Министерство науки и высшего образования 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 формы обучения направлений
09.03.02 – «Информационные системы и технологии»
и 09.03.03 – «Прикладная информатика»

2 часть

Севастополь
2022

УДК 004.42 (075.8)

Методические указания к лабораторным работам по дисциплине «Алгоритмизация и программирование» для студентов дневной и заочной формы обучения направлений 09.03.02 — «Информационные системы и технологии» и 09.03.03 — «Прикладная информатика», часть 2 / Сост. В. Н. Бондарев, Т. И. Сметанина, А. Ю. Абрамович. — Севастополь: Изд-во СевГУ, 2022. — 58с.

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 C/C++.

Методические указания составлены в соответствии с требованиями программы дисциплины «Алгоритмизация и программирование» для студентов направлений 09.03.02 — «Информационные системы и технологии» и 09.03.03 — «Прикладная информатика « и утверждены на заседании кафедры «Информационные системы» протокол № ____ от

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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цели и задачи лабораторных работ

Основная цель выполнения настоящих лабораторных работ – получение практических навыков создания и отладки программ на языке C/C++ с использованием сложных типов 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 Dev-C++ (или Eclipse CDT), навыки разработки программ, использующих текстовые и типизированные файлы, структуры, динамические списковые и древовидные структуры данных.

Выбор вариантов и график выполнения лабораторных работ

Варианты заданий приведены в каждой лабораторной работе и уточняются преподавателем.

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

– проработать по конспекту и рекомендованной литературе, приведенной в конце настоящих методических указаний, основные теоретические положения лабораторной работы и подготовить ответы на контрольные вопросы;

– разработать алгоритм решения задачи и составить схему алгоритма;

– описать схему алгоритма;

– написать программу на языке C/C++ и описать ее;

– оформить результаты первого этапа в виде заготовки отчета по лабораторной работе.

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

– отладить программу;

– разработать и выполнить тестовые примеры.

Выполнение лабораторной работы завершается защитой отчета. Студенты должны выполнять и защищать работы **строго по графику**.

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 постановку задачи; результаты выполнения программы; выводы по работе; приложения. Содержание отчета указано в методических указаниях к каждой лабораторной работе.

1 ЛАБОРАТОРНАЯ РАБОТА №1 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 символом с кодом ноль (`'\0'`). Этот завершающий (терминирующий) ноль в конце строки также называют терминирующим нулем или терминатором – признаком конца строки. При объявлении строкового массива и работе с ним необходимо принимать во внимание наличие терминатора (`'\0'`) в конце строки, отводя под строку на один байт больше максимального количества символов. Удобно задавать длину строки именованной константой:

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

В такой 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 освобождения памяти соответственно:

```
free(s1);
delete []s2; // всегда [] при освобождении памяти массива
```

Для ввода-вывода строк можно использовать как *объекты* `cin` и `cout` языка C++, так и функции языка С.

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

```
const int n = 80;
```

```
int main() {
 char s[n];
 scanf("%s",s); // считывание до пробела, но не более 79 симво-
лов
 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, используется спецификация `%c`:

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

В этом случае количество считываемых символов может быть только константой (и равно 20), и короткие строки придется при вводе дополнять пробелами до требуемой длины строки.

```
const int n = 80;
int main() {
     char s[n];
     scanf("%10c",s); // считывание 10 символов
     printf("%s",s);
}
```

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.

```
const int m = 11;
int main() {
     char t[] = "Язык Си";
     puts(t);
     char *s = new char[m];
     puts("введите не более 10 символов");
     gets(s); //чтение не более 10 символов
     puts(s);
     delete []s;
}
```

Большинство функций работы со строками описаны в заголовочном 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`:

```
const int m = 10;
     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. Длина строки это количество ненулевых символов в строке, считая от начала строки, до первого ноля.

Таблица 1.1 — Функции для обработки строк

char* strcpy(s,ct)	копирует строку ***ct*** в строку ***s***; возвращает ***s***
char* strcat(s,ct)	соединяет строку ***s*** и ***ct***
char* strcmp(cs,ct)	сравнивает строки ***cs*** и ***ct***
char* strchr(cs,c)	возвращает указатель на первое вхождение символа ***c*** в строку ***cs***
char* strrchr(cs,c)	возвращает указатель на последнее вхождение символа ***c*** в строку ***cs***
size_t strspn(cs,ct)	возвращает число символов в начале строки ***cs***, ни один из которых не входит в строку ***ct***
size_t strcspn(cs,ct)	возвращает число символов в начале строки ***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 строки ***ct*** в строку ***cs***
size_t strlen(cs)	возвращает длину строки ***cs***
char* strtok(s,ct)	ищет в строке ***s*** лексему, ограниченную литерами из строки ***ct***

Рассмотрим ввод и вывод строк с использованием объектов `cin` и `cout` языка C++:

```
#include <iostream>
using namespace std;
const int n = 80;
int main() {
     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>
using namespace std;
const int n = 80;
int main() {
     char s[n];
     cin.getline(s,n); // 1 способ
     cout<<s<<endl;
     cin.get(s,n); // 2 способ
     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()`.

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 значений 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 1.2.

Таблица 1.2 — 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.

Чтобы указать, что данный файл открывается или создается как текстовый, добавьте символ `t` в строку режима (например, `"rt"`,`"w+t"` и т.п.).

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:\\1\\old.txt","rt"))== NULL) {
      fprintf(stderr," Не могу открыть входной файл \n");
      return 1;
 }
 if ((out = fopen("c:\\1\\new.txt","wt"))== NULL) {
      fprintf(stderr,"Не могу открыть выходной файл \n");
      return 1;
 }
 i = 0;
 while (fscanf(in,"%d",&x)!=EOF) { // чтение чисел
      i++;
      fprintf(out,"%d\t%d\n",i,x); // печать чисел
 }
 fclose(in); //закрытие файлов
 fclose(out);
 return 0;
}
```

Программа копирует целые числа из входного файла `old.txt` в выходной файл `new.txt`. 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.

Рассмотрим еще один пример файлового ввода и вывода с помощью функций `fscanf()` и `fprintf()`:

```
#include <stdio.h>
int main() {
 char x;
 FILE *in,*out; // описание указателей на файлы
 in = fopen("c:\\1\\old.txt","rt");
 if (in == NULL) {
      fprintf(stderr," Не могу открыть входной файл \n");
```

```
        return 1;
    }
    out = fopen("c:\\1\\new.txt","wt");
    if (out== NULL) {
        fprintf(stderr,"Не могу открыть выходной файл \n");
        return 1;
    }
    while (1) { // вечный цикл
        fscanf(in,"%c",&x);
        if (feof(in)) break; // выход из цикла
        fprintf(out,"%c",x);
    }
    fclose(in);
    fclose(out);
    return 0;
}
```

Программа копирует символы из входного файла `old.txt` в выходной файл `new.txt` в цикле пока не встретит метку конца файла.

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`>. 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.

1.2.4 Пример программы обработки текстового файла

Написать программу, которая построчно копирует содержимое файла `text.txt` в файл `out.txt`.

```
#include <fstream>
#include <iostream>
using namespace std;
const int len = 80;
int main() {
 ifstream fin("text.txt", ios::in);
 if (!fin) {
      cout<<"ошибка открытия";
      return 1;
 }
 ofstream fout("out.txt",ios::out | ios::app);
 if (!fout) {
      cout<<"ошибка открытия";
      return 1;
 }
```

```
    char s[len];
    while (1) {
         fin.getline(s,len);
         if (fin.eof()) break;
         fout<<s<<endl;
    }
    fin.close();
    fout.close();
}
```

В заключение отметим, что файл с тестовым примером должен содержать нескольких строк различной длины.

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

Написать две программы: в первой ввод-вывод осуществлять с помощью средств С, во второй ввод-вывод осуществлять с помощью классов С++.

Вариант 1

Написать программу, которая считывает текст из файла и определяет количество символов *X* в каждой строке.

Вариант 2

Написать программу, которая считывает текст из файла и заменяет все восклицательные знаки многоточием.

Вариант 3

Написать программу, которая считывает текст из файла и определяет количество арифметических операций (+, -, *, /) в каждой строке.

Вариант 4

Написать программу, которая считывает текст из файла и определяет номер самой длинной строки.

