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. — 118 с.

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

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 С++, 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
символов
  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 С.

1.6.2 Как хранится строка языка С в памяти ЭВМ?

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

1.6.4 Как выделить динамическую память под строку?

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

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

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

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

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

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

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

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

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

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

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