файл.

Результаты являются частью исходных данных, поэтому дополнительно под них пространство выделять не требуется.

Для организации вывода предложений понадобятся переменные целого типа, в которых будем хранить позиции начала и конца предложения.

Алгоритм решения задачи.

1. Открыть файл.
2. Определить его длину.
3. Выделить в динамической памяти соответствующий буфер.
4. Считать файл с диска в буфер.
5. Анализируя буфер посимвольно, выделять предложения. Если предложение оканчивается вопросительным знаком выводить его в файл.

Детализируем последний пункт алгоритма. Для вывода предложения необходимо хранить позиции его начала и конца. Предложение может оканчиваться точкой, восклицательным или вопросительным знаком. В двух первых случаях предложение пропускается. Это выражается в том, что значение позиции начала предложения обновляется. Она будет соответствовать позиции символа, следующего за текущим символом, и просмотр буфера продолжается.

В программе будем использовать функции неформатированного чтения блоков данных из входного файла, так как применение функций посимвольного чтения неэффективно.

Для определения длины файла воспользуемся функциями **seekg()** и **tellg()** класса **ifstream**. Для этого переместим указатель текущей позиции в конец файла с помощью **seekg()**, а затем с помощью **tellg()** получим значение конечной позиции, запомнив его в переменной **len.**

```
#include <fstream>
#include <stdio>
int main(){
//создание  входного потока и открытие файла
ifstream fin("c:\\bc\\work\\text.txt", ios::in);
if (!fin) {
```

```
        cout<<"Can't open input file"<< endl;
         return 1;
    }

    fin.seekg(0,ios::end);              //указатель в конец файла
    long len = fin.tellg();             //запомнить длину файла
    char *buf = new char [len +1];      //выделить память под буфер
    //неформатированное чтение текстового файла
    fin.seekg(0,ios::beg);              //указатель в начало файла
    fin.read(buf,len);                  //скопировать len символов из fin  в буфер
    buf[len] = '\0';                    //поместить в буфер нуль-литеру
    //создание выходного потока и открытие файла
    ofstream fout("c:\\bc\\work\\textout.txt", ios::out);
    if (!fout) {
        cout<<"Can't open output file"<< endl;
         return 1;
    }
    long n=0,                           //индекс символа начала предложения
        i=0,                            //индекс символа  конца предложения
        j=0;                            //текущий индекс символа вопросит. предл.
    while(buf[i]) {                     //просмотр символов в буфере
       if( buf[i] == '?') {             //Если i-ый символ – вопрос,
          for(j=n;j<=i;j++)
                fout<<buf[j];           //то вывод предложения в поток,
          n=i+1;                        //обновление индекса начала предложения.
       }
       if(buf[i]=='.'||buf[i]=='!')     //Если предложение не вопросительное,
          n=i+1;                        //то только обновление индекса n .
       i++;
    }
    fin.close();
    fout.close();
    cout<<endl;
    return 0;
}
```

Вариант решения этой же задачи с использованием функций языка С.

```
#include <stdio.h>
int main(){
    //открытие  входного файла
    FILE *fin;
    fin=fopen("text.txt","r"); // файл хранить в папке проекта

    if (!fin) {
      puts("Can't open input file");
      return 1;
```

```
    }
  fseek(fin,0,SEEK_END);            //указатель в конец файла
  long len=ftell(fin);              //запомнить длину файла
  char *buf=new char[len+1];        //выделить память под буфер
  //неформатированное чтение текстового файла (поблочное)
  const int l_block=1024;                    //задать длину блока для чтения
  int num_block=len/l_block;                 //определить число блоков
  rewind(fin);                               //указатель в начало файла
  fread(buf,l_block,num_block+1,fin);        //чтение блоков из файла
  buf[len]='\0';                             //поместить в буфер нуль-литеру
  //создание выходного файла
  FILE *fout;
  fout = fopen("textout.txt","w");
  if (!fout) {
    puts("Can't open output file");
    return 1;
    }
  long  n=0,                    //индекс символа начала предложения
        i=0,                    //индекс символа  конца предложения
        j=0;                    //текущий индекс символа вопросит. предл.
  while(buf[i]) {               //просмотр символов в буфере
     if(buf[i]=='?') {          //Если i-ый символ – вопрос,
        for(j=n;j<=i;j++)
        putc(buf[j],fout);      //то вывод предложения в поток,
        n=i+1;                  //обновление индекса начала предложения.
     }
     if (buf[i]=='.'||buf[i]=='!')  //Если предложение не вопросительное,
        n=i+1;                  //то обновление только индекса n .
     i++;
  }
  fclose(fin);
  fclose(fout);
  printf("\n");
  return 0;
}
```

В заключение отметим, что файл с тестовым примером должен содержать предложения различной длины – от нескольких символов до нескольких строк.

Программа, написанная с использованием функций ввода-вывода языка С, а не классов языка С++ часто является более быстродействующей, но менее защищенной от ошибок ввода данных.

1.3 Варианты заданий

Вариант 1

Написать программу, которая считывает из текстового файла три предложения и выводит их в обратном порядке. Ввод-вывод осуществлять с помощью средств С.

Вариант 2

Написать программу, которая считывает текст из файла и выводит на экран только предложения, содержащие введенное с клавиатуры слово. Ввод-вывод осуществлять с помощью классов С++.

Вариант 3

Написать программу, которая считывает текст из файла и выводит на экран только строки, содержащие двузначные числа. Ввод-вывод осуществлять с помощью средств С.

Вариант 4

Написать программу, которая считывает английский текст из файла и выводит на экран слова, начинающиеся с гласных букв. Ввод-вывод осуществлять с помощью классов С++.

Вариант 5

Написать программу, которая считывает текст из файла и выводит его на экран, меняя местами каждые два соседних слова. Ввод-вывод осуществлять с помощью средств С.

Вариант 6

Написать программу, которая считывает текст из файла и выводит на экран только предложения, не содержащие запятых. Ввод-вывод осуществлять с помощью классов С++.

Вариант 7

Написать программу, которая считывает текст из файла и определяет, сколько в нем слов, состоящих не более чем из четырех букв. Ввод-вывод осуществлять с помощью средств С.

Вариант 8

Написать программу, которая считывает текст из файла и выводит на экран только цитаты, то есть предложения, заключенные в кавычки. Ввод-вывод осуществлять с помощью классов С++.

Вариант 9

Написать программу, которая считывает текст из файла и выводит на экран только предложения, состоящие из заданного количества слов. Ввод-вывод осуществлять с помощью средств С.

Вариант 10

Написать программу, которая считывает английский текст из файла и выводит на экран слова текста, начинающиеся и оканчивающиеся на гласные буквы. Ввод-вывод осуществлять с помощью классов С++.

Вариант 11

Написать программу, которая считывает текст из файла и выводит на экран только строки, не содержащие двузначных чисел. Ввод-вывод осуществлять с помощью средств С.

Вариант 12

Написать программу, которая считывает текст из файла и выводит на экран только предложения, начинающиеся с тире, перед которым могут находиться только пробельные символы. Ввод-вывод осуществлять с помощью классов С++.

Вариант 13

Написать программу, которая считывает английский текст из файла и выводит его на экран, заменив каждую первую букву слов, начинающихся с гласной буквы, на прописную. Ввод-вывод осуществлять с помощью средств С.

Вариант 14

Написать программу, которая считывает текст из файла и выводит его на экран, заменив цифры от 0 до 9 на слова «ноль», «один»...«девять», начиная каждое предложение с новой строки. Ввод-вывод осуществлять с помощью классов С++.

Вариант 15

Написать программу, которая считывает текст из файла, находит самое длинное слово и определяет, сколько раз оно встретилось в тексте. Ввод-вывод осуществлять с помощью средств С.

Вариант 16

Написать программу, которая считывает текст из файла и выводит на экран сначала вопросительные, а затем восклицательные предложения. Ввод-вывод осуществлять с помощью классов С++.

Вариант 17

Написать программу, которая считывает текст из файла и выводит его на экран, после каждого предложения добавляя, сколько раз встретилось в нем введенное с клавиатуры слово. Ввод-вывод осуществлять с помощью средств С.

Вариант 18

Написать программу, которая считывает текст из файла и выводит на экран все его предложения в обратном порядке. Ввод-вывод осуществлять с помощью классов С++.

Вариант 19

Написать программу, которая считывает текст из файла и выводит на экран сначала предложения, начинающиеся с однобуквенных слов, а затем все остальные. Ввод-вывод осуществлять с помощью средств С.

Вариант 20

Написать программу, которая считывает текст из файла и вычисляет количество открытых закрытых скобок. Дописать вычисленные значения в конец каждой строки. Результаты записать в новый файл. Ввод-вывод осуществлять с помощью классов С++.

Вариант 21

Написать программу, которая считывает текст из файла и выводит на экран предложения, содержащие максимальное количество знаков пунктуации. Ввод-

вывод осуществлять с помощью средств С.

Вариант 22

Даны два текстовых файла. Удалить из этих файлов строки, которые имеют одинаковые номера, но сами не являются одинаковыми. Результаты записать в новый файл. Ввод-вывод осуществлять с помощью классов С++.

Вариант 23

В каждой строке текстового файла найти наиболее длинную последовательность цифр. Значение ее длины преобразовать в строку, которую записать в начале строки файла. Результаты записать в новый файл. Ввод-вывод осуществлять с помощью средств С.

Вариант 24

Даны два текстовых файла. Создать третий файл из символов, которые записаны в позициях с одинаковыми номерами, но различаются между собой. Ввод-вывод осуществлять с помощью классов С++.

Вариант 25

Дан текстовый файл с программой. Исключить комментарии в тексте С программы. Ввод-вывод осуществлять с помощью средств С.

Вариант 26

Написать программу, которая подсчитывает количество пустых строк в текстовом файле. Ввод-вывод осуществлять с помощью классов С++.

Вариант 27

Написать программу, которая находит максимальную длину строки текстового файла и печатает эту строку. Ввод-вывод осуществлять с помощью средств С.

Вариант 28

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые начинаются и оканчиваются одной и той же литерой. Ввод-вывод осуществлять с помощью классов С++.

Вариант 29

Пусть текстовый файл разбит на непустые строки. Написать программу для подсчета числа строк, которые состоят из одинаковых литер. Ввод-вывод осуществлять с помощью средств С.

Вариант 30

Написать программу, переписывающую содержимое текстового файла t2 в текстовый файл t1 (с сохранением деления на строки). Ввод-вывод осуществлять с помощью классов С++.

1.4 Порядок выполнения работы

1.4.1 В ходе самостоятельной подготовки изучить основы работы со строками и файлами в языке С/С++.

1.4.2 Выбрать способ представления исходных данных задачи и результатов.

1.4.3 Разработать алгоритм решения задачи, разбив его при необходимости на отдельные функции.

1.4.4 Разработать программу на языке С/С++.

1.4.5 Разработать тестовые примеры, следуя указаниям, изложенным в разделе 3.

1.4.6 Выполнить отладку программы.

1.4.7 Получить результаты работы программы и исследовать её свойства для различных режимов работы, сформулировать выводы.

1.5 Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

1.6 Контрольные вопросы

1.6.1 Приведите примеры описания и инициализации строк в языке С.

1.6.2 Как хранится строка языка С в памяти ЭВМ?

1.6.3 Объясните назначение и запишите прототипы функций **gets, puts**.

1.6.4 Как выделить динамическую память под строку?

1.6.5 Как выполнить ввод и вывод строки с помощью функций **scanf()** и **printf()**?

1.6.6 Как выполнить ввод и вывод строк с помощью объектов **cin** и **cout**?

1.6.7 Как ввести строку с помощью функций **getline** и **get** объекта **cin**?

1.6.8 Для чего предназначены функции **strcpy()** и **strncpy()**? Как их вызвать?

1.6.9 Для чего предназначена функция **strcmp()**? Как её вызвать? Какие результаты она возвращает?

1.6.10 Объясните, как открыть файл? Какие имеются режимы открытия файлов в языке С?

1.6.11 Опишите функции **fgets** и **fputs**. В чем их отличие от функций **gets** и **puts**?

1.6.12 Приведите примеры чтения и записи значений в файл с помощью функций **fscanf()** и **fprintf()**?

1.6.13 Приведите примеры вызова функций **fread()** и **fwrite()**. Объясните назначение аргументов этих функций.

1.6.14 Как выполняется изменение и чтение счетчика текущей позиции файла в языке С?

1.6.15 Как открыть файловый поток в режиме чтения в языке С++?

1.6.16 Как открыть файловый поток в режиме записи в языке С++?

1.6.17 Объяснить назначение функций **seekg()** и **seekp()**. Приведите примеры их вызова.

1.6.18 Приведите примеры вызова функций **read()** и **write()** языка С++.

2 ЛАБОРАТОРНАЯ РАБОТА №6

ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРУКТУРАМИ И БИНАРНЫМИ ФАЙЛАМИ

2.1. Цель работы

Изучение способов описания структур данных на языке С. Исследование особенностей обработки бинарных файлов, хранящих структурные типы данных.

2.2. Краткие теоретические сведения

2.2.1. Структуры

Структуры языка С, аналогично комбинированному типу языка Паскаль, объединяют в себе компоненты (поля) разных типов. Описание структуры начинается с ключевого слова **struct** и содержит список описаний полей в фигурных скобках. За словом **struct** может следовать имя структуры, которое рассматривается как имя структурного типа:

```
struct <имя структуры>
{  <тип1> <поле 1>;
  <тип2> <поле 2>;
  . . .
  <тип n> <поле n>;
}
```

Например, пусть требуется фиксировать сведения о расписании работы секций конференции. При этом для каждого мероприятия фиксируется время, тема, фамилия организатора и количество участников. Для этого можно описать следующую структуру:

```
struct section
{  int hour, min;
     char theme[100];
     char name[30];
     int num;
  } s1,s[10];
```

Здесь **section —** это *имя типа*. Переменные структурного типа описываются либо одновременно с описанием структуры (см. выше), либо отдельно. Например:

```
section s1,s[10];   // структура и массив структур
```

При совместном описании структурного типа и переменных допускается имя структуры не указывать, если оно нигде не используется.

Переменные структурного типа можно размещать и в динамической области памяти, для этого надо описать указатель на структуру:

```
section *sptr=new section;              //указатель на структуру
  section *mptr=new section[m];          //указатель на массив структур
```

Поля структуры могут быть любого основного типа, массивом, указателем, структурой или объединением. Для доступа к полям структуры используется операция выбора, обозначаемая точкой:

<имя структурной переменной>.<имя поля>

Например:

```
s1.hour=9;
s1.min=30;
strcpy(s1.theme,”Архитектура компьютеров”);
s[1].hour=13;
s[1].min=0;
strcpy(s[1].theme,”Программное обеспечение”);
```

Если структуры размещены в динамической памяти, то доступ к полям выполняют через указатели:

```
(*sptr).hour=9;
(*sptr).min=30;
```

Скобки здесь необходимы, поскольку приоритет операции **“.”** выше, чем у операции раскрытия ссылки. Существует еще одна форма доступа к полям структуры при работе с указателями на структуру:

```
sptr->num=30;
```

Если имеется указатель на массив структур, то доступ к ним выполняется так:

```
mptr[2].hour=15; //(*(mptr+2)).hour=15;
```

Структуры одного типа можно присваивать друг другу:

```
*sptr=s1;
mptr[1]=s1;
mptr[2]=s[0];
```

Ввод-вывод структур, как и массивов, выполняется поэлементно. Например, ввод-вывод структуры **s1** с использованием классов языка C++ можно выполнить следующим образом:

```
cin >> s1.hour >> s1.min;
cin.getline(s1.theme,100);
cout << s1.hour << s1.min << ‘ ‘ << s1.theme << endl;
```

Этот же ввод-вывод можно выполнить и с помощью функций ввода-вывода языка C:

```
scanf(“%d%d”,&s1.hour,&s1.min);
gets(s1.theme);
printf(“%d%d%s”,s1.hour,s1.min,s1.theme);
```

Статические структуры можно инициализировать перечислением значений их полей:

```
section s2={10,30,”Архитектура компьютеров”,25};
```

1.2.2. Пример программы обработки структур

Имеются записи о сотрудниках, содержащие фамилию и инициалы, год рождения и оклад сотрудника. Требуется написать программу с использованием самостоятельно определяемых функций, которая

- вводит записи о сотрудниках с клавиатуры в бинарный файл и упорядочивает его по фамилии;
- осуществляет поиск сотрудников в файле по фамилии и вычисляет средний оклад этих сотрудников;
- отображает содержимое бинарного файла.

Исходные данные и результаты.

Исходные данные:

Сведения об одном сотруднике будем представлять в виде структуры. При этом фамилию и инициалы представим строкой из 15 символов, год рождения – целым типом, оклад – вещественным типом. Поскольку количество записей о сотрудниках не оговорено будем хранить все сведения в бинарном файле, а не массиве.

Результаты:

В результате работы программы требуется вывести на экран требуемые элементы файла. Так как результаты представляют собой выборку из исходных данных, дополнительная память для них не отводится. Кроме того, необходимо подсчитать средний оклад для найденных сотрудников. Для этого введем переменную вещественного типа.

Алгоритм решения задачи.

1. Ввести с клавиатуры сведения о сотрудниках в бинарный файл. При этом будем прекращать ввод, если пользователь введет символ ‘*’.
2. Выполнить сортировку файла по Ф.И.О., используя возможность прямого доступа к элементам бинарного файла. В качестве метода сортировки будем использовать метод прямого обмена (пузырьковую сортировку);
3. Обеспечить вывод сведений о сотруднике:
   a) ввести с клавиатуры фамилию;
   b) выполнить поиск сотрудника в файле по фамилии;
   c) увеличить суммарный оклад и счетчик количества сотрудников;
   d) вывести сведения о сотруднике или сообщение о том, что его нет в списке;
   e) вывести средний оклад.
4. Отобразить содержимое бинарного файла.

Каждый из указанных пунктов алгоритма представим в виде функции, которую можно будет вызывать из основной программы. При этом контроль за порядком вызова функций полностью возлагается на пользователя.

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>
```

```
//----------------------константы и структуры-----------------------------
const int len_fio=15;                   //длина строки с Ф.И.О.
struct person
{       char fio[len_fio];              // Ф.И.О
        int year;                       //год рождения
        float salary;                   //зарплата
};
const int size_p=sizeof(person);        //размер структуры
//------------------------прототипы функций-----------------------------
int create_file(FILE *fbin);            //запись в файл
int sort_file(FILE *fbin);              //сортировка файла
int print_file(FILE *fbin);             //вывод файла
int search_persons(FILE *fbin);         //поиск сотрудника и вычисления
//-------------------------основная функция-----------------------------
int main()
{       FILE *fbin;
        char c;
        fbin=fopen("person.dat","r+b");   //Открытие существующего файла
                                          //для чтения и записи в конец
        if (!fbin) {
                fbin=fopen("person.dat","w+b");  //Создание нового файла
                                                 //для обновления
                    if(!fbin){
                    puts("Не могу открыть (создать) файл\n");
                    return 1;
                }
 }//вывод меню и запуск соответствующих функций
  while (1)
    {           clrscr();
                puts("1- Запись в файл");
                puts("2- Сортировка файла");
                puts("3- Вывод файла");
                puts("4- Поиск и вычисления”);
                puts("5- Выход");
                puts("_____________________________");
                puts("Введите номер пункта меню\n");
                c=getch();
                switch (c)
              {  case '1':create_file(fbin);break;
                 case '2':sort_file(fbin);break;
                 case '3':print_file(fbin);break;
                 case '4':search_persons(fbin);break;
                 case '5':return 0;
               }
        }
```

```
}
//-------------------------запись в файл-----------------------------------
int create_file(FILE *fbin){
        person elem;
        fseek(fbin,0,SEEK_END);     //указатель в конец файла
        puts("Ввод данных о сотрудниках");
        puts("Для выхода введите символ *");
        puts("_____________________________________\n");
        while (1) {
                puts("Введите Ф.И.О. (обязательно с инициалами)");
                getline(elem.fio,len_fio);        //ввод Ф.И.О.
                if (!strcmp(elem.fio,"*"))        //если введена “*”,
                        return 1;                 // то выход
                puts("Введите год рождения");
                scanf("%i",&elem.year);           //ввод года рождения
                puts("Введите зарплату");
                scanf("%f",&elem.salary);         //ввод зарплаты
                fwrite(&elem,size_p,1,fbin);      //запись структуры в файл
        }
}
//-------------------------вывод файла-------------------------------------
int print_file(FILE *fbin){
        person elem;
        int n;
        clrscr();
        rewind(fbin);                             //указатель в начало файла
        puts("Ф.И.О.        Год   Зарплата");
        do {
                n=fread(&elem,size_p,1,fbin);//чтение структуры из файла
                if (n<1) break;                   //если n<1, то конец файла
                printf("%-15s%-6i%-8.2f\n",elem.fio,elem.year,elem.salary);
           } while (1);
        puts("_____________________________");
        puts("Нажмите любую клавишу");
        getch();
        return 0;
}
//-------------------------сортировка записей в файле----------------------
int sort_file(FILE *fbin) {
        long i,j;
        person elem1,elem2;

        puts("Для сортировки нажмите любую клавишу");
        getch();
        fseek(fbin,0,SEEK_END);     //указатель в конец
```

```
        long len=ftell(fbin)/size_p;     //определяем длину файла
        rewind(fbin);                    //указатель в начало
        //пузырьковая сортировка
        for(i=len-1;i>=1;i--)
          for (j=0;j<=i-1;j++) {
              fseek(fbin,j*size_p,SEEK_SET);    //указатель на j-ую запись
              fread(&elem1,size_p,1,fbin);      //читаем запись j в elem1
              fread(&elem2,size_p,1,fbin);      //читаем след. запись в elem2
              if (strcmp(elem1.fio,elem2.fio)>=1) {     //сравниваем Ф.И.О.
                 fseek(fbin,(-2)*size_p,SEEK_CUR);  //указатель на 2 поз. назад
                 //обмен значений
                 fwrite(&elem2,size_p,1,fbin);     //сначала записываем elem2
                 fwrite(&elem1,size_p,1,fbin);     // затем записываем elem1
              }
          }
        puts("Для выхода нажмите любую клавишу");
        getch();
        return 0;
}
// ———поиск сотрудников и вычисление средней зарплаты———
int search_persons(FILE *fbin){
        int not_found;                   //флаг  поиска
        char s[len_fio];                 //строка для ввода фамилии
        int n_persons=0;                 //счетчик сотрудников
        int n;
        float summa_salary=0;            //сумма зарплат
        person elem;
        while (1) {
              puts("Введите фамилию или  * ");
              cin.getline(s,len_fio);           //запоминаем фамилию в строке s
              if (!strcmp(s,"*"))  break;       //выход, если ввели *
              rewind(fbin);                     //указатель в начало файла
              not_found=1;                      //флаг – сотрудник не найден
              do {
                n=fread(&elem,size_p,1,fbin);       //читаем запись
                if (n<1) break;                     // если n<1, то конец файла
                if (strstr(elem.fio,s))             //ищем строку s в поле fio
                  if (elem.fio[strlen(s)]==' ') {//есть пробел после фамилии ?
                    strcpy(s,elem.fio);             //копируем fio в s
                    puts("_____________________________");
                    puts("Ф.И.О.        Год   Зарплата");
                    printf("%-15s%-6i%-8.2f\n",elem.fio,elem.year,elem.salary);
                    puts("_____________________________");
                    n_persons+=1;                   //счетчик сотрудников
                    summa_salary+=elem.salary;      //суммирование зарплат
```

```
                not_found=0;                    //сотрудник найден
              }
        } while (1);
        if (not_found)
        puts("Такого сотрудника нет в файле");
    }
    if (n_persons>0)                    //вычисление средней зарплаты
        printf("Средняя зарплата=%8.2f\n",summa_salary/n_persons);
    puts("_____");
    puts("Нажмите любую клавишу");
    getch();
    return 0;
}
```

Программа достаточно подробно прокомментирована. В тексте основной программы осуществляется либо открытие файла (режим “**r+b**”), либо создание файла (режим “**w+b**”) для обновления, если файл “**person.dat**” на диске не обнаружен. Это позволяет один раз ввести исходные данные в файл, а затем его многократно использовать. Если потребуется начать работу с новым файлом, то старый файл следует удалить с диска средствами операционной системы.

Все необходимые действия с файлом выполняются с помощью функций, вызываемых из пунктов меню, отображаемых на экране. При заполнении файла требуется обязательно вводить инициалы, которые должны отделяться от фамилии пробелом. Фамилия должна начинаться с первой позиции каждой строки. Это свойство используется в процедуре поиска сотрудников.

1.3. Варианты заданий

Вариант 1

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию номера группы;
- чтение данных из этого файла, корректировку данных в файле по номеру записи;
- вывод на дисплей фамилий и номеров групп для всех студентов, если средний балл студента больше 4.0;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 2

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по возрастанию среднего балла;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 3

Описать структуру с именем STUDENT, содержащую следующие поля:

- фамилия и инициалы;
- номер группы;
- успеваемость (массив из пяти элементов).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 4

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть упорядочены по возрастанию номера рейса;
- чтение данных из этого файла;
- вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 5

Описать структуру с именем AEROFLOT, содержащую следующие поля:

- название пункта назначения рейса;
- номер рейса;
- тип самолета.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;

- чтение данных из этого файла;
- вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры;
- если таких рейсов нет, выдать на дисплей соответствующее сообщение.

Вариант 6

Описать структуру с именем WORKER, содержащую следующие поля:
- фамилия и инициалы работника;
- название занимаемой должности;
- год поступления на работу.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из структур типа WORKER; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры;
- если таких работников нет, вывести на дисплей соответствующее сообщение.

Вариант 7

Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть размещены в алфавитном порядке по названиям пунктов назначения;
- чтение данных из этого файла;
- вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 8

Описать структуру с именем TRAIN, содержащую следующие поля:
- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по времени отправления поезда;
- чтение данных из этого файла;
- вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры;

- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 9

Описать структуру с именем TRAIN, содержащую следующие поля:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN; записи должны быть упорядочены по номерам поездов;
- чтение данных из этого файла;
- вывод на экран информации о поезде, номер которого введен с клавиатуры;
- если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 10

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршруте, номер которого введен с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 11

Описать структуру с именем MARSH, содержащую следующие поля:

- название начального пункта маршрута;
- название конечного пункта маршрута;
- номер маршрута.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа MARSH; записи должны быть упорядочены по номерам маршрутов;
- чтение данных из этого файла;
- вывод на экран информации о маршрутах, которые начинаются или оканчиваются в пункте, название которого введено с клавиатуры;
- если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 12

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;

- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о человеке, номер телефона которого введен с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 13

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть размещены по алфавиту;
- чтение данных из этого файла;
- вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 14

Описать структуру с именем NOTE, содержащую следующие поля:

- фамилия, имя;
- номер телефона;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа NOTE; записи должны быть упорядочены по трем первым цифрам номера телефона;
- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 15

Описать структуру с именем ZNAK, содержащую следующие поля:

- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;

- чтение данных из этого файла;
- вывод на экран информации о человеке, чья фамилия введена с клавиатуры;
- если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 16

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по датам рождения;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 17

Описать структуру с именем ZNAK, содержащую следующие поля:
- фамилия, имя;
- знак Зодиака;
- дата рождения (массив из трех чисел).

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа ZNAK; записи должны быть упорядочены по знакам Зодиака;
- чтение данных из этого файла;
- вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры;
- если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 18

Описать структуру с именем PRICE, содержащую следующие поля:
- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью функций:
- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям товаров;
- чтение данных из этого файла;
- вывод на экран информации о товаре, название которого введено с клавиатуры;
- если таких товаров нет, выдать на дисплей соответствующее сообщение

Вариант 19

Описать структуру с именем PRICE, содержащую следующие поля:

- название товара;
- название магазина, в котором продается товар;
- стоимость товара в грн.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа PRICE; записи должны быть размещены в алфавитном порядке по названиям магазинов;
- чтение данных из этого файла;
- вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры;
- если такого магазина нет, выдать на дисплей соответствующее сообщение.

Вариант 20

Описать структуру с именем ORDER, содержащую следующие поля:

- расчетный счет плательщика;
- расчетный счет получателя;
- перечисляемая сумма в грн.

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из элементов типа ORDER; записи должны быть размещены в алфавитном порядке по расчетным счетам плательщиков;
- чтение данных из этого файла;
- вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры;
- если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

Вариант 21

Описать структуру с именем STUDENT, содержащую следующие поля:

- номер;
- фамилия и имя;
- год рождения;
- год поступления в университет;
- структура OCENKI, содержащая четыре поля: физика, математика, программирование, история;

Написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов отличников;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 22

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 3;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 23

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все двойки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 24

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших все пятерки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 25

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей анкетных данных студентов, получивших одну оценку 4, а все остальные – 5;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 26

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по алфавиту;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘A’, и их оценки;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 27

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’, и год их рождения;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 28

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’или ‘Г’, и год их поступления;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 29

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по фамилиям;
- чтение данных из этого файла;
- вывод на дисплей фамилий студентов, имеющих средний балл выше среднего балла группы;
- если таких студентов нет, вывести соответствующее сообщение.

Вариант 30

Для структуры, указанной в варианте 21, написать программу, выполняющую следующие действия с помощью функций:

- ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT; записи должны быть упорядочены по номеру;
- чтение данных из этого файла;
- вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки;
- если таких студентов нет, вывести соответствующее сообщение.

1.4. Порядок выполнения работы

1.4.1. В ходе самостоятельной подготовки изучить основы работы со структурами и бинарными файлами в языках C/C++.

1.4.2. Выбрать способ представления исходных данных задачи и результатов Разработать алгоритм решения задачи, разбив его на отдельные функции.

1.4.3. Разработать программу на языке C/C++.

1.4.4. Разработать тестовые примеры, следуя указаниям раздела 3.

1.4.5. Выполнить отладку программы.

1.4.6. Получить результаты работы программы и исследовать её свойства в различных режимах работы, сформулировать выводы.

1.5. Содержание отчета

Цель работы, вариант задания, структурная схема алгоритма решения задачи с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

1.6. Контрольные вопросы

1.6.1. Что называют структурой в языке C/C++?

1.6.2. Приведите пример описания структурного типа.

1.6.3. Как описать структурный тип совместно и раздельно с соответствующей переменой?

1.6.4. Как описать массив структур? Для чего его применяют?

1.6.5. Как выполняется доступ к полям статических структурных переменных?

1.6.6. Как выполняется доступ к полям динамических структурных переменных?

1.6.7. Как выделяют динамическую память под структуры?

1.6.8. Покажите на примерах ввод-вывод структур с помощью функций языка C.

1.6.9. Покажите на примерах ввод-вывод структур с помощью классов языка C++.

1.6.10. Как выполняется инициализация структур?

Как1.6.11. выполнить чтение и запись структурных переменных в бинарные файлы?

1.6.12. Как обнаружить конец файла при использовании функции **fread**?

1.6.13. Как определить длину файла с помощью функций **seek** и **ftell**?

3. ЛАБОРАТОРНАЯ РАБОТА №7
ПРОГРАММИРОВАНИЕ ЛИНЕЙНЫХ СПИСКОВ НА ЯЗЫКЕ С/С++

3.1. Цель работы

Изучение списковых структур данных и приобретение навыков разработки и отладки программ, использующих динамическую память. Исследование особенностей организации списков средствами языка С/С++.

3.2. Краткие теоретические сведения

3.2.1. Динамические структуры данных

В предыдущих лабораторных работах были рассмотрены задачи, в которых объем памяти, необходимый программе, был известен либо до компиляции программы, либо на этапе ввода данных. В первом случае память резервировалась с помощью операторов описания, во втором случае — с помощью функций выделения памяти. В каждом из этих случаев выделялся непрерывный участок памяти.

Если до начала работы программы невозможно определить, сколько памяти потребуется для хранения данных, то память выделяется по мере необходимости отдельными блоками, которые связывают друг с другом при помощи указателей. Такой способ выделения памяти предполагает организацию в памяти ЭВМ *динамических структур данных*, поскольку их размер изменяется во время выполнения программы. Из динамических структур в программах чаще всего используются различные линейные списки, стеки, очереди, деревья. Они различаются способами связи отдельных элементов и допустимыми операциями над ними. Динамическая структура может занимать несмежные участки оперативной памяти. В процессе работы программы элементы структуры могут по мере необходимости добавляться и удаляться.

Напомним, что *линейным списком* называется структура данных, при которой логический порядок следования элементов задается путем ссылок, т.е. каждый элемент списка содержит указатель на следующий элемент (предыдущий элемент). Доступ к первому элементу списка выполняется с помощью специального указателя – указателя на начало (голову) списка.

Односвязным линейным списком называют список, в котором предыдущий элемент ссылается на следующий. *Двусвязный линейный список* – это список, в котором предыдущий элемент ссылается на следующий, а следующий – на предыдущий. *Односвязный циклический список* – это односвязный линейный список, в котором последний элемент ссылается на первый. *Стек* – это односвязный список, в котором компоненты добавляются и удаляются только со стороны вершины списка. *Очередь* – это односвязный список, в котором компоненты добавляются в конец списка, а удаляются со стороны вершины списка.

Элемент любой динамической структуры данных состоит из полей, часть из которых предназначена для связи с соседними элементами. Например, элемент

линейного списка, предназначенного для хранения целых чисел, можно описать следующим образом:

```
struct Element {
int d;
struct element *next;
}
```

Здесь поле **next** является указателем на структуру своего собственного типа и предназначено для связи элементов друг с другом.

Определив необходимые указатели, напишем фрагмент программы, выполняющей организацию списка:

```
element *beg=NULL;   //указатель на начало списка
element *temp;
int x;
while (1)
 { scanf("%d",&x);          //ввод x
   if (x==9999) break;
   temp=new element;   //выделение памяти под элемент
   temp->d=x;          //присваивание значений  полю d
   temp->next=beg;     //установление связи с предыдущим элементом
  beg=temp;            //beg указывает на последний введенный эл-т
}
```

Здесь в цикле с клавиатуры вводятся числа. Если введено значение 9999, то происходит выход из цикла. Затем с помощью оператора **new** динамически выделяется память под один элемент списка, и адрес этой области памяти запоминается в указателе **temp**. В поле **d** вновь созданного элемента копируется значение **x**, а в поле **next** – значение указателя **beg**. После этого указателю **beg** присваивается значение **temp**. В итоге в памяти создается список, на начало которого указывает **beg** (рисунок 3.1).

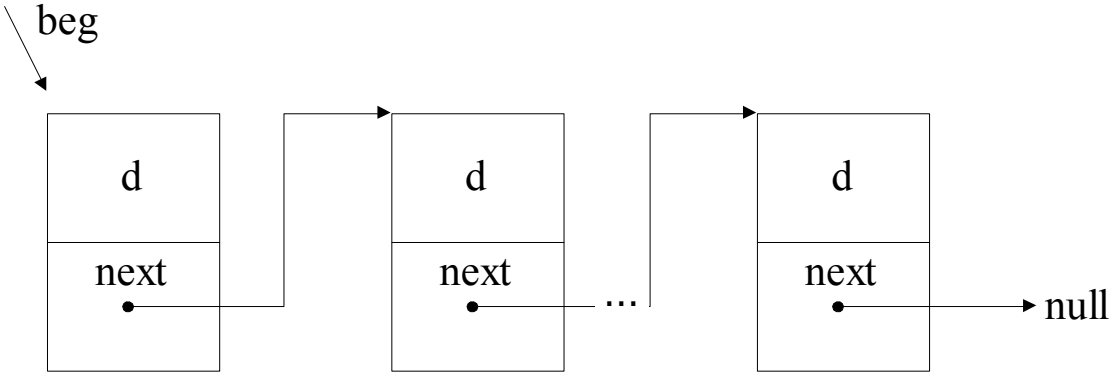

Рисунок 3.1 — Линейный список

Над списками выполняют следующие операции:

- добавление элемента в список;
- исключение элемента из списка;
- просмотр значений элементов списка;
- поиск заданного значения в списке и др.

Например, добавление элемента в указанный список можно выполнить так, как показано в приведенном выше фрагменте программы.

Исключение элемента, на который указывает указатель **beg,** выполняется так:

```
temp=beg;          //запоминание указателя
beg=beg->next;     //перемещение beg на следующий элемент
delete temp;             //освобождение памяти
```

Просмотр значений элементов списка можно выполнить следующим образом:

```
temp=beg;                    //копирование указателя beg
while (temp) {               // пока temp не равен NULL
printf("%d\n",temp->d);      //вывод значения поля d по указателю temp
temp=temp->next;             //перемещение temp на следующий элемент
}
```

3.2.2. Пример программы обработки очереди

Рассмотрим программу, выполняющую организацию очереди, добавление элемента в очередь, удаление элемента из очереди, просмотр очереди.

Очередь – это линейный список, добавление элемента в который выполняется с одной стороны, а исключение – с другой. При этом используются специальные указатели начала и конца очереди. В примере элементами очереди являются структуры данных о заявках на ремонт автомобилей. В заявке указывается фамилия владельца, марка автомобиля, вид работы, дата приема заказа и стоимость работы. После выполнения работы распечатывается квитанция.

Интерфейс программы организуем в виде меню:

1) добавление заявки в очередь;
2) печать и удаление заявки из очереди;
3) просмотр списка заявок;
4) сохранение списка заявок в файле;
5) выход.

Для задания длины строк определим символические константы, чтобы при необходимости можно было легко их изменять.

Для работы с очередью введем два указателя: **beg** – указатель начала очереди и **end** – указатель конца очереди.

Работу с очередью выполняют три функции. Функция **dob_first** формирует первый элемент очереди и возвращает указатель на него. Функция **dob** выполняет добавление элемента в конец очереди, поэтому ей передается указатель на конец очереди и элемент, который следует добавить, а возвращает она измененный указатель на конец очереди.

Удаление элементов выполняется с головы (начала) очереди функцией **udal**. Для этого функции передается указатель **beg**. После удаления элемента функция возвращает измененное значение **beg**. В ходе удаления элемента вызывается функция **print**, осуществляющая вывод на экран сведений о выполненном за-

казе. При этом продемонстрирована работа с датами с помощью функций, содержащихся в головном файле **<time.h>**.

В функциях **dob_first**, **dob** и **print** параметр-структура передается как константная ссылка. Это исключает лишнее копирование структур и их возможную модификацию в теле функций.

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <iomanip.h>
#include <conio.h>
#includt <iostream>
using namespace std;
//--------------------------константы-----------------------------------------
const int d_n=20,                    //длина Ф.И.О.
          d_m=20,                    //длина названия модели автомобиля
          d_w=80;                    //длина строки описания работ

//----------------------структура элемента-------------------------------------
struct zakaz
{       char name[d_n],              //Ф.И.О.
             model[d_m],             //название модели автомобиля
             work[d_w];              //описание работы
        time_t time;                 //время приема заказа
        float price;                 //стоимость
        zakaz* next;                 //указатель на следующий элемент
};
//-----------------------прототипы функций--------------------------------------
zakaz* dob(zakaz* end, const zakaz& z); //добавление элемента в очередь
zakaz* dob_first(const zakaz& z);       //добавление первого элемента
zakaz* udal(zakaz* beg);                //удаление элемента из очереди
zakaz vvod_zakaza();                    //ввод значений элемента
int  menu();                            //отображение меню и ввод его пункта
void print(const zakaz& z);             //вывод значений  элемента
void prosmotr(zakaz* beg);              //просмотр очереди
int  read_file(char* filename, zakaz** beg, zakaz** end);   //чтение файла
int  write_file(char* filename, zakaz* temp);               //запись в файл
//-----------------------основная функция----------------------------
int main()
{    zakaz *beg=0,                          //указатель начала очереди
           *end=0;                          //указатель конца очереди
     char *filename="zakaz.txt";            //имя файла
     time_t now;                            // текущее время
     tm t;                                  // дата
     read_file(filename,&beg, &end);        //чтение данных из файла в очередь
```

```
    while (1)
{       switch (menu())//отобразить меню и сделать выбор
   {    case 1:                          //пункт меню 1 – добавление элемента
          if (beg)                       //если очередь не пустая,
              end=dob(end,vvod_zakaza());//то добавляем элемент в конец,
          else {                         //иначе
              beg=dob_first(vvod_zakaza());//создаем первый элемент очереди.
              end=beg;                    //указатели начала и конца равны
              }
          break;
        case 2:                          //пункт меню 2 –
          beg=udal(beg);                 //печать и удаление элемента
          now = time(0);                 //получить текущее время в сек.
          t = *(localtime(&now));        //преобразовать  в дату
          cout<<"Дата выполнения заказа: " //вывод даты
              <<t.tm_mday<<'.'<<t.tm_mon+1<<'.'<<1900+t.tm_year<<endl;
          cout<<"Нажмите любую клавишу"<<endl;
          cin.get();
          break;
        case 3:                          // пункт меню 3 –
          prosmotr(beg);                 //просмотр очереди
          break;
        case 4:                          //пункт меню 4 –
          write_file(filename,beg);      //запись в файл
          break;
        case 5:                          //пункт 5 –выход
          return 0;
        default:                         //если неверно введен номер пункта
          cout<<"Вам следует ввести номер от 1 до 5"<< endl;
          cin.get();
          break;
        }
    }
}
//----------------------добавление элемента----------------------------
zakaz* dob(zakaz *end,const zakaz& z)    //возвращает указатель на доб. эл-т
{       zakaz *newE=new zakaz;           //выделение памяти под элемент
        *newE=z;                         //присвоить значения элементу
        newE->next=0;                    //ссылка на следующий эл-т нулевая
        end->next=newE;                  //добавить эл-т в очередь
        end=newE;                        //установить указатель end на newE
        return end;                      //вернуть измененное значение end
}
//----------------------создание первого элемента-----------------------
zakaz* dob_first(const zakaz& z)
```

```
{       zakaz *beg=new zakaz;           //выделение памяти под элемент
        *beg=z;                         //присвоить значения элементу
        beg->next=0;                    //ссылка на следующий эл-т нулевая
        return beg;                     //вернуть указатель на элемент
}
// ---------- печать и удаление элемента-------------------------
zakaz* udal(zakaz *beg)
{       zakaz *temp;
        if (!beg) { cout<<"Очередь пустая"<<endl; return 0; }
        cout<<"==============================="<<endl;
        print(*beg);                    //печать значений головного эл-та
        cout<<"==============================="<<endl;
        temp=beg;                       //запомнить указатель на голову
        beg=beg->next;                  //перевод beg на следующий эл-т
        delete temp;                    //удаление элемента
        return beg;                     //вернуть измененное значение beg
}
//------------------ввод значений элемента----------------------------
zakaz vvod_zakaza()
{   char buf[10];
        zakaz z;
        cout<<"Введите Ф.И.О."<<endl;
        cin.getline(z.name,d_n);
        cout<<"Введите модель автомобиля"<<endl;
        cin.getline(z.model,d_m);
        cout<<"Введите описание работы"<<endl;
        cin.getline(z.work,d_w);
        z.time=time(0);                 //зафиксировать время в поле time
        do                              //проверка ввода числа
         {  cout<<"Введите стоимость работы"<<endl;
           cin>>buf;
        } while (!(z.price=(float)atof(buf)));
        return z;
}
//----------------отображение меню и  ввод пункта меню----------------------
int menu() {
        char buf[10];
        int item;
        do
         {clrscr();
           cout<<endl;
           cout<<"===СИСТЕМА УЧЕТА ЗАКАЗОВ==="<<endl<<endl;
           cout<<"1- Добавление элемента в очередь"<<endl;
           cout<<"2- Печать и удаление элемента"<<endl;
           cout<<"3- Просмотр очереди"<<endl;
```

```
        cout<<"4- Запись данных в файл"<<endl;
        cout<<"5- Выход"<<endl;
        cout<<"============================"<<endl;
        cout<<"Введите номер пункта меню"<<endl;
        cin>>buf;                       //прочитать введенное значение
        cin.get();
        item=atoi(buf);                 //преобразовать его в целое
        if (!item) {                    //анализ ошибки ввода
                cout<<"Вам следует вводить число от 1 до 5"<<endl;
                cin.get();
                }
    } while (!item);                    //повторять пока не введет число
    return item;                        //вернуть номер введенного пункта меню
}
// ----------------------------печать заказа----------------------------
void print(const zakaz& z)
{   tm t=*(localtime(&z.time));
    cout<<"Ф.И.О.: " <<z.name<<endl;
    cout<<"Модель автомобиля: "<<z.model<<endl;
    cout<<"Описание работ: "<<z.work<<endl;
    cout<<"Дата  приема: "
       <<t.tm_mday<<'.'<<t.tm_mon+1<<'.'<<1900+t.tm_year<<endl;
    cout<<"Стоимость: "<<z.price<<endl;
}
//----------------------------просмотр очереди--------------------------
void prosmotr(zakaz *beg)
{   if (!beg) { cout<<"Очередь пустая"<<endl; return;  }
    zakaz *temp=beg;                //указатель temp устанавливаем в начало
    cout<<"============================="<<endl;
    while (temp) {                  //просматриваем пока temp!=0
        print(*temp);               //печатаем значения элемента по указателю
        cout<<"============================="<<endl;
        cout<<"Нажмите любую клавишу"<<endl;
        cin.get();
        temp=temp->next;            //перемещаем temp на следующий эл-т
    }
}
// ----------------------------чтение из файла----------------------------
int read_file(char* filename,zakaz** beg, zakaz** end)
{   ifstream fin(filename,ios::in | ios::nocreate);     //открытие файла
    if (!fin) {cout<<"Нет файла"                        //выход если нет файла
                   <<filename<<endl; return 1;}
    zakaz z;
    *beg = 0;
    while (fin.getline(z.name,d_n))     //чтение фио пока не конец файла
```

```
        {   fin.getline(z.model,d_m);          //чтение марки  автомобиля
            fin.getline(z.work,d_w);           //чтение описания работы
            fin>>z.time>>z.price;              //чтение времени и стоимости
            fin.get();
            if (*beg)                          //если очередь не пустая,
                 *end=dob(*end,z);             //то добавляем элемент в конец,
            else                               //иначе
                {*beg=dob_first(z); *end=*beg;} //создаем первый элемент
        }
        return 0;
}
// ---------------------------запись в файл ---------------------------
int write_file(char* filename, zakaz* temp)
{       ofstream fout(filename);               //открытие файла
        if (!fout) {cout<<"Не могу открыть файл для записи"<<endl; return 1;}
        while (temp)                          //пока temp!=0 выводим эл-ты в файл
          {  fout<<temp->name<<endl;          //вывод фио
             fout<<temp->model<<endl;         //вывод марки автомобиля
             fout<<temp->work<<endl;          //вывод описания работ
             fout<<temp->time<<' '<<temp->price<<endl;      //время и стоимость
             temp=temp->next;                 //переместить указатель на след. эл-т
        }
        cout<<"Данные сохранены в файле: "<<filename<<endl;
        cout<<"============================="<<endl;
        cout<<"Нажмите любую клавишу"<<endl;
        cin.get();
        return 0; }
```

Текущую дату и время можно получить с помощью функции **time**. Она возвращает ее в виде секунд, прошедших с полуночи 1 января 1970г. Функция **localtime** преобразует эту величину в стандартную структуру **tm**, определенную в том же головном файле, и возвращает указатель на эту структуру. Поля структуры содержат: год (количество лет, прошедших с 1900г.), месяц (от 0 до 11), день (от 1 до 31), час (0-23),минуту (0-59) и секунду (0-59).

Ввод-вывод организован с помощью классов C++. Обратите внимание на использование в ряде функций метода **cin.get()**. Он необходим для считывания из входного потока признака конца строки.

Функция **read_file** обеспечивает чтение элементов очереди из файла и её организацию в памяти ЭВМ. В качестве входного параметра ей передается указатель на имя файла, а в качестве результата она возвращает через список своих параметров значения указателей **beg** и **end**. Поэтому эти два параметра передаются в функцию по адресу, т.е. как указатель на указатель. Альтернативный способ передачи указателя по адресу – с использованием ссылки на указатель приведен в примере к следующей лабораторной работе.

В программе предусмотрена защита от неправильного ввода данных. Если пользователь введет неверный пункт меню или нечисловые символы там, где это

делать нельзя, программа корректно обработает эту ситуацию (см. функции **menu** и **vvod_zakaza**).

3.3. Варианты заданий

Представить одну из приведенных ниже таблиц в виде линейного списка L, элементами которого являются строки таблицы. Написать функции организации, добавления элемента в список, исключения элемента из списка, просмотра списка, а также одну из функций в соответствии с вариантом, приведенным ниже.

Значения и количество записей в таблице студент выбирает самостоятельно. Исходные данные после организации списка должны сохраняться в файле и при повторном запуске программы считываться из файла. Количество строк таблицы не задается.

Таблица 3.1 — Ведомость

N	Фамилия	Имя	Отчество	Оценки		
				Математика	История	Физика

Таблица 3.2 — Расписание

N	Поезда	Станция отправления	Станция назначения	Время отправления	Время прибытия	Стоимость билета

Таблица 3.3 — Анкета

N	ФИО	Год рождения	Пол	Семейное состояние	Количество детей	Оклад

Вариант 1

Функцию, которая вставляет в начало очереди новый элемент.

Вариант 2

Таблица 3.2. Функцию, которая вставляет в начало очереди новый элемент.

Вариант 3

Таблица 3.3. Функцию, которая вставляет в начало очереди новый элемент.

Вариант 4

Таблица 3.1. Функцию, которая вставляет в конец стека новый элемент.

Вариант 5

Таблица 3.2. Функцию, которая вставляет в конец стека новый элемент.

Вариант 6

Таблица 3.3. Функцию, которая вставляет в конец стека новый элемент.

Вариант 7

Таблица 3.1. Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 8

Таблица 3.2. Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 9

Таблица 3.3. Функцию, которая вставляет новый элемент E после первого элемента непустого списка L.

Вариант 10

Таблица 3.1. Функцию, которая вставляет в список L новый элемент E1 за

каждым вхождением элемента E.

Вариант 11

Таблица 3.2. Функцию, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

Вариант 12

Таблица 3.3. Функцию, которая вставляет в список L новый элемент E1 за каждым вхождением элемента E.

Вариант 13

Таблица 3.1. Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 14

Таблица 3.2. Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 15

Таблица 3.3. Функцию, которая вставляет в непустой список L пару новых элементов E1 и E2 перед его последним элементом.

Вариант 16

Таблица 3.1. Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

Вариант 17

Таблица 3.2. Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность.

Вариант 18

Таблица 3.3. Функцию, которая вставляет в непустой список L, элементы которого упорядочены по возрастанию значений одного из полей таблицы, новый элемент E так, чтобы сохранилась упорядоченность

Вариант 19

Таблица 3.1. Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 20

Таблица 3.2. Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 21

Таблица 3.3 Функцию, которая удаляет из непустого списка L первый элемент.

Вариант 22

Таблица 3.1. Функцию, которая удаляет из непустого списка L второй элемент, если такой есть.

Вариант 23

Таблица 3.2. Функцию, которая удаляет из непустого списка L второй

элемент, если такой есть.

Вариант 24

Таблица 3.3. Функцию, которая удаляет из непустого списка L второй элемент, если такой есть.

Вариант 25

Таблица 3.1. Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 26

Таблица 3.2. Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 27

Таблица 3.3 Функцию, которая удаляет из непустого списка L за каждым вхождением элемента E один элемент, если такой есть и он отличен от E.

Вариант 28

Таблица 3.1. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

Вариант 29

Таблица 3.2. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

Вариант 30

Таблица 3.3. Функцию, которая проверяет, есть ли в списке L хотя бы два элемента с равными значениями второго поля.

3.4 Порядок выполнения работы

3.4.1. В ходе самостоятельной подготовки изучить основы работы с динамическими списковыми структурами в языке C/C++.

3.4.2. Выбрать вид линейного списка, подходящий для решения задачи.

3.4.3. Разработать алгоритм решения задачи, разбив его на отдельные процедуры и функции, так, как указано в варианте задания.

3.4.4. Разработать программу на языке C/C++.

3.4.5. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы.

3.4.6. Выполнить отладку программы.

3.4.7. Получить результаты работы программы и исследовать её свойства в различных режимах работы, и сформулировать выводы.

3.5. Содержание отчета

Цель работы, вариант задания, структурные схемы алгоритмов с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

3.6. Контрольные вопросы

3.6.1. Какие операции определены над указателями в языке C/C++?

3.6.2. Что называется списковой структурой данных?

3.6.3. Определите различные виды линейных односвязных списков.

3.6.4. Как описать элемент списковой структуры на языке C/C++?

3.6.5.Напишите на языке C/C++ следующие функции для работы с однонаправленными списками:

- создание списка;
- добавления элемента в список;
- удаления элемента из списка;
- просмотра списка.

4. ЛАБОРАТОРНАЯ РАБОТА №8
ПРОГРАММИРОВАНИЕ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ НА ЯЗЫКЕ С/С++

4.1. Цель работы

Изучение нелинейных структур данных и приобретение навыков разработки и отладки программ, использующих древовидные структуры. Исследование особенностей работы с поисковыми бинарными деревьями на языке С/С++.

4.2. Краткие теоретические сведения

4.2.1. Бинарные деревья и алгоритмы их обработки

В лабораторной работе требуется выполнить операции над бинарными деревьями. В бинарном дереве каждый узел содержит указатель на левое и правое поддерево. Поэтому узел дерева представляют в виде структуры с тремя полями:

```
struct tree
{   int  data;
    struct tree *left;
    sruct tree *riqht;
}
```

Здесь данные, содержащиеся в узле, представлены целыми значениями. При решении конкретной задачи необходимо внести соответствующие изменения в описание поля данных.

Обычно над бинарными деревьями выполняют следующие операции: организацию бинарного дерева, добавления узла к дереву, удаление узла из дерева, просмотр дерева, поиск соответствующего узла в дереве. Алгоритмы этих операций аналогичны рассмотренным ранее алгоритмам обработки деревьев средствами языка Паскаль [5].

Рассмотрим в качестве примера функцию, выполняющую добавление узла в дерево. Процесс добавления узла в дерево является рекурсивным. Если добавленный узел меньше, чем корневой элемент, то добавляем элемент в левое поддерево, иначе добавляем его в правое поддерево. Добавление узла выполняется в этом случае на уровне листьев дерева.

```
struct  tree  *addtree(struct tree *top, int newnode)
{  if (!top) {                    //если находимся на уровне листа,
        top=new tree;             //то выделить память под узел
        if (!top){
              printf("Исчерпана память     \n");
              return NULL;        //выход, если нет памяти
              }
        top->data=newnode;        //запись значения в узел
        top->left=NULL;           //обнуление указателей поддеревьев
        top->riqht=NULL;
```

```
    }
    else                                    //иначе
        if (top->data<newnode)              //сравниваем значение узла с
                                            //добавляемым и добавляем узел
        top->left=addtree(top->left,newnode);     //либо в левое поддерево
        else
        top->riqht=addtree(top->riqht,newnode); //либо в правое поддерево
    return top;                             //возвращаем указатель на корень
    }
```

Если вызов **new** возвращает нулевой указатель, то функция **addtree** завершает свою работу сообщением "Исчерпана память" и возвращает нулевой указатель.

Применяя функцию **addtree** многократно можно разместить в узлах дерева необходимые данные, а затем организовать их обработку в соответствии с вариантом задания.

4.2.2. Пример программы обработки бинарного дерева

Требуется написать функции: создания, добавления листа в бинарное дерево, просмотра бинарного дерева, отображения структуры дерева, а также рекурсивную функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T (корень считать вершиной 1-го уровня).

Ниже приведен текст соответствующей программы с комментариями. С каждым узлом дерева связана структура, состоящая из фамилии и адреса грузополучателя. Дерево упорядочено по фамилиям.

Программа позволяет сохранять дерево в файле и восстанавливать дерево при её повторном запуске. При этом, в отличие от примера в предыдущей лабораторной работе, для передачи указателя по адресу соответствующий параметр (top) функции read_file объявляется как ссылка на указатель. Это упрощает обращение к функции (отпадает необходимость в операции взятия адреса &) и са текст функции (не нужна операция разадресации *).

```
#include <fstream.h>
#include <stdlib.h>
#include <string.h>
#include <iomanip.h>
#include <conio.h>
//---------------------константы----------------------------------------
const int     d_f=20,           //длина строки с ф.и.о.
              d_a=80;           //длина строки с адресом
//--------------------структуры-----------------------------------------
struct gruz_p                   // Описание  записи о грузополучателе
        {char fio[d_f],
        address[d_a];
        };
struct node                 // Описание узла дерева
      { gruz_p data;
```

```
        node* left;
        node* right;
        };
//------------прототипы функций-----------------------------------
node* addtree(node *top,const gruz_p& newnode);   //добавление узла
int menu();                                       //отображение меню
int node_count(node *top,int level,int &n);       //подсчета количества вершин
                                                  //уровня Level
void otobr(node *top, int otstup);                //отображение структуры //дерева
void prosmotr(node *top);                         //просмотр  значений узлов
                                                  //дерева слева направо
int write_file(ofstream &f, node* top);           //запись в файл
int read_file(char* filename, node* &top);        //чтение из файла
gruz_p vvod();                                    //ввод данных
//-----------------------основная функция----------------------------
int main()
{     node *top=0;
     char *filename="gruz.txt";
     ofstream fout;                  //объявление выходного потока
     read_file(filename,top);        //чтение данных из файла
                                     //и создание дерева
     while (1)
     {   switch (menu())             //отображение и ввод пункта меню
        {case 1:                     //пункт 1
            top=addtree(top,vvod()); //ввод и  добавление элемента
            break;
        case 2:                      //пункт 2 – отображение структуры дерева
           otobr(top,1);
           cout<<"Нажмите любую клавишу "<<endl;
           cin.get();
           break;
        case 3:                      //пункт 3 – просмотр значений в узлах
           prosmotr(top);
           cout<<"Нажмите любую клавишу "<<endl;
           cin.get();
           break;
        case 4:                      //пункт 4 – подсчет количества вершин
           int level;                //на уровне level дерева
           int n=0;
           cout<<"Введите значение уровня"<<endl;
           cin>>level;               //ввод значения уровня
           cin.get();
           cout<<"На данном уровне:"<<node_count(top,level,n)
               <<"вершин"<<endl;
           cout<<"Нажмите любую клавишу "<<endl;
```

```
            cin.get();
            break;
        case 5:                         //пункт 5 – запись в файл
            fout.open(filename);        //открытие файла
            if (!fout){
                cout<<"Ошибка открытия файла"<<endl; return 1;}
            write_file(fout,top);               //сохранение данных в файле
            cout<<"Данные сохранены в файле: "<<filename<<endl;
            cout<<"============================="<<endl;
            fout.close();
            cout<<"Нажмите любую клавишу "<<endl;
            cin.get();
            break;
        case 6:                                 //пункт 6 – выход
            return 0;
        default:                                //если неверно введен пункт меню
            cout<<"Вам следует ввести число от 1 до 6"<< endl;
            cin.get();
            break;
        }
    }
}
//----------------------добавление узла в дерево------------------------
node* addtree(node *top,const gruz_p& newnode)
{       if (!top)                               //если находимся на уровне листа,
                {top=new node;                  //то выделить память под узел
                if (!top) {cout<<"Не хватает памяти"<<endl;
                        return NULL;            //выход если память не выделена
                }
                top->data=newnode;              //запись данных в узел
                top->left=NULL;                 //обнуление указателей
                top->right=NULL;
        }
        else                                                    //иначе
                if (strcmp(top->data.fio,newnode.fio)>0)        //сравниваем значение в
                                                                узле с добавляемым и
                top->left=addtree(top->left,newnode);           //добавляем в левое
                                                                //поддерево
                else
                top->right=addtree(top->right,newnode);//или правое поддерево
return top;                             //возвращаем указатель на корень дерева
}
// ----------------отображение структуры дерева-------------------------
//      Функция  отображения структуры дерева.
//     Дерево отображается повернутым на 90 градусов против
```

```
//      часовой стрелки. Узлы дерева, находящиеся на одном
//      уровне, отображаются с  одинаковым отступом от края
//      экрана.}
void otobr(node *top, int otstup)
{       if (top)
                {otstup+=3;                     //отступ от края экрана
                otobr(top->right,otstup);       //обход правого поддерева
                //вывод значения фамилии, соответствующего узла
                cout<<setw(otstup)<<'*'<<top->data.fio<<endl;
                otobr(top->left,otstup);        //обход левого поддерева
        }
}
// ----------------просмотр дерева---------------------------------------
void prosmotr(node *top)                        //просмотр сверху вниз
{       if (top)
                {cout<<top->data.fio<<endl;     //вывод значения корня
                cout<<top->data.address<<endl;
                prosmotr(top->left);            //обход левого поддерева
                prosmotr(top->right);           //обход правого поддерева
        }
}
//-------------------ввод данных-----------------------------------------
gruz_p vvod()
{       gruz_p p;
        cout<<"Введите ф.и.о."<<endl;
        cin.getline(p.fio,d_f);
        cout<<"Введите адрес"<<endl;
        cin.getline(p.address,d_a);
        return p;
}
//---------------отображение и ввод пунктов меню-----------------
int menu()
{       char buf[10];
        int item;
        do
          {clrscr();
           cout<<endl;
           cout<<"==========ДЕРЕВЬЯ==========="<<endl<<endl;
           cout<<"1- Добавить элемент в дерево"<<endl;
           cout<<"2- Отобразить структуру дерева"<<endl;
           cout<<"3- Просмотр дерева"<<endl;
           cout<<"4- Подсчет узлов на заданном уровне"<<endl;
           cout<<"5- Запись данных в файл"<<endl;
           cout<<"6- Выход"<<endl;
           cout<<"============================"<<endl;
```

```
        cout<<"Введите номер пункта меню"<<endl;
        cin>>buf;                                   //ввод номера пункта
        cin.get();
        item=atoi(buf);                             //преобразовать его в целое
        if (!item)                                  //если ошибка
           {  cout<<"Вам следует ввести число от 1 до 6"<<endl;
              cin.get();
           }
      } while (!item);          //повторять пока не будет введено правильно
      return item;              //вернуть номер введенного пункта меню
}
//----------подсчет узлов на заданном уровне дерева------------
int node_count(node *top,int level,int &n)      //n передается по ссылке
{                                               //т.к. модифицируется
      if ((level>=1)&&top)
       {    if (level==1) n++;                  //увеличить счетчик, если
                                                //на уровне level есть вершина.
            n=node_count(top->left,level-1,n);  //подсчет узлов в левом
                                                //поддереве
            n=node_count(top->right,level-1,n);//подсчет узлов в правом
                                                //поддереве
      }
      return n;
}
// ----------------------------чтение файла-----------------------------
int read_file(char* filename, node* &top )
{     ifstream fin(filename,ios::in);   //открытие файла
      if (!fin) {cout<<"Не найден файл"<<filename<<endl; return 1;}
      gruz_p p;
      top = 0;
      while (fin.getline(p.fio,d_f))          //чтение ф.и.о. пока не конец файла
      {     fin.getline(p.address,d_a);       //чтение адреса
            top=addtree(top,p);               //добавить эл-т в дерево
      }
      return 0;
}
// -------------------запись данных в файл-----------------------------
int write_file(ofstream &f, node* top)
{     if (top)
         {  f<<top->data.fio<<endl;           //запись корня под(дерева)
            f<<top->data.address<<endl;
            write_file(f,top->left);          //запись левого поддерева
            write_file(f,top->right);         //запись правого поддерева
         }
      return 0;
```

}

Функции **write_file** передается ссылка на открытый выходной поток, так как она является рекурсивной. Это исключает возможность открытия потока внутри функции по аналогии с функцией **read_file.**

4.3. Варианты заданий

Представить приведенную в предыдущей работе таблицу в виде бинарного дерева. Написать функции создания и обхода дерева, а также одну из функций, приведенных ниже. Значения полей и количество записей в таблице студент выбирает самостоятельно. Программа должна сохранять дерево в файле и создавать его заново при её повторном запуске.

Вариант 1

Таблица 3.1. Функция, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 2

Таблица 3.2. Функцию, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 3

Таблица 3.3. Функцию, которая присваивает параметру E элемент из самого левого листа непустого дерева T.

Вариант 4

Таблица 3.1. Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 5

Таблица 3.2. Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 6

Таблица 3.3. Функцию, которая определяет уровень, на котором находится элемент E в дереве T.

Вариант 7

Таблица 3.1. Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

Вариант 8

Таблица 3.2. Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

Вариант 9

Таблица 3.3. Функцию, которая вычисляет среднее арифметическое всех элементов непустого дерева T (по одному из полей таблицы, которое имеет числовое значение)

Вариант 10

Таблица 3.1. Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 11

Таблица 3.2. Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 12

Таблица 3.3. Функцию, которая заменяет в дереве T все элементы меньшие, чем некоторое положительное число A, на это число (по одному из полей таблицы, которое имеет числовое значение).

Вариант 13

Таблица 3.1. Функцию, которая печатает элементы всех листьев дерева.

Вариант 14

Таблица 3.2. Функцию, которая печатает элементы всех листьев дерева.

Вариант 15

Таблица 3.3. Функцию, которая печатает элементы всех листьев дерева.

Вариант 16

Таблица 3.1. Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 17

Таблица 3.2. Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 18

Таблица 3.3. Функцию, которая находит в непустом дереве T длину (число ветвей) пути от корня до вершины с элементом E.

Вариант 19

Таблица 3.1. Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 20

Таблица 3.2. Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 21

Таблица 3.3. Функцию, которая подсчитывает число вершин на n-ом уровне непустого дерева T.

Вариант 22

Таблица 3.1. Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 23

Таблица 3.2. Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 24

Таблица 3.3. Написать рекурсивную функцию, которая определяет, входит ли элемент в дерево T.

Вариант 25

Таблица 3.1. Написать рекурсивную функцию, которая определяет число вхождений элемента E в дерево T.

Вариант 26

Таблица 3.2. Написать рекурсивную функцию, которая определяет число вхождений элемента E в дерево T.

Вариант 27

Таблица 3.3. Написать рекурсивную функцию, которая определяет число вхождений элемента E в дерево T.

Вариант 28

Таблица 3.1. Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

Вариант 29

Таблица 3.2. Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

Вариант 30

Таблица 3.3. Написать рекурсивную функцию, которая вычисляет сумму элементов (по одному из полей таблицы) непустого дерева T.

4.4. Порядок выполнения работы

4.4.1. Изучить в ходе самостоятельной подготовки функции обработки бинарных деревьев на языке C/C++.

4.4.2. Описать элемент бинарного дерева в соответствии с вариантом задачи.

4.4.3. Разработать алгоритм решения задачи, разбив его на отдельные функции, так как указано в варианте задания.

4.4.4. Разработать программу на языке C/C++.

4.4.5. Разработать тестовые примеры, которые предусматривают проверку корректности работы программы в разных режимах.

4.4.6. Выполнить отладку программы

4.4.7. Получить результаты работы программы и исследовать её свойства в различных режимах работы, сформулировать выводы.

4.5. Содержание отчета

Цель работы, вариант задания, структурные схемы алгоритмов с описанием, текст программы с комментариями, тестовые примеры, результаты вычислений, выводы.

4.6. Контрольные вопросы

4.6.1. Что понимают под нелинейной структурой данных?

4.6.2. Что называется бинарным деревом данных?

4.6.3. Как построить упорядоченное бинарное дерево?

4.6.4. Отчего зависит эффективность поиска в бинарном дереве?

4.6.5. Что понимают под симметричным и выровненным деревом?

4.6.6. Как формируется выровненное дерево?

4.6.7. Напишите на языке C/C++ функции для работы с бинарными деревьями:

- создания упорядоченного бинарного дерева;
- добавления элементов в дерево;
- удаления листа дерева;
- обхода дерева.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Методические указания к лабораторным работам по дисциплине “Алгоритмизация и программирование” для студентов дневной и заочной форм обучения направления 09.03.02 – “Информационные системы и технологии”, часть 1/ Сост. В.Н. Бондарев, Т.И. Сметанина, А.К. Забаштанский.— Севастополь: Изд-во СевГУ, 2016. — 76с.
2. Белецкий Я. Энциклопедия языка С: пер. с польск. / Я. Белецкий. — М.:Мир, 1992. — 687 с.
3. Вирт Н. Алгоритмы и структуры данных: Пер. с англ. / Н. Вирт. — М.: Мир, 1989. — 360с.
4. Глушаков С.В. Язык программирования С++ / С.В. Глушаков, А.В. Коваль, С.В. Смирнов. — Харьков: Фолио, 2002. — 500 с.
5. Керниган Б., Ритчи Д. Язык программирования Си: пер. с англ. / Под ред. и с предисл. В.С. Штаркмана.—2-е изд., перераб. и доп.— М. ; СПб. ; К. : Вильямс, 2006.—272с.
6. Павловская Т.А. С/ С++. Программирование на языке высокого уровня : учеб. для студ. вузов, обуч. по напр. «Информатика и вычислительная техника» / Т. А. Павловская.— СПб.: Питер, 2009. – 461 с.
7. Павловская Т.А. С/С++. Структурное программирование: практикум / Т.А. Павловская, Ю.А. Щупак. — СПб.: Питер, 2004.—239 с.

Заказ № _____ от «_____» _____2016г. Тираж _____экз.
Изд-во СевГУ