
Институт
Радиоэлектроники
и Информационных
Технологий
ВСТ
кафедра

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Р.Е. АЛЕКСЕЕВА»

Институт радиоэлектроники и информационных технологий

Кафедра «Вычислительные системы и технологии»

ПРОГРАММИРОВАНИЕ НА C++
Практика

Учебное пособие для студентов дневной формы обучения
по направлению подготовки 09.03.01
"Информатика и вычислительная техника"

Профиль подготовки
“Вычислительные машины, комплексы, системы и сети”

Квалификация (степень) - бакалавр

Нижний Новгород
2019 год

Составитель Мартынов Д.С.

УДК 004.432.2

Рецензент – доцент, кандидат технических наук Д.В. Жевнерчук

Программирование на C++: Практика. учеб.пособие/ Д.С. Мартынов; Нижегородский государственный технический университет им. Р.Е. Алексеева, 2019. – 00 с.

Данное пособие содержит учебный материал по основам алгоритмизации и программирования, задания для лабораторных работ, контрольные вопросы по дисциплине "Программирование" в соответствии с учебным планом для направления подготовки 09.03.01 "Информатика и вычислительная техника" во втором учебном семестре. В качестве языка программирования выбран язык *C++* в его спецификации *C++11*.

В приложении приведены примеры программ, использующие средства управления консольного форматного вывода в спецификации *ISO C++11*.

Редактор _____

Подп. к печ. _____Формат 60x84 $^{1}/_{16}$. Бумага офсетная. Печать офсетная. Печ. л. _6__. Уч.-изд. л. ____. Тираж ____ экз. Заказ ____.

Нижегородский государственный технический университет
имени Р.Е. Алексеева.
Типография НГТУ.
Адрес университета и полиграфического предприятия:
603950, г. Н.Новгород, ул. К.Минина, 24.

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Настоящее учебное пособие содержит учебно-методический материал, предназначенный для изучения курса "Программирование" студентами дневного отделения по направлению подготовки 09.03.01 "Информатика и вычислительная техника". Пособие включает в себя несколько тематических блоков, затрагивающих разделы объектно-ориентированного программирования на языке C++ и использования средств его стандартной библиотеки. Приведены задания для лабораторных работ.

По каждой лабораторной работе оформляется отчет в установленной форме и производится защита теоретических знаний по изучаемому разделу курса.

Отчет к лабораторной работе должен включать в себя текст задания, описание метода решения задачи в виде блок-схемы и псевдокода, исходный текст программы, включая комментарии к исходному тексту программы, распечатку вывода результатов работы программы.

Для оформления результатов самостоятельной работы студент должен завести отдельную рабочую тетрадь. Отчет по каждому заданию оформляется в письменной форме, допускается вклеивание листингов исходных текстов программ на изучаемом языке программирования. Результат выполнения каждого задания должен быть представлен скриншотом.

1. Механизмы управление программами средствами POSIX API

Согласно концепции, заложенной в архитектуру UNIX, компьютер представляет собой совокупность файлов, любые программные и аппаратные ресурсы, соответственно, рассматриваются как файлы. При этом, для файлов реализованы следующие стандартные команды-действия: открыть (*open*), закрыть (*close*), прочитать (*read*), записать (*write*), переместиться внутри файла в заданную позицию (*lseek*) и выполнить файл (*exec*). Данные команды образуют низкоуровневый прикладной программный интерфейс (*API*). Его базис был реализован при разработке языка программирования С и операционной системы *UNIX*. По мере развития языков программирования и операционных систем, данный интерфейс также развивался, в него вводились новые языковые и системные средства. Однако данный базовый набор до сих пор используется на низкоуровневых программно-аппаратных платформах, таких, как встраиваемые системы и системы на микроконтроллерах.

В данном разделе рассматриваются задачи, связанные с запуском программ на исполнение, управлением данным процессом и определением ресурсов для запускаемых программ.

Процесс – это программа, находящаяся в стадии выполнения. В полном объеме материал, связанный с управлением процессами будет рассмотрен в рамках курса «Параллельные вычисления», сейчас вы познакомитесь лишь с базовыми возможностями по управлению процессом запуска программ и организацией взаимодействия между ними. Важно знать, что процесс является артефактом операционной системы (ОС), ему назначаются системные ресурсы, и он обладает определенными свойствами, которые принято называть атрибутами процесса.