Вариант 5

Написать программу, которая считывает текст из файла и определяет количество строк, содержащих хотя бы одну из арифметических операций (+, -, *, /).

Вариант 6

Написать программу, которая считывает текст из файла и определяет номер самой короткой строки.

Вариант 7

Написать программу, которая считывает текст из файла и определяет количество строк, не содержащих символы препинания (. : ; ! ?,).

Вариант 8

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся и заканчивающихся одним и тем же символом.

Вариант 9

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся с пробела.

Вариант 10

Написать программу, которая считывает текст из файла и заменяет символ A на символ B.

Вариант 11

Написать программу, которая считывает текст из файла и удаляет все символы A.

Вариант 12

Написать программу, которая считывает текст из файла и удаляет все фразы в фигурных скобках {}. Количество открывающихся скобок равно количеству закрывающихся скобок.

Вариант 13

Написать программу, которая считывает текст из файла и заменяет любую последовательность точек одним символом *. Например исходный текст: "9.00…1……27.3.4…"; результат: "9*00*1*27*3*4*".

Вариант 14

Написать программу, которая считывает текст из файла и заменяет цифры их названиями (например: 1->один, 2->два т.д.).

Вариант 15

Написать программу, которая считывает текст из файла и заменяет символ «+» словом «плюс».

Вариант 16

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся с цифры.

Вариант 17

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся и заканчивающихся цифрой.

Вариант 18

Написать программу, которая считывает текст из файла и определяет: верно ли, что в файле количество открытых скобок равно количеству закрытых скобок.

Вариант 19

Написать программу, которая считывает текст из файла и определяет количество строк, содержащих два пробела подряд.

Вариант 20

Написать программу, которая считывает текст из файла и заменяет строчные гласные буквы на заглавные.

Вариант 21

Написать программу, которая считывает текст из файла и заменяет заглавные гласные буквы на строчные.

Вариант 22

Написать программу, которая считывает текст из файла и удаляет все, кроме фраз, заключенных в фигурные скобки {}. Количество открывающихся скобок равно количеству закрывающихся скобок.

Вариант 23

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся с гласной буквы.

Вариант 24

Написать программу, которая считывает текст из файла и определяет количество строк, начинающихся и заканчивающихся пробелом.

Вариант 25

Написать программу, которая считывает текст из файла и определяет количество слов в каждой строке.

Вариант 26

Написать программу, которая считывает текст из файла и определяет для каждой строки файла количество символов ‘*’.

Вариант 27

Написать программу, которая считывает текст из файла и определяет количество запятых в каждой строке.

Вариант 28

Написать программу, которая считывает текст из файла и определяет для каждой строки файла номер первой запятой.

Вариант 29

Написать программу, которая считывает текст из файла и определяет для каждой строки файла номер последней запятой.

Вариант 30

Написать программу, которая считывает текст из файла и определяет количество строк, состоящих из одинаковых символов.

1.4 Порядок выполнения работы

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

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 две программы: одну на языке C, другую на языке C++.

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, структурные схемы алгоритма решения задачи с описанием, тексты программ с комментариями, тестовые примеры, выводы.

1.6 Контрольные вопросы

1.6.1 Приведите примеры описания и инициализации строк в языке C.

1.6.2 Как хранится строка языка C 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 C?

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 Как открыть файловый поток в режиме чтения в языке C++?

1.6.14 Как открыть файловый поток в режиме записи в языке C++?

2 ЛАБОРАТОРНАЯ РАБОТА №2 ПРОГРАММИРОВАНИЕ ОПЕРАЦИЙ НАД СТРУКТУРАМИ И БИНАРНЫМИ ФАЙЛАМИ

2.1 Цель работы

Исследование особенностей обработки бинарных файлов, хранящих структурные типы данных.

2.2 Краткие теоретические сведения

2.2.1 Бинарные файлы

Содержимое файлов представляет собой набор байтов (чисел от 0 до 255). Для интерпретации этих данных как полезной для человека информации нужно знать правила кодирования – декодирования. Например, для представления текста используемые символы были пронумерованы и сведены в таблицы.

Бинарные (двоичные) файлы содержат данные в представлении, в котором компьютеру удобнее считывать, обрабатывать и сохранять данные. Обычно в бинарном файле данные находятся в том же виде, в котором они лежат в памяти компьютера. И поэтому скорость обработки данных в бинарном виде гораздо выше, так как отсутствуют промежуточные операции преобразования.

Стандартная библиотека C/C++ для работы с бинарными файлами предоставляет специальные функции.

Рассмотрим основные из них: `fread` и `fwrite`. Эти функции позволяют читать и записывать блоки данных любого типа.

```
size_t fread ( void * считываемое_данное,
               size_t размер_элемента_байты,
               size_t количество_элементов,
               FILE * указатель_на_файл);
```

Функция считывает из файла указанное число элементов (блоков) заданного размера. Размер блока задается в байтах. Функция возвращает число прочитанных элементов. Если число прочитанных элементов не равно заданному, то при чтении возникла ошибка или встретился конец файла.

```
size_t fwrite(const void *буфер,
              size_t размер_элемента_байты,
              size_t количество_элементов,
              FILE * указатель_на _файл);
```

Функция `fwrite`() возвращает количество записанных элементов. Если ошибка не произошла, то возвращаемый результат будет равен значению `количество_элементов`.

Одним из самых полезных применений функций `fread`() и `fwrite`() является чтение и запись данных пользовательских типов, особенно структур. Например:

```
#include <stdio.h>
struct some_struct { /* … */ };
int main () {
//   . . . . . .
 FILE * fp;
 char filename[] = "some_file";
 struct some_struct buff[64]; // массив структур из 64-х
     fp =fopen(filename, "wb");
     if( fp == NULL ) {
          printf("Error opening file %s\n",filename);
     } else {
          fwrite(buff, sizeof(struct some_struct), 64, fp);
          fclose(fp);
     }
//   . . . . . .
     fp =fopen(filename, "rb");
     if( fp == NULL ) {
          printf("Error opening file %s\n",filename);
     } else {
          fread(buff, sizeof(struct some_struct), 64, fp);
          fclose(fp);
     }
     return 0;
}
```

2.2.2 Позиционирование в файле

Каждый открытый файл имеет так называемый указатель на текущую позицию в файле. Все операции над файлом (чтение и запись) выполняются с этой позиции. При каждом выполнении функции чтения или записи, указатель смещается на количество прочитанных или записанных байт, то есть устанавливается сразу за прочитанным или записанным блоком данных в файле. Это так называемый последовательный доступ к данным. Последовательный доступ удобен, когда необходимо последовательно работать с данными в файле. Но иногда необходимо читать или писать данные в произвольном порядке. Это достигается путем установки указателя чтения/записи на некоторую заданную позицию в файле функцией `fseek()`.

```
int fseek(FILE *stream, long offset, int whence);
```

Параметр `offset` задает количество байт, на которое необходимо сместить указатель от точки отсчета, указанной `whence` (таблица 2.1).

Таблица 2.1 – Возможные значения параметра `whence`

Символьное имя	Значение	Точка отсчета
SEEK_SET	0	Смещение от начала файла
SEEK_CUR	1	Смещение от текущей позиции
SEEK_END	2	Смещение от конца файла

Величина смещения может быть как положительной, так и отрицательной, но нельзя сместиться за пределы начала файла (назад от начала файла).

Для перемещения указателя текущей позиции в файле на начало файла, наряду с функцией `fseek`(), может также быть использована функция `rewind`():

```
void rewind(FILE * указатель_на _файл);
```

Но кроме перемещения указателя позиции чтения/записи на начало, `rewind`() всегда выполняет сброс состояния ошибки в потоке, если до этого при работе с файлом возникла ошибка.

Узнать текущее значение указателя позиции чтения/записи можно с помощью функции `ftell()`:

```
long int ftell(FILE * указатель_на _файл);
```

Она возвращает позицию в байтах. При возникновении ошибки функция ftell() возвращает значение -1.

Совместное использование с другими функциями предоставляет дополнительные возможности – например, чтобы узнать размер файла в байтах, нужно переставить указатель на конец файла, а затем узнать его текущее значение.

2.2.3 Пример программы обработки структур

Имеется структура, описывающая рациональную дробь. В структуре два поля: числитель и знаменатель. Требуется написать программу, которая

– вводит дроби с клавиатуры в бинарный файл;

– упорядочивает его по возрастанию;

– отображает содержимое бинарного файла на экран в читаемом человеком виде.

Исходные данные и результаты.

Исходные данные:

Значение одной дроби представим в виде структуры, состоящей из двух полей: числителя и знаменателя. Оба поля будут целочисленными. Поскольку количество дробей не оговорено, то будем хранить все сведения в бинарном файле, а не массиве.

Результаты:

В результате работы программы требуется вывести на экран отсортированные по возрастанию дроби.

Алгоритм решения задачи.

1. Ввести с клавиатуры числитель и знаменатель дроби в бинарный файл. При этом будем прекращать ввод, если пользователь введет символ ‘0’.

2. Выполнить сортировку файла по возрастанию, используя возможность прямого доступа к элементам бинарного файла. В качестве метода сортировки будем использовать метод прямого обмена (пузырьковую сортировку);

3. Обеспечить вывод отсортированной последовательности дробей:

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

```
#include <stdio.h>
#include <string.h>
#include <conio.h>
//------------------константы и структуры-------------------------
struct drob {
     int ch,zn;
};
const int size_drob=sizeof(drob);   //размер структуры
//---------------------прототипы функций-------------------------
int create_file(FILE *f);      //запись в файл
int sort_file(FILE *f);        //сортировка файла
int print_file(FILE *f);       //вывод файла

//---------------------основная функция--------------------------
int main() {
     char c;
     //Открытие существующего файла для чтения и записи в конец
     FILE *f=fopen("data.dat", "rb+");
     if (!f) {
          //Создание нового файла для обновления
          f=fopen("data.dat","wb+");
          if(!f) {
               puts("Не могу открыть (создать) файл\n");
               return 1;
          }
     }
     //вывод меню и запуск соответствующих функций
     while (1) {
          puts("1- Запись в файл");
          puts("2- Сортировка файла");
          puts("3- Вывод файла");
          puts("5- Выход");
          puts("_____________________________");
          puts("Введите номер пункта меню\n");
          c=getch();
          switch (c) {
               case '1':
                    create_file(f);
                    break;
               case '2':
                    sort_file(f);
                    break;
               case '3':
                    print_file(f);
                    break;
               case '5':
                    return 0;
          }
     }
}
//----------------------запись в файл-----------------------------

int create_file(FILE *f) {
```

```
     drob d;
     fseek(f,0,SEEK_END);          //указатель в конец файла
     while (1) {
          puts("Введите числитель (0-exit)");
          scanf("%d",&d.ch);
          if (d.ch==0)
               return 1;
          puts("Введите знаменатель");
          scanf("%i",&d.zn);
          fwrite(&d,size_drob,1,f);
     }
}
//----------------------вывод файла-----------------------------
int print_file(FILE *f) {
     drob d;
     int n;
     rewind(f);                    //указатель в начало файла
     puts("_____________________________________");
     puts("| числитель   |  знаменатель |");
     do {
          n=fread(&d,size_drob,1,f);//чтение структуры из файла
          if (n<1) break;          //если n<1, то конец файла
          printf("| %-13d | %-14d|\n",d.ch, d.zn);
     } while (1);
     puts("_____________________________________");
     puts("Нажмите любую клавишу");
     getch();
     return 0;
}
int sort_file(FILE *f) {
     long i,j;
     drob d1,d2;

     fseek(f,0,SEEK_END);        //указатель в конец
     long len=ftell(f)/size_drob;//определяем количество записей
     rewind(f);                 //указатель в начало
//пузырьковая сортировка
for(i=len-1; i>=1; i--)
     for (j=0; j<=i-1; j++) {
      fseek(f,j*size_drob,SEEK_SET); //указатель на j-ую запись
      fread(&d1,size_drob,1,f);      //читаем запись j в d1
      fread(&d2,size_drob,1,f);      //читаем след. запись в d2
          if (d1.ch*d2.zn>d2.ch*d1.zn) {
               //указатель на 2 поз. назад
          fseek(f,(-2)*size_drob,SEEK_CUR);
                    //обмен значений
          fwrite(&d2,size_drob,1,f);     //сначала записываем d2
          fwrite(&d1,size_drob,1,f);     // затем записываем d1
          }
     }
     puts("Сортировка успешна завершена");
     getch();
     return 0;}
```

Подробные комментарии в тексте программы позволят быстро разобраться в её работе. В тексте основной программы осуществляется либо открытие файла (режим «r+b»), либо создание файла (режим «w+b») для обновления, если файл «data.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.

2.3 Варианты заданий

Вариант 1

Описать структуру с именем STUDENT, содержащую следующие поля:

– порядковый номер;

– фамилия и инициалы;

– номер группы;

– успеваемость (не менее трех дисциплин).

Написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий и номеров групп для всех студентов, чей средний балл больше 4.0, а если таких студентов нет, то вывести соответствующее сообщение;

– отсортировать записи по возрастанию номера группы.

Вариант 2

Для структуры, указанной в варианте 1, написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих оценки 4 и 5, а если таких студентов нет, то вывести соответствующее сообщение;

– отсортировать записи по возрастанию среднего балла.

Вариант 3

Для структуры, указанной в варианте 1, написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из структур типа STUDENT;

– чтение данных из этого файла;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий и номеров групп для всех студентов, имеющих хотя бы одну оценку 2, если таких студентов нет, то вывести соответствующее сообщение;

– отсортировать записи по полю фамилия по алфавиту.

Вариант 4

Описать структуру с именем AEROFLOT, содержащую следующие поля:

– порядковый номер;

– название пункта отправления рейса;

– 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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры, а если таких рейсов нет, то выдать на дисплей соответствующее сообщение;

– отсортировать записи по возрастанию номера рейса.

Вариант 5

Для структуры, указанной в варианте 4, написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из элементов типа AEROFLOT;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран пунктов назначения и номеров рейсов, обслуживаемых самолетом, тип которого введен с клавиатуры, а если таких рейсов нет, выдать на дисплей соответствующее сообщение;

– отсортировать записи в алфавитном порядке по названиям пунктов назначения.

Вариант 6

Описать структуру с именем WORKER, содержащую следующие поля:

– порядковый номер;

– 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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры, если таких работников нет, то вывести на дисплей соответствующее сообщение;

– отсортировать записи по названию занимаемой должности.

Вариант 7

Описать структуру с именем TRAIN, содержащую следующие поля:

– порядковый номер;

– название пункта отправления;

– название пункта назначения;

– номер поезда;

– время отправления (часы и минуты).

Написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о поездах, отправляющихся после введенного с клавиатуры времени, если таких поездов нет, выдать на дисплей соответствующее сообщение;

– отсортировать записи по алфавиту по названиям пунктов назначения.

Вариант 8

Для структуры, указанной в варианте 7, написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о поездах, направляющихся в пункт, название которого введено с клавиатуры, а если таких поездов нет, то выдать на дисплей соответствующее сообщение;

– отсортировать записи по времени отправления поезда.

Вариант 9

Для структуры, указанной в варианте 7, написать программу, выполняющую следующие действия с помощью функций:

– ввод с клавиатуры данных в файл, состоящий из элементов типа TRAIN;

– чтение данных из этого файла;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о поезде, номер которого введен с клавиатуры, а если таких поездов нет, то выдать на дисплей соответствующее сообщение;

– отсортировать записи по номерам поездов.

Вариант 10

Описать структуру с именем MARSH, содержащую следующие поля:

– порядковый номер;

– 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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о маршруте, номер которого введен с клавиатуры, а если таких маршрутов нет, то выдать на дисплей соответствующее сообщение;

– отсортировать записи по номерам маршрутов.

Вариант 11

Для структуры, указанной в варианте 10, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– 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:

– порядковый номер;

– 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о человеке, номер телефона которого введен с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 13

Для структуры, указанной в варианте 12, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 14

Для структуры, указанной в варианте 12, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 15

Описать структуру с именем ZNAK, содержащую следующие поля:

– порядковый номер;

– 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о человеке, чья фамилия введена с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 16

Для структуры, указанной в варианте 15, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о людях, родившихся под знаком, название которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 17

Для структуры, указанной в варианте 15, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о людях, родившихся в месяц, значение которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 18

Описать структуру с именем PRICE, содержащую следующие поля:

– порядковый номер;

– название товара;

– название магазина, в котором продается товар;

– стоимость товара в руб.

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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о товаре, название которого введено с клавиатуры, а если таких товаров нет, выдать на дисплей соответствующее сообщение.

Вариант 19

Для структуры, указанной в варианте 18, 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о товарах, продающихся в магазине, название которого введено с клавиатуры, а если такого магазина нет, выдать на дисплей соответствующее сообщение.

Вариант 20

Описать структуру с именем ORDER, содержащую следующие поля:

– порядковый номер;

– расчетный счет плательщика;

– расчетный счет получателя;

– перечисляемая сумма в руб.

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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на экран информации о сумме, снятой с расчетного счета плательщика, введенного с клавиатуры, а если такого расчетного счета нет, выдать на дисплей соответствующее сообщение.

Вариант 21

Описать структуру с именем STUDENT, содержащую следующие поля:

– порядковый номер;

– 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– 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;

– корректировку данных в файле по номеру записи;

– вывод на дисплей анкетных данных студентов, получивших одну оценку 3, а 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 и вывод их на экран;

– вывод на дисплей анкетных данных студентов, получивших все двойки, а 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 и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей анкетных данных студентов, получивших все пятерки, а 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;

– корректировку данных в файле по номеру записи;

– 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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий студентов, которые начинаются с литеры ‘А’, и их оценок, а если таких студентов нет, вывести соответствующее сообщение;

– отсортировать записи по номеру по убыванию.

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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’, и год их рождения, а если таких студентов нет, вывести соответствующее сообщение;

– отсортировать записи по номеру по возрастанию.

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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий студентов, которые начинаются с литеры ‘Б’ или ‘Г’, и год их поступления, а если таких студентов нет, вывести соответствующее сообщение;

– отсортировать записи по фамилиям по убыванию.

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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий студентов, имеющих средний балл выше среднего балла группы, а если таких студентов нет, вывести соответствующее сообщение;

– отсортировать записи по фамилиям по возрастанию.

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;

– чтение данных из этого файла и вывод их на экран;

– корректировку данных в файле по номеру записи;

– вывод на дисплей фамилий и года рождения студентов, не получивших ни одной тройки, а если таких студентов нет, вывести соответствующее сообщение

– отсортировать записи по номеру.

2.4 Порядок выполнения работы

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

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

2.4.3 Разработать программу на языке C/C++.

2.4.4 Разработать тестовые примеры, следуя указаниям раздела 3.

2.4.5 Выполнить отладку программы.

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

2.5 Содержание отчета

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

2.6 Контрольные вопросы

2.6.1 Что называют структурой в языке C/C++?

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

2.6.3 Как выполнить чтение и запись структурных переменных в бинарные файлы?

2.6.4 Как обнаружить конец файла при использовании функции `fread`?

2.6.5. Как определить длину файла с помощью функций `seek` и `ftell`?

3 ЛАБОРАТОРНАЯ РАБОТА №3 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.

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 my_list {
   int inf;                    // инф. поле
   struct my_list* next;       //поле-указатель на след. элемент
   };
```

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:

```
my_list *beg=NULL;         //указатель на начало списка
my_list *temp;
int x;
while (1){
   scanf("%d", &x);            //ввод x
   if (x==9999) break;         //условие выхода из цикла
                               //выделение памяти под элемент
 temp=(struct my_list *)malloc(sizeof(struct my_list));
 temp->inf=x;                  //присваивание значений  полю inf
 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 **malloc** динамически выделяется память под один элемент списка, и адрес этой области памяти запоминается в указателе **temp**. В поле **inf** 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).

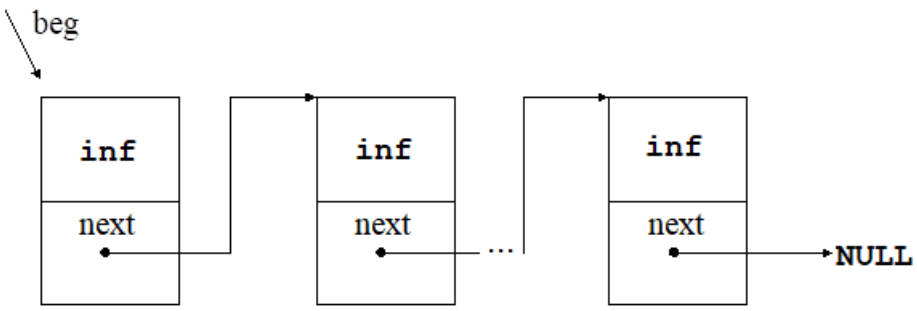

Рисунок 3.1 – Линейный список

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
free (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->inf);    //вывод значения инф. поля
temp=temp->next;         //перемещение temp на следующий элемент
}
```

3.2.2 Пример программы обработки списка

Рассмотрим программу, выполняющую организацию списка, добавление элемента в список, удаление элемента из списка и просмотр списка.

В примере элементами списка являются структуры данных о детях. Информационное поле содержит порядковый номер ребенка, его имя и возраст.

Интерфейс программы организуем в виде меню:

1) добавление элемента в список;
2) просмотр – вывод на экран элементов;
3) удаление первого элемента из списка;
4) выход.

Для работы со списком введем два указателя: **beg** – указатель на первый элемент списка и **end** – указатель на последний элемент списка.

Работу со списком разобьем на шесть функций. Функция **add_first** выполняет добавление элемента в начало списка (перед первым элементом); первым параметром ей передается указатель на начало списка и в качестве результата его же функция и возвращает, вторым параметром передаются данные для нового элемента (для заполнения информационного поля). Данные считывает и в качестве результата возвращает функция **read_data**. Функция **add_last** добавляет новый элемент в конец списка (после последнего элемента).

Функция **create_my_list** создает список, данные считываются с клавиатуры, если список пустой, то формируется список из одного элемента, иначе новые элементы добавляются в конец списка; параметром передается указатель на первый элемент и его же функция возвращает.

Просмотр списка осуществляет функция **print_my_list**, параметром ей передается указатель на начало списка, функция ничего не возвращает.

Удаление элементов выполняется с головы (начала) очереди функцией **del_first**. Для этого функции передается указатель **beg**. После удаления элемента функция возвращает измененное значение **beg**. Работа программы реализована в виде меню.

```
#include <stdio.h>
#include <windows.h>
//--------------- описание типа инф. поля -----------------
struct data {
 int num;
 char name[30];
 int age;
};
//--------------- описание списка ------------------------
struct my_list {
 struct data inf;
 struct my_list *next;
};
//-----------------прототипы функций-----------------------
struct my_list * del_first(struct my_list *beg);
struct data read_data();
struct my_list * add_first(struct my_list * beg, struct data z);
struct my_list * add_last(struct my_list * beg, struct data z);
struct my_list * create_my_list(struct my_list * beg);
void print_my_list(my_list *beg);
//----------------основная функция-----------------------------
int main() {
 struct my_list *beg,*end,*t;
 beg=end=NULL;
 int answer;
 struct data z;
 while (1) {
      printf("\n1. добавление в начало\n"
             "2. добавление после последнего\n"
             "3. создание списка\n"
             "4. просмотр списка\n"
             "5. удаление первого\n"
             "0. выход\n");
      printf("\n введите - >"),scanf("%d",&answer);
      switch (answer) {
           case 0:
                return 0;
           case 1: {
                z=read_data();
                beg=add_first(beg,z);
                break;
           }
           case 2:
                beg=add_last(beg,read_data());
                break;
           case 3:
                beg=create_my_list(beg);
```

```
                    break;
               case 4:
                    print_my_list(beg);
                    break;
               case 5:
                    beg=del_first(beg);
                    break;
               default:
                    printf("Команада не распознана\n");
          }
     }
}
//------------ удаление первого элемента ----------------
struct my_list * del_first(struct my_list *beg) {
 if (beg){
 my_list *t=beg;
 beg=t->next;
 free(t);
 }
 else printf("список пуст");
 return beg;
}
//--------------- чтение данных --------------------------
struct data read_data() {
 struct data z;
 printf(" введите номер, имя и возраст \n");
 scanf("%d %s %d", &z.num,&z.name,&z.age);
 return z;
}
//----------добавление в начало списка -----------------------
struct my_list * add_first(struct my_list * beg, struct data z) {
 my_list *t;
 t=(struct my_list *)malloc(sizeof(struct my_list));
 t->inf=z;
 t->next=beg;
 beg=t;
 return beg;
}
//----------- добавление в конец списка ------------------------
struct my_list * add_last(struct my_list * beg, struct data z) {
 my_list *t, *end;
 t=(struct my_list *)malloc(sizeof(struct my_list));
 t->inf=z;
 t->next=NULL;
 if (beg==NULL)
      beg=t;
 else {
      end=beg;
      for(; end->next!=NULL; end=end->next);
      end->next=t;
 }
 return beg;
}
```

```
//------------- создание списка -------------------------------
struct my_list * create_my_list(struct my_list * beg) {
 int fl=0;
 struct data z;
 do {
      if (beg==NULL) {    //если список пуст, то создать список
           z=read_data();
           beg=add_first(beg,z);
      } else {   // иначе, добавить новый элемент в конец списка
           z=read_data();
           add_last(beg,z);
      }
      printf("еще 1-да 0-нет");
      scanf("%d",&fl);
 } while(fl);
 return beg;
}
//------------ просмотр списка --------------------------------
void print_my_list(my_list *beg) {
 my_list *t;
 if (beg==NULL) {    // если список пуст, то вывести сообщение
      printf("список пуст");
      return;
 }                   // иначе вывести инф. поля элем. списка
 printf("| Номер записи |  имя    |   возраст   |\n");
 t=beg;
 while (t!=NULL) {
      printf("| %10d | %7s | %10d |\n", t->inf.num,t->inf.name,t-
>inf.age);
      t=t->next;
 }
 printf("_____________________________________________\n");
 return;
}
```

3.3 Варианты заданий

Структуру данных из предыдущей, второй, лабораторной работы представить в виде линейного списка, элементами которого являются строки таблицы. Написать функции:

- организации списка;
- просмотра списка;
- добавления элемента в список;
- исключения элемента из списка;
- сохранения в файл;
- загрузки данных из файла и создания по этим данным нового списка;
- освобождения динамической памяти (обязательно вызывать при выходе из программы);
- *одну из функций в соответствии с вариантом, приведенным ниже*.

Значения и количество записей в таблице пользователь выбирает самостоятельно (т.е. количество строк таблицы не задается). Работу программы необходимо оформить в виде меню.

Вариант 1

Функцию, которая вставляет в начало очереди два новых элемента (перед первым элементом).

Вариант 2

Функцию, которая удаляет из непустого списка два первых элемента.

Вариант 3

Функцию, которая вставляет два новых элемента в непустой список после первого элемента.

Вариант 4

Функцию, которая удаляет из непустого списка два элемента, после первого, т.е. удаляет второй и третий элемент.

Вариант 5

Функцию, которая вставляет в конец списка два новых элемента (после последнего).

Вариант 6

Функцию, которая удаляет из непустого списка два последних элемента.

Вариант 7

Функцию, которая вставляет два новых элемент после второго элемента непустого списка.

Вариант 8

Функцию, которая вставляет новый элемент после третьего элемента непустого списка.

Вариант 9

Функцию, которая вставляет два новых элемента перед последним элемента непустого списка.

Вариант 10

Функцию, которая удаляет два предпоследних элемента из непустого списка.

Вариант 11

Функцию, которая вставляет в непустой упорядоченный список по возрастанию значений одного из полей таблицы, новый элемент так, чтобы сохранилась упорядоченность.

Вариант 12

Функцию, которая переносит первый элемент непустого списка после последнего.

Вариант 13

Функцию, которая переносит последний элемент непустого списка перед первым.

Вариант 14

Функцию, которая меняет местами первый и последний элементы непустого списка.

Вариант 15

Функцию, которая меняет местами первый и второй элементы непустого списка.

Вариант 16

Функцию, которая меняет местами второй и третий элементы непустого списка.

Вариант 17

Функцию, которая меняет местами второй и последний элементы непустого списка.

Вариант 18

Функцию, которая меняет местами первый и предпоследний элементы непустого списка.

Вариант 19

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

Вариант 20

Функцию, которая удаляет из непустого списка первый и второй элементы.

Вариант 21

Функцию, которая удаляет из непустого списка второй и третий элементы.

Вариант 22

Функцию, которая удаляет из непустого списка последний и предпоследний элементы, если такие есть.

Вариант 23

Функцию, которая удаляет из непустого списка два предпоследних элемента, если такие есть.

Вариант 24

Функцию, которая добавляет в непустой список два новых элемента перед последним элементом.

Вариант 25

Функцию, которая добавляет в непустой список два новых элемента перед первым элементом**.**

Вариант 26

Функцию, которая добавляет в непустой список два новых элемента перед вторым элементом**.**

Вариант 27

Функцию, которая добавляет в непустой список два новых элемента после последнего элемента**.**

Вариант 28

Функцию, которая добавляет в непустой список два новых элемента после второго элемента**.**

Вариант 29

Функцию, которая переносит второй элемент в конец списка.

Вариант 30

Функцию, которая переносит последний элемент после первого элемента в непустом списке.

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 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, 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 №4 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 данных. 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

Нелинейные структуры предназначены для отображения связей подчиненности элементов данных. Примером нелинейных структур являются иерархические древовидные структуры.

Древовидные структуры данных – это нелинейные структуры, где связь между элементами выражается в терминах «потомок-предок», т.е. такие структуры предназначены для отображения связей иерархической подчиненности элементов данных.

Их применение позволяет значительно увеличить скорость поиска данных.

В программировании часто используются бинарные деревья.

Дерево состоит из вершин (узлов) и связей.

Узел бинарного дерева содержит информационное поле и два указателя – на левое и правое поддерево. Связи между узлами дерева называются его *ветвями*. Вершина (узел) дерева, которая не имеет вышестоящей вершины (предка), называется *корнем*. Считается, что корень дерева расположен на первом уровне.

Максимальный уровень дерева называется его *глубиной*. Количество непосредственных потомков узла называется *степенью узла*. Максимальная степень узла дерева называется *степенью дерева*.

Вершина дерева, которая не имеет дочерних вершин, называется *листом*.

Длина пути до узла – это число ветвей, которое нужно пройти от корня к этому узлу (длина пути к корню равна нулю, узел на уровне `i` имеет длину пути равную `i`).

Степень бинарного дерева равна двум, т.е. каждый узел содержит два указателя: на левое бинарное поддерево (ПД) и правое бинарное поддерево.

Таким образом, бинарное (под)дерево состоит из корня, левого бинарного поддерева и правого бинарного поддерева (рекурсивное определение).

Ниже приведен пример описания узла бинарного дерева:

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

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

Наиболее распространенным условием организации бинарных деревьев является упорядоченность. Элементы дерева в этом случае снабжаются ключевыми признаками. Каждый элемент в упорядоченном дереве имеет на своей левой ветви элементы с меньшими, чем у него, значениями ключей, а на правой ветви элементы с большими значениями ключей.

Упорядоченные бинарные деревья обеспечивают быстрый поиск записи по ее ключу. Такие деревья называют *деревьями бинарного поиска*.

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 рассмотрены в лекциях.

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  my_tree * addtree(my_struct tree *top, int newnode) {
 if (!top) {              //если находимся на уровне листа,
                          //то выделить память под узел
       top=(my_tree*)malloc(sizeof(my_tree));
       if (!top){        // если память не выделена, то выход
       printf("Исчерпана память\n");
       return NULL;
      }
     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                                    //либо в правое поддерево
  top->riqht=addtree(top->riqht,newnode);
 return top;                      //возвращаем указатель на корень
}
```

Если вызов **malloc** 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 соответствующих функций с комментариями. С каждым узлом дерева связана структура, состоящая из фамилии и адреса грузополучателя. Дерево упорядочено по фамилиям.

```
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <iomanip>
#include <conio.h>
#include <iostream>
using namespace std;
//---------------------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
//int n;
     char fio[d_f],
          address[d_a];
};
struct tree {       // описание узла дерева
     gruz_p data;
     tree* left;
     tree* right;
};
//----------------------добавление узла в дерево-------------
tree* addtree(tree *top,const gruz_p& newtree) {
     if (!top) {              //если находимся на уровне листа,
//то выделить память под узел
          top=(tree*)malloc(sizeof(tree));
          if (!top) {         //выход если память не выделена
               cout<<"Не хватает памяти"<<endl;
               return NULL;
          }
          top->data=newtree;            //запись данных в узел
          top->left=NULL;               //обнуление указателей
          top->right=NULL;
     } else    // иначе сравниваем значение в узле с добавляемым
          if (strcmp(top->data.fio,newtree.fio)>0)
          // и добавляем в левое поддерево
                 top->left=addtree(top->left,newtree);
          else                //или правое поддерево
               top->right=addtree(top->right,newtree);
     return top;         //возвращаем указатель на корень дерева
}
// ----------------отображение 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(tree *top, int otstup) {
     if (top) {
          otstup+=3;                    //отступ от края экрана
          otobr(top->right,otstup);    //обход правого поддерева
                //вывод значения фамилии, соответствующего узла
          for(int i=0; i<otstup; i++) printf(" ");
          printf("%s\n",top->data.fio);
          otobr(top->left,otstup);     //обход левого поддерева
     }
}
// ----------------просмотр дерева----------------------------
void prosmotr(tree *top) {         //просмотр сверху вниз
     if (top) {                    //вывод значения корня
          printf("FIO = %s \t",top->data.fio);
          printf("address = %s \n",top->data.address);
          prosmotr(top->left);          //обход левого поддерева
          prosmotr(top->right);        //обход правого поддерева
     }
}
//-------------------ввод данных-----------------------------
gruz_p vvod() {
     gruz_p p;
     printf("\nВведите Ф.И.О.");
          gets(p.fio);
     printf("Введите адрес");
          gets(p.address);
     return p;
}
//------------------- организация дерева ---------------------
tree * org_tree() {
     tree *top=NULL;
     while (1) {
          top=addtree(top,vvod()); //ввод и  добавление элемента
          printf("хотите продолдить (выход - 0)");
          if (getch()=='0')break;
     }
     return top;
}
//-------------------- удаление узла дерева -----------------
tree* delete_node(tree* node, char val[d_f]) {
     if(node == NULL)
          return node;
     if(strcmp(val, node->data.fio)==0) {
          tree* tmp;
          if(node->right == NULL)
               tmp = node->left;
          else {
               tree* ptr = node->right;
               if(ptr->left == NULL) {
                    ptr->left = node->left;
```

```
                    tmp = ptr;
                } else {
                    tree* pmin = ptr->left;
                    while(pmin->left != NULL) {
                        ptr  = pmin;
                        pmin = ptr->left;
                    }
                    ptr->left   = pmin->right;
                    pmin->left  = node->left;
                    pmin->right = node->right;
                    tmp = pmin;
                }
            }
            delete node;
            return tmp;
        } else if(strcmp(val, node->data.fio)<0)
            node->left  = delete_node(node->left, val);
        else
            node->right = delete_node(node->right, val);
        return node;
    }
    //---------- подсчет узлов на заданном уровне дерева ----------
    int tree_count(tree *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=tree_count(top->left,level-1,n); //подсчет узлов в левом ПП
       n=tree_count(top->right,level-1,n);//подсчет в правом ПП
      }
        return n;
    }
```

4.3 Варианты заданий

Структуру данных из предыдущей лабораторной работы использовать в качестве информационного поля бинарного дерева. Написать функции:

- организации дерева,
- просмотра (обхода) дерева,
- отображения структуры дерева;
- добавления узла в дерево,
- исключения узла из дерева,
- сохранения в файл,
- загрузки данных из файла и создания по этим данным нового дерева,
- освобождения динамической памяти, занимаемой деревом (обязательно вызывать при выходе из программы),
- выполнения действий в соответствии с вариантом, приведенным ниже.

Значения и количество записей в таблице пользователь выбирает самостоятельно (т.е. количество строк таблицы не задается, память под узлы дерева выделяется динамически). Работу программы необходимо оформить в виде меню.

Вариант 1

Функция печати самого левого узла непустого дерева.

Функция печати фамилий всех студентов, чей средний балл больше 4.0, а если таких студентов нет, то вывести соответствующее сообщение.

Вариант 2

Функция печати самого правого узла непустого дерева.

Функция печати фамилий всех студентов, имеющих оценки 4 и 5, а если таких студентов нет, то вывести соответствующее сообщение.

Вариант 3

Функция, которая определяет уровень, на котором находится самый левый узел непустого дерева.

Функция печати фамилий всех студентов, имеющих хотя бы одну оценку 2, а если таких студентов нет, то вывести соответствующее сообщение.

Вариант 4

Функция, которая определяет уровень, на котором находится самый правый узел непустого дерева.

Функция печати номеров рейсов, у которых название пунктов отправления и назначения совпадают.

Вариант 5

Функция печати самого левого узла непустого дерева.

Функция печати номеров рейсов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры, а если таких рейсов нет, то выдать на дисплей соответствующее сообщение.

Вариант 6

Функция печати самого правого узла непустого дерева.

Функция печати фамилий работников, чей стаж работы в организации превышает значение, введенное с клавиатуры, а если таких работников нет, то вывести на дисплей соответствующее сообщение.

Вариант 7

Функция, которая определяет уровень, на котором находится самый левый узел непустого дерева.

Функция печати информации о поездах, отправляющихся после введенного с клавиатуры времени, а если таких поездов нет, выдать на дисплей соответствующее сообщение.

Вариант 8

Функция, которая определяет уровень, на котором находится самый правый узел непустого дерева.

Функция печати информации о поездах, направляющихся в пункт, название которого введено с клавиатуры, а если таких поездов нет, то выдать на дисплей соответствующее сообщение.

Вариант 9

Функция, которая заменяет в дереве все элементы меньшие, чем некоторое положительное число A, на это число (по полю «номер поезда»).

Функция печати информации о поездах, отправляющихся из пункта, название которого введено с клавиатуры, а если таких поездов нет, то выдать на дисплей соответствующее сообщение.

Вариант 10

Функция, которая печатает элементы всех листьев дерева.

Функция печати информации о маршруте, номер которого введен с клавиатуры, а если таких маршрутов нет, то выдать на дисплей соответствующее сообщение.

Вариант 11

Функция, которая находит в непустом дереве длину (число ветвей) пути от корня до вершины с элементом, значение порядкового номера которого введено с клавиатуры.

Функция печати информации о маршрутах, которые начинаются или заканчиваются в пункте, название которого введено с клавиатуры, если таких маршрутов нет, выдать на дисплей соответствующее сообщение.

Вариант 12

Функция, которая печатает элементы всех листьев дерева.

Функция печати информации о человеке, номер телефона которого введен с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 13

Функция, которая определяет уровень, на котором находится самый правый узел непустого дерева.

Функция печати информации о людях, чьи дни рождения приходятся на месяц, значение которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 14

Функция, которая определяет уровень, на котором находится самый левый узел непустого дерева.

Функция печати информации о человеке, чья фамилия введена с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 15

Функция, которая печатает элементы всех листьев дерева.

Функция печати информации о человеке, чья фамилия введена с клавиатуры, а если такого нет, выдать на дисплей соответствующее сообщение.

Вариант 16

Функция, которая находит в непустом дереве длину (число ветвей) пути от корня до вершины с элементом, значение порядкового номера которого введено с клавиатуры.

Функция печати информации о людях, родившихся под знаком, название которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 17

Функция, которая определяет уровень, на котором находится самый правый узел непустого дерева.

Функция печати информации о людях, родившихся в месяц, значение которого введено с клавиатуры, а если таких нет, выдать на дисплей соответствующее сообщение.

Вариант 18

Функция печати самого левого узла непустого дерева.

Функция печати информации о товаре, название которого введено с клавиатуры, а если таких товаров нет, выдать на дисплей соответствующее сообщение.

Вариант 19

Функция печати самого правого узла непустого дерева.

Функция печати информации о товарах, продающихся в магазине, название которого введено с клавиатуры, а если такого магазина нет, выдать на дисплей соответствующее сообщение.

Вариант 20

Функция, которая печатает элементы всех листьев дерева.

Функция вычисления среднего арифметического значения всей перечисленной суммы.

Вариант 21

Функция, которая определяет уровень, на котором находится самый правый узел непустого дерева.

Функция печати анкетных данных студентов, получивших одну оценку 3, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 22

Функция, которая печатает элементы всех листьев дерева.

Функция вычисления среднего арифметического значения оценок по физике.

Вариант 23

Функция, которая определяет уровень, на котором находится самый левый узел непустого дерева.

Функция печати анкетных данных студентов, получивших хотя бы одну двойку, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 24

Функция, которая печатает элементы всех листьев дерева.

Функция вычисления среднего арифметического значения оценок по программированию.

Вариант 25

Функция вычисления среднего арифметического значения оценок по истории.

Функция печати фамилий студентов, которые начинаются с литеры ‘А’, и их оценок, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 26

Функция вычисления среднего арифметического значения оценок по математике.

Функция печати анкетных данных студентов, получивших одну оценку 4, а все остальные – 5, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 27

Функция, которая печатает элементы всех листьев дерева.

Функция печати количества студентов, фамилии которых начинаются с литеры ‘Б’, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 28

Функция, которая вычисляет сумму оценок по истории.

Функция печати фамилий студентов, имеющих средний балл выше 4.0, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 29

Функция, которая вычисляет сумму оценок по программированию.

Функцию печати анкетных данных студентов, получивших все пятерки, а если таких студентов нет, вывести соответствующее сообщение.

Вариант 30

Функция, которая подсчитывает количество всех листьев дерева.

Функция печати анкетных данных студентов, не получивших ни одной тройки, а если таких студентов нет, вывести соответствующее сообщение.

4.4 Порядок выполнения работы

4.4.1 Изучить в ходе самостоятельной подготовки способы организации и обработки данных с помощью бинарных деревьев на языке C/C++.

4.4.2 Описать узел бинарного дерева в соответствии с вариантом задачи.

4.4.3 Разработать алгоритм решения задачи, разбив его на отдельные функции согласно варианту 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 От чего 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.

5 ЛАБОРАТОРНАЯ РАБОТА №5 «ПРОГРАММИРОВАНИЕ С ИСПОЛЬЗОВАНИЕМ МОДУЛЕЙ»

5.1 Цель работы

Приобретение навыков разработки и отладки программ, использующих модули.

5.2 Краткие теоретические сведения

5.2.1 Модули

При увеличении объема кода программы человеку становится сложно удерживать все подробности работы в своей памяти. Естественным способом борьбы со сложностью любой задачи является ее разбиение на части. В C++ это может быть сделано с помощью функций, описывающих простые и обозримые задачи. Тогда всю программу можно рассматривать в более укрупненном виде – на уровне взаимодействия функций. Это важно, поскольку человек способен одновременно работать с ограниченным количеством фактов. Использование функций – первый шаг к повышению степени абстракции программы и ведет к упрощению понимания ее структуры.

Использование функций позволяет также уйти от избыточности кода, поскольку функцию записывают один раз, а вызывать ее можно многократно из разных точек программы. Процесс отладки программы, содержащей функции, можно лучше структурировать. Часто используемые функции можно помещать в библиотеки. При разработке могут использоваться функции-заглушки. Таким образом создаются более простые в отладке и сопровождении программы.

Следующим шагом в повышении уровня абстракции программы является группировка функций и связанных с ними данных в отдельные файлы (модули), компилируемые раздельно. Получившиеся в результате компиляции объектные модули объединяются в исполняемую программу с помощью компоновщика. Разбиение на модули уменьшает время перекомпиляции и облегчает процесс отладки, скрывая несущественные детали за интерфейсом модуля и позволяя отлаживать программу по частям (или разными программистами). Модуль содержит данные и функции их обработки. Другим модулям нежелательно иметь собственные средства обработки этих данных, они должны пользоваться для этого функциями первого модуля. Чтобы использовать модуль, нужно знать только его интерфейс. Чем более независимы модули, тем легче отлаживать программу. Это уменьшает общий объем информации, которую необходимо одновременно помнить при отладке. Разделение программы на максимально обособленные части является сложной задачей, которая должна решаться на этапе проектирования программы.

Препроцессором называется первая фаза работы компилятора. Инструкции препроцессора называются **директивами**. Они должны начинаться с символа #.

Директива #include <имя_файла> вставляет содержимое указанного файла в ту точку исходного файла, где она записана. Включаемый файл также может содержать директивы #include. Поиск файла, если не указан полный путь, ведется в стандартных каталогах включаемых файлов. Вместо угловых скобок могут использоваться кавычки **(« »)** – в этом случае поиск файла ведется в каталоге, содержащем исходный файл, а затем уже в стандартных каталогах.

Реальные задачи требуют создания многофайловых проектов с распределением решаемых задач по модулям (файлам).

Исходные тексты совокупности функций для решения какой-либо подзадачи, как правило, размещаются в отдельном модуле (файле). Такой файл называют исходным (*source file* -расширение **.**c или **.**cpp).

Прототипы всех функций исходного файла выносят в отдельный **заголовочный файл** (*header file*). Для него принято расширение .h или .hpp.

Таким образом, заголовочный файл xxx.h содержит **интерфейс** для некоторого набора функций, а исходный файл xxx.c (xxx.cpp) содержит **реализацию** этого набора.

Если некоторая функция из указанного набора вызывается из какого-то другого исходного модуля, то вы обязаны включить в этот модуль заголовочный файл xxx.h с помощью директивы #include.

Негласное правило требует включения этого же заголовочного файла и в исходный файл xxx.c (xxx.cpp).

Директива #include является простейшим средством обеспечения согласованности объявлений в различных файлах, она **включает в файлы информацию об интерфейсе функций** из заголовочных файлов.

В **заголовочных файлах** принято размещать:

- определения типов, задаваемых пользователем, констант;
- объявления (прототипы) функций;
- объявления внешних глобальных переменных (extern);
- комментарии.

В заголовочном файле ***не должно*** быть определений функций и данных. Эти правила не являются требованием языка, а отражают разумный способ использования директивы #include**.**

Обратим внимание, что при использовании директивы #include может возникнуть ***проблема повторного включения заголовочных файлов.***

Самим ходом препроцессирования можно управлять с помощью **условных директив (#if)**. Они представляют собой средство для выборочного включения того или иного текста в программу в зависимости от значения условия, вычисляемого во время компиляции.

Вычисляется **константное целое выражение**, заданное в строке #if. Если оно имеет ненулевое значение, то будут включены все последующие строки вплоть до #endif, или #elif, или #else (директива препроцессора #elif похожа на else if.). Выражение defined(имя), заданное внутри #if есть 1, если имя было определено ранее, и 0 в противном случае.

Например, ***застраховаться от повторного включения*** заголовочного файла `hdr.h` можно следующим образом:

```
#if !defined(HDR) /* страж включения */
#define HDR
                  /* здесь содержимое hdr.h */
#endif
```

При первом включении файла `hdr.h` будет определено имя `HDR`, а при последующих включениях препроцессор обнаружит, что имя `HDR` уже определено, и перейдет сразу на `#endif`.

Директивы `#ifdef` и `#ifndef` специально предназначены для проверки того, определено или нет заданное в них имя. И пример, приведенный выше, можно записать и в таком виде:

```
#ifndef HDR    /* страж включения */
#define HDR
                  /* здесь содержимое hdr.h */
#endif
```

5.2.2 Пример модульной программы

Рассмотрим на примере, пусть написана программа для работы с дробями:

```
#include <stdio.h>
struct drob {
     int ch, zn;
};
//-----------------------------------------
struct drob read_drob()  { //чтение дроби с клавиатуры
     drob d;
     printf("введите числитель и знаменатель\n");
     scanf("%d%d", &d.ch, &d.zn);
     return d;
}
//-----------------------------------------
void write_drob (struct drob d)  {  //печать дроби на экран
     printf("числитель и знаменатель\n");
     printf("%d %d\n", d.ch, d.zn);
}
//-----------------------------------------
int compare(struct drob d1, struct drob d2) {//сравнение
     if (d1.ch*d2.zn==d1.zn*d2.ch)
          return 0;
     else if (d1.ch*d2.zn>d1.zn*d2.ch)
          return 1;
     else return -1;}
//-----------------------------------------
struct drob cancat(struct drob d1, struct drob d2) {
     struct drob d;  //сложение двух дробей
     d.zn=d1.zn*d2.zn;
     d.ch=d1.ch*d2.zn+d1.zn*d2.ch;
     return d;
}
```

```
//-----------------------------------------
int main() {
     drob d1,d2;
     d1=read_drob();       //чтение дроби d1
     d2=read_drob();       //чтение дроби d2
     write_drob(d1);           //печать дроби d1  на экран
     write_drob(d2);           //печать дроби d2 на экран
     int x=compare(d1,d2);     //сравнение d1 и d2
     if (x==0) printf(" дроби равны");
     else if (x==1) printf("d1<d2");
     else printf("d1>d2");
     printf("\n результат суммы: \n");
     d1=cancat(d1, d2);        //сложение двух дробей
     write_drob(d1);
     return 0;
}
```

Разделим нашу программу на несколько файлов:

- в первый вынесем функции ввода-вывода дроби и назовем `io_drob`;

- во-второй вынесем функцию сравнения и сложения двух дробей и назовем `drob_oper`;

- в третий вынесем содержимое функции `main()` и назовем `drob_work`.

Содержимое файла: `io_drob.h`

```
#ifndef IO_DROB_H
#define IO_DROB_H
#include "drob_work.h"
struct drob read_drob();
void write_drob(struct drob d);
#endif
```

Содержимое файла: `io_drob.cpp`

```
#include "io_drob.h"
#include <stdio.h>
struct drob read_drob()  {
     drob d;
     printf("введите числитель и знаменатель\n");
     scanf("%d%d", &d.ch, &d.zn);
     return d;
}
void write_drob (struct drob d)  {
     printf("числитель и знаменатель\n");
     printf("%d %d\n", d.ch, d.zn);
}
```

Содержимое файла: `drob_oper.h`

```
#ifndef DROB_OPER_H
#define DROB_OPER_H
#include "drob_work.h"
int compare(struct drob d1, struct drob d2);
struct drob cancat(struct drob d1, struct drob d2);
#endif
```

Содержимое файла: drob_oper.cpp

```
#include "drob_oper.h"
int compare(struct drob d1, struct drob d2) {
     if (d1.ch*d2.zn==d1.zn*d2.ch)
          return 0;
     else if (d1.ch*d2.zn>d1.zn*d2.ch)
          return 1;
     else return -1;
}
struct drob cancat(struct drob d1, struct drob d2) {
     struct drob d;
     d.zn=d1.zn*d2.zn;
     d.ch=d1.ch*d2.zn+d1.zn*d2.ch;
     return d;
}
```

Содержимое файла: drob_work.h

```
#ifndef DROB_MOD_H
#define DROB_MOD_H
struct drob {
     int ch, zn;
};
void work();
#endif
```

Содержимое файла: drob_work.cpp

```
#include "io_drob.h"
#include "drob_oper.h"
#include <stdio.h>
void work() {
     drob d1,d2;
     d1=read_drob();
     d2=read_drob();
     write_drob(d1);
     write_drob(d2);
     int x=compare(d1,d2);
     if (x==0) printf(" дроби равны");
     else if (x==1) printf("d1<d2");
     else printf("d1>d2");
     printf("\n результат суммы: \n");
     d1=cancat(d1, d2);
     write_drob(d1);
     return;
}
```

Содержимое файла main.cpp

```
#include "drob_work.h"
/* run this program using the console pauser or add your own
getch, system("pause") or input loop */
int main(int argc, char** argv) {
     work();
     return 0;
```

```
}
```

Создание проекта пошагово описано в лекции №8.

5.3 Варианты заданий

Программу из лабораторной работы №3 и №4 необходимо оформить в виде модулей.

5.4 Порядок выполнения работы

5.4.1. Разбить программу на несколько частей. Создать заголовочные файлы.

5.4.2. Создать проект, подключить свои библиотеки

5.4.3. Выполнить компиляцию программы.

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

5.4.5. Выполнить отладку программы.

5.4.6. Исследовать работу программы при различных режимах ее использования.

5.5 Содержание отчета

Цель работы, вариант задания, текст модулей и программы с комментариями, тестовые примеры, выводы.

5.6 Контрольные вопросы

5.6.1 Для чего программу делят на отдельные модули?

5.6.2 Определение препроцессора.

5.6.3 Для чего предназначена иректива #include ?

5.6.4 Определение ***source file***?

5.6.5 Определение ***header file***?

5.6.6 Что принято размещать в заголовочных файлах?

5.6.7 Что подразумевают под проблемой повторного включения заголовочных файлов? Как ее решают?

5.6.8 Что условные директивы?

5.6.9 Определение «страж включения»?

5.6.5 Перечислите стадии разработки проекта. Опишите каждый из четырех этапов.

5.6.11 Что такое тестирование и отладка программы?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вирт, Н. Алгоритмы и структуры данных. Новая версия для Оберона [Электронный ресурс] : учебное пособие / Н. Вирт. — Электрон. дан. — Москва : ДМК Пресс, 2010. — 272 с. — Режим доступа: https://e.lanbook.com/book/1261.— Загл. с экрана.

2. Солдатенко, И. С. Практическое введение в язык программирования Си : учебное пособие / И. С. Солдатенко, И. В. Попов. — Санкт-Петербург : Лань, 2021. — 132 с. — ISBN 978-5-8114-3150-2. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/169287 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

3. Гуськова, О. И. Объектно ориентированное программирование в Java : учебное пособие / О. И. Гуськова. — Москва : МПГУ, 2018. — 240 с. — ISBN 978-5-4263-0648-6. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/122311 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

4. Кауфман, В.Ш. Языки программирования. Концепции и принципы [Электронный ресурс] / В.Ш. Кауфман. — Электрон. дан. — Москва : ДМК Пресс, 2010. — 464 с. — Режим доступа: https://e.lanbook.com/book/1270. — Загл. с экрана.

5. Андрианова, А.А. Алгоритмизация и программирование. Практикум [Электронный ресурс] : учебное пособие / А.А. Андрианова, Л.Н. Исмагилов, Т.М. Мухтарова. — Электрон. дан. — Санкт-Петербург : Лань, 2019. — 240 с. — Режим доступа: https://e.lanbook.com/book/113933. — Загл. с экрана.

6. Березовская, Ю. В. Основы программирования на JAVA: лабораторный практикум : учебно-методическое пособие / Ю. В. Березовская. — Архангельск : САФУ, 2016. — 113 с. — ISBN 978-5-98450-442-3. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/161794 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

7. Макаров, Е. М. Элементы двумерной графики в Java : учебно-методическое пособие / Е. М. Макаров. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2017. — 56 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/152985 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

8. Хабитуев, Б. В. Программирование на языке Java: практикум : учебное пособие / Б. В. Хабитуев. — Улан-Удэ : БГУ, 2020. — 94 с. — ISBN 978-5-9793-1548-5. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/171791 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

9. Пруцков, А. В. Язык программирования Java. Введение в курс: объектно-ориентированное программирование : учебное пособие / А. В. Пруцков. — Рязань : РГРТУ, 2016. — 56 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/168308 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

10. Сыромятников, В. П. Структуры и алгоритмы обработки данных: Практикум : учебное пособие / В. П. Сыромятников. — Москва : РТУ МИРЭА, 2020. — 244 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/163915 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

11. Кораблин, Ю. П. Структуры и алгоритмы обработки данных : учебно-методическое пособие / Ю. П. Кораблин, В. П. Сыромятников, Л. А. Скворцова. — Москва : РТУ МИРЭА, 2020. — 219 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/163860 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

12. Пруцков, А. В. Язык программирования Java. Введение в курс: операторы и типы данных : учебное пособие / А. В. Пруцков. — Рязань : РГРТУ, 2016. — 72 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/168307 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

13. Гуркова, М. А. Программирование на языке Си: Практикум : учебное пособие / М. А. Гуркова, Э. Р. Резникова. — Москва : РУТ (МИИТ), 2020. — 70 с. — Текст : электронный // Лань : электронно-библиотечная система. — URL: https://e.lanbook.com/book/175947 (дата обращения: 17.08.2021). — Режим доступа: для авториз. пользователей.

Заказ № _____ от «_____» _____2022г. Тираж _____экз.
Изд-во СевГУ