Процесс запуска программы на исполнение инициируется пользователем. Но программу на исполнение запускает операционная система. При этом под пользователем может подразумеваться не только человек, но и программная система. Важно, что пользователю в системе назначаются определенные права, определяющие разрешения и запреты на выполнение определенных действий в отношении системных ресурсов. Самый важный пользователь в *UNIX/Linux* – системе имеет имя *root*. Его часто называют суперпользователем. Все остальные пользователи в системе имею меньше прав и только *root* вправе изменять права всех остальных пользователей, зарегистрированных в системе. Но это уже вопрос, больше относящийся к области администрирования операционной системы, нас же будут интересовать вопросы и задачи программирования. Тем не менее, необходимо знать, что на *UNIX/Linux* – платформах для того чтобы иметь возможность запускать исполняемые файлы, нужно

обладать на это правами, заданными в профиле пользователя. То есть, системная политика позволяет определять такие свойства для пользователей, при которых последний сможет запускать на исполнение лишь строго заданный список программ и не более того. Причем задается и список директорий, в которых может выполняться запуск разрешенных программ.

Аргументы функции *main*()

Ранее вам уже пришлось познакомиться с принципом программного управления, предложенным Норбертом Винером, и сегментной организацией адресного пространства приложения, работающего под управлением ОС семейства *UNIX/Linux*. Вы уже знаете, что работа программы начинается с запуска на исполнение реализованной в ней функции *main*().

Стандарт *POSIX* определяет формат для прототип функции *main*(), с передачей управления которой начинается работа любого приложения:

```
int main(int argc, char** argv);
```

либо равноценный

```
int main(int argc, char* argv[]);
```

Данный прототип регламентирует количество и список аргументов функции *main*(), которые она получает через свой стек. Две формы указания типа второго параметра: *char** argv* и *char* argv*[] – эквивалентны.

Командный интерпретатор, при помощи которого осуществляется запуск программы, формирует и передает запускаемой программе аргументы, указанные в командной строке вместе и именем программы. То есть строка, переданная командному интерпретатору, содержащая имя запускаемой программы, это и есть аргументы, передаваемые функции *main*() запускаемой программы. Даже если вы осуществляете запуск программы из графической среды, имейте в виду, что запуск программы все равно инициируется командным интерпретатором пусть и в скрытой от вас форме. Архитектура большинства операционных систем не предусматривает другого механизма запуска программ, как только с использованием средств командного интерпретатора (командной оболочки). Фактически это означает, что правами на запуск программ пользователя обладает только командный интерпретатор, который осуществляет специальные системные вызовы. Любой запуск программ на исполнение, минуя командный интерпретатор, на системах, работающих

под управлением многозадачных ОС, невозможен. Ну, если только система не взломана, а это уже совершенно другая тема.

Рассмотрим параметры функции main():

- *int argc* – (*ARGument Count*) , количество переданных аргументов;
- *char** argv* – (*ARGument Vector*), массив строк переданных функции *main*() аргументов.

Ряд платформ поддерживает ещё и третий, не обязательный параметр – *char** envp* – массив строк переменных окружения. В этом случае прототип функции *main*() принимает следующий вид:

```
int main(int argc, char** argv, char** envp);
```

Предложенное именование аргументов функции *main*() носит рекомендательный характер, но данная рекомендация обеспечивает хорошую читаемость исходного кода программы. В ряде источников встречаются примеры определения параметров функции *main*(), подобные следующему: *int main*(*int a, char** b*), но такое именование параметров существенно менее информативно. Поэтому, целесообразно придерживаться предложенной системы именования параметров функции *main*().

Рассмотрим механизм передачи фактических параметров функции *main*() со стороны командного интерпретатора на следующем примере:

```
>> cat file1 file2 <Enter>
```

Ввод данных, переедаемых командному интерпретатору, завершается с нажатием клавиши <Enter>. Символы, стоящие в начале строки данного примера (">>"), обозначают начало строки, формируемой командным интерпретатором в качестве приглашения для ввода команд. Командный интерпретатор считывает и обрабатывает вводимые строки: выделяет отдельные блоки символов, заключенные в кавычки и лексемы (слова). В данном примере строка ввода включает в себя три лексемы: "cat", "file1" и "file2". Будет сформированы следующие значения аргументов для функции *main*(): *argc* = 3, *argv*[0] = "cat", *argv*[1] = "file1", *argv*[2] = "file2" и *argv*[3] = NULL. Все эти аргументы доступны для чтения в самой программе, но изменять их непосредственно возможности, да и необходимости, нет.

Рассмотрим пример, демонстрирующий механизм работы с аргументами командной строки.

```
// filename: progr01.c
```

```
#include <stdio.h>
int main(int argc, char** argv) {
    int i=1;
    printf("Argumen counting: %d\n", argc);
    printf("Programs name : %s\n", argv[0]);
    if(argv > 1) {
        while(argv[i] != NULL) {
            printf("Argment[%d] : %s\n", i, argv[i]);
            i++;
        }
    }
    return 0;
}
```

Сохраним этот файл с именем progr01.c и скомпилируем его из командной строки.

```
$gcc –o my_program progr01.c
```

Имя созданной программы – *my_program*. Данная программа при запуске выводит сообщение, содержащее информацию о количестве аргументов, с которыми она была запущена, ее название и аргументы командной строки. Имя программы передается в функцию *main*() в качестве параметра *argv*[0].

Параметры запуска являются лексемами и разделяются пробельными символами. Разрешается группировать параметры в виде отдельной строки, заключенной в двойные кавычки.

Данную схему взаимодействия с программным окружением используют многие сервисные программы, запускаемые из командной строки. Она позволяет передавать задаваемым программам дополнительную управляющую информацию, определяющую логику их работы. Даже компилятор получает информацию из командной строки о том, какой исходный файл необходимо ему обработать и что именно с ним нужно сделать.

Набор параметров, передаваемых программе при запуске, может содержать параметры двух категорий: опции запуска, начинающиеся с символа тире или пары символов тире, именуемые опциями запуска и параметры опций, которые следуют после опций.

Например, программа – утилита *ls* (*list*), предназначенная для вывода на экран содержимого текущего каталога, поддерживает набор управляющих опций, определяющих, как именно нужно организовать вывод информации о содержимом текущего каталога. Эти управляющие опции называются ключами запуска.

Рассмотрим следующий пример:

```
>> ls       /* вывести список имен файлов в текущей директории */
>> ls -l    /* просмотр файлов в одно колоночном формате */
>> ls –a    /* отображать все файлы, включая скрытые */
```

Результат работы программы зависит от того, с какими ключами будет она запущена.

В первом случае программа при запуске получает только один аргумент, содержащий имя программы: "*ls*". Во втором и третьем случае, программа запускается с двумя аргументами: "*ls*" и "*-l*", и "*ls*" и "*-a*" соответственно. По стандарту *POSIX* программа при запуске получает дополнительный аргумент, завершающий набор строк ключей, это *NULL* – указатель. Рассмотрим это на примере использования утилиты *echo*, входящей в состав поставки большинства операционных систем.

```
$echo Hello, World
```

Значение параметра *argv* равно трем. Набор строк – аргументов запуска, переданных ОС программе *echo* в стек времени исполнения:

```
argv[0] = "echo",
argv[1] = "Hello,",
argv[2] = "World",
argv[3] = NULL.
```

Для следующего примера количество аргументов запуска будет рабно двум:

```
$echo “Hello, World!”,
```

так набор символов, заключенных в двойные кавычки, является строкой и интерпретируется как единый аргумент.

Реализация программы *echo* на языке *C* достаточно проста. Данная утилита выводит в стандартный поток вывода переданные аргументы. Она принимает значение управляющего ключа "-n", запрещающего переход на новую строку после завершения вывода.

```
#include <stdio.h>
int main(int argc, char** argv) {
    int i=1, sign=1;
    if(argc>1) {
        if(argv[1][0]==’-‘ && argv[1][1]==’n’) {
            sign=0;
        }
        printf(“\n”);
        while(argv[i++]!=NULL) {
             printf(“ %s”, argv[i]);
```

```
        }
    }
    if(sign==1) {
        printf("\n");
    }
    return 0;
}
```

Следует отметить, что существовали различные способы передачи управляющей информации в запускаемую программу. В ряде случаев управляющие ключи начинались с символа слеш ('/'), в других – с символа тире ('-'), либо двух символов тире ('--'). Кроме того, существуют механизмы для задания параметров для отдельных ключей. Рассмотрим следующий формат утилиты MS-DOS format.

FORMAT том [/FS:файловая_система] [/V:метка] [/Q] [/A:размер] [/C] [/X] [/P:проходы] [/S:состояние]

где *том* – определяет букву диска (с последующим двоеточием),

/FS:файловая_система – указывет тип файловой системы,

/V:метка – метка тома,

/Q – быстрое форматирование.

Например, "*C:> format C:*" – «любимая» многими поколениями прикладных программистов команда, предназначенная для форматирования жесткого диска…

Для получения подробной справки по работе с данной утилитой использовался ключ '/?': *"format /?".*

Для пользователей операционной системы семейства Linux отдельный интерес может представлять утилита *at* – планировщик запуска заданий. Вот пример ее использования:

$ at -f my_script.sh 8:00am

- выполнить файл сценария *my_script.sh* в 8 утра сегодня, или завтра, если 8 утра уже наступило.

Подробное изучение системных утилит предусмотрено учебным планом дисциплины «Принципы и методы организации программных средств», в рамках курса «Программирование» выполняется лишь начальное ознакомление с принципами разработки системного программного обеспечения.

На данном этапе обучения программирования перед студентами ставится задача изучения механизма обработки ключей из разрабатываемых ими консольных приложений.

Задание

1. Запустите терминал. Определите название используемого командного интерпретатора.

2. Ознакомьтесь с встроенной справочной системой (man). Изучите способы получения контекстной справочной информации.
3. Изучите базовые команды для навигации по файловой системе и манипуляции с файлами: `ls`, `pwd`, `cd`, `cp`, `rm`, `mv`, `mkdir`.
4. Изучите базовые команды для управления вывода на экран: `cat`, `more`, `less`, `pg`.
5. Изучите справку по команде more. Опишите алгоритм ее работы и используемые механизмы. Создайте функционально близкий функции more аналог средствами языка *C*++. Сравните работу базовой и созданной функции. Сделать выводы.
6. Изучите справку по команде cat. Описать алгоритм ее работы и используемые механизмы. Создать функционально близкий функции more аналог средствами языка *C*++. Сравнить работу базовой и созданной функции. Сделать выводы.

ЛАБОРАТОРНАЯ РАБОТА №1
Передача аргументов программе из командной строки. Переменные среды

Цель работы

1. Изучить правила передачи параметров программе при запуске.
2. Исследовать отличия между длинными и короткими опциями в стандарте POSIX.
3. Научится использовать механизмы управления запуска программ в своих программных проектах.
4. Научиться внедрять в текст программу справочную информацию.

Задания к лабораторной работе

Согласно заданию составить алгоритм и написать программу на языке C++. Программа компилируется и запускается под управлением ОС Linux. Разработанная программа должна содержать встроенную справочную информации, описывающую правила использования, цель назначения и информацию о разработчике. Аргументы запуска программа должна обрабатывать согласно рекомендациям POSIX.

Варианты заданий

1. Написать программу, работающую в текстовом режиме, которая должна выводить на экран содержимое текстового файла, имя которого передается в программу с помощью аргументов командной.

Все строки в файле длиннее 80 символов обрезать до 80 символов. Цвет символов - белый. Вверху экрана выводить имя файла. Цвет фона - черный. При нажатии клавиш *Up* & *Down* прокручивать экран по вертикали. При

нажатии клавиш *Left* & *Right* прокручивать экран по горизонтали, т.е. выводить ранее обрезанные строки.

2. Написать программу, работающую в текстовом режиме (ширина экрана равна 80 столбцов), которая должна выводить на экран содержимое бинарного файла, имя которого передается в программу с помощью аргументов командной строки.

Содержимое бинарного файла выводить в шестнадцатеричном виде по 16 байт в строке, между каждыми четырьмя байтами вставляя пробел. Вверху экрана выводить имя файла. В левой части экрана выводить смещение (в шестнадцатеричной форме) 16-байтной строки относительно начала файла. Цвет символов - белый. Цвет фона - черный. При нажатии клавиш Up & Down прокручивать экран по вертикали.

Пример выполнения:

```
test.exe
00000000  455D103D 56431DFC 66CA41AA 123309BB
00000010  D54D1000 06434D8C 66CA41AA 123309BB
...
```

3. Написать программу, работающую в текстовом режиме (ширина экрана равна 80 столбцов), которая должна выводить на экран содержимое текстового файла, имя которого передается в программу с помощью аргументов командной.

Все строки в файле длиннее 80 символов необходимо переносить по словам на следующую строку экрана. Вверху экрана выводить имя файла. Цвет символов - белый. Цвет фона - черный. При нажатии клавиш *Up & Down* прокручивать экран по вертикали.

4. Написать программу, принимающую в качестве аргумента командной строки имя текстового файла и выводящую содержимое файла в стандартный поток вывода. Если содержимое файла не умещается на одном машинном экране, файл выводится постранично: сначала первая страница, потом вторая и так далее. Каждая последующая страница выводится на экран после нажатия пользователем клавиши *Enter*.

5. Подготовить несколько исполнимых файлов, выводящих на экран различные сообщения. Подготовить текстовый файл, в котором заданы пути к этим исполнимым файлам. Написать программу, читающую из текстового файла имена исполнимых программ и запускающую их в порядке, противоположном тому, в котором они указаны в файле. Имя текстового файла жестко задано в теле программы.

6. Написать программу, принимающую в качестве аргументов командной строки два имени текстового файла и параметр, определяющий режим записи. Программа считывает содержимое первого файла и записывает его во второй

файл. Режим записи определяет, будет ли второй файл перезаписан или данные первого файла будут добавляться в конец второго файла.

7. Написать программу, читающую информацию со стандартного потока ввода и записывающую ее в файл. Имя файла программа принимает в качестве аргумента командной строки. Чтение со стандартного потока продолжается до тех пор, пока пользователь не нажмет '*Ctrl+Z*' (*EOF*).

8. Написать программу, работающую в текстовом режиме (ширина экрана равна 80 столбцов), которая должна выводить на экран текстовый файл (имя передается через аргументы командной строки). Строки длиннее 80 символов обрезаются. Цвет символов - серый. Цвет фона - черный. Слова, список который жестко задан в массиве:

```
char *keyWords[] = {
"int",
"float",
...
};
```

должны выводиться белым цветом. При нажатии клавиш *Up*, *Down* прокручивать экран.

9. Подготовить несколько исполнимых файлов, выводящих на экран различные сообщения. Подготовить два текстовых файла, содержащих пути к этим исполнимым файлам, порядок следования исполняемых файлов для каждого текстового файла произвольный. Написать программу, принимающую в качестве аргумента командной строки путь к текстовому файлу, читающую из файла имена исполнимых программ и запускающую их в том порядке, в котором они указаны в файле.

10. Написать программу, принимающую в качестве аргумента командной строки имя текстового файла и выводящую содержимое файла в стандартный поток ошибок(*STDERR*) в обратном порядке.

11. Написать программу, принимающую в качестве аргумента командной строки имя текстового файла и выводящую содержимое файла построчно в стандартный поток вывода: сначала первая строка, потом вторая и так далее. Каждая последующая строка выводится на экран после нажатия пользователем клавиши *Enter*.

12. Написать программу, принимающую в качестве первого аргумента командной строки имя текстового файла, в качестве второго и третьего аргумента - любые слова. Программа должна выводить содержимое файла в стандартный поток вывода, заменяя все вхождения в файле слова, заданного вторым аргументом, словом, заданным третьим аргументом.

13. Написать программу, принимающую в качестве аргумента командной строки имя файла. Если этот файл имеет расширение *.txt*, программа должна вывести его содержимое в стандартный поток вывода. Если этот файл имеет расширение *.exe*, программа должна запустить его на выполнение. (Необходимо подготовить исполнимый файл, выводящих на экран какое-нибудь сообщение). Если этот файл имеет какое-то другое расширение, программа должна вывести соответствующее сообщение и завершиться.

14. Написать программу, работающую в текстовом режиме (ширина экрана равна 80 столбцов), которая должна выводить на экран содержимое двух бинарных файлов, имена которых передаются в программу с помощью аргументов командной строки.

Содержимое бинарных файлов выводить в шестнадцатеричном виде по 16 байт в строке, между каждыми четырьмя байтами вставляя пробел. В левой части экрана выводить смещение (в шестнадцатеричной форме) 16-байтной строки относительно начала файла. Файлы отделять вертикальной чертой. Вверху экрана выводить имена файлов. Различающиеся байты файлов отображать красным цветом. Цвет остальных символов - белый. Цвет фона - черный. При нажатии клавиш *Up* & *Down* прокручивать экран по вертикали.
Пример выполнения:

```
test.exe bc.exe
00000000  455D103D 56431DFC 66CA41AA 123309BB | 4D000000 00101000 60CA1111
1131111B
00000010  D54D1000 06434D8C 66CA41AA 123309BB | D54D1000 06434D8C 66CA41AA
123309BB
...
```

15. Написать программу, работающую в текстовом режиме (ширина экрана равна 80 столбцов), которая должна выводить на экран список файлов и каталогов в текущей директории (в которой запущена программа). Если весь список не умещается на одном экране, то при нажатии клавиш *Up*, *Down* прокручивать список.

16. Написать программу, принимающую в качестве аргументов командной строки четыре имени текстовых файлов. Программа считывает содержимое первого файла и записывает: во второй файл - гласные латинские символы, встречающиеся в первом файле, в третий файл - согласные латинские символы, встречающиеся в первом файле, в четвертый файл - остальные символы, встречающиеся в первом файле.

17. Написать программу, принимающую в качестве аргумента командной строки имя текстового файла. Программа считывает содержимое переменных окружения системы и записывает их в файл, имя которого принимает в качестве параметра.

18. Написать программу, выполняющую перемещение файла, начальное и конечное имена файлов передаются в программу через аргументы командной строки.

19. Написать программу, принимающую в качестве аргументов командной строки два имени текстовых файлов. Программа считывает содержимое первого файла, преобразует символы латиницы нижнего регистра в символы латиницы верхнего регистра, а символы латиницы верхнего регистра в символы латиницы нижнего регистра, и записывает его во второй файл.

20 Для текстового файла, имя которого передается через командную строку, подсчитать количество: 1) символов; 2) символов без пробелов; 3) букв; 4) слов; 5) предложений, и вывести сформированную статистику на экран.

21 Считать текстовый файл, имя которого передано через командную строку, произвести преобразование текста, написанного латиницей, к верхнему регистру. Результат работы сохраняется в обрабатываемом файле
Пример: begin -> BEGIN

22 Подготовить исполнимый файл (1), принимающий из командной строки один параметр (имя файла) и выводящий содержимое файла на экран. Подготовить исполнимый файл (2), принимающий из командной строки два параметра: имя исполнимого файла, который он запускает, и строковый параметр, который он передает запускаемому файлу через командную строку. Осуществить запуск исполнимого файла (2) в связке с исполнимым файлом (1).

23 Программа принимает через командную строку: 1) имя текстового файла; 2) произвольную строку символов (ключ).

Программа должна вывести на экран только те строки файла, в которых содержится строка символов (ключ), переданная через командную строку вторым аргументом. При выводе на экран строки файла, выделять вхождение ключа другим цветом.

24 Написать программу, которая разбивает большой файл на ряд файлов меньшего размера и собирает ряд файлов в один. При разбивке имя разбиваемого файла и размер конечного файла передавать через командную строку. При сборке имя первого собираемого файла передавать через командную строку, остальные файлы искать в текущей директории, увеличивая номер в расширении файла.

У разбиваемого файла менять имя и расширение:
filename.ext -> filename_ext.pXX, где *XX* - это номер части (от 00 до 99).
При сборке производить обратную операцию.
Пример:
файл *BigFile.exe* размер 5 456 223 байт
конечные файлы:

BigFile_exe.p01 размер 2 000 000 байт
BigFile_exe.p02 размер 2 000 000 байт
BigFile_exe.p03 размер 1 456 223 байт

Пример работы программы:
разбиение:
```
c:\>file_sc.exe -s BigFile.exe 2000000
```

сборка:
```
c:\>file_sc.exe -c BigFile_exe.p01
```

25 Написать программу, которая составляет список файлов в текущей директории, сортирует файлы в списке по расширению (в алфавитном порядке) и записывает этот отсортированный список в текстовый файл.

26 Написать программу, принимающую в качестве аргументов командной строки два имени текстовых файлов. Программа считывает содержимое первого файла, подсчитывает число вхождений каждой буквы, и записывает результат во второй файл.

27 Подготовить несколько текстовых файлов с сообщениями. Подготовить специальный текстовый файл, в котором заданы пути к текстовым файлам с сообщениями. Написать программу, принимающую в качестве аргумента командной строки путь к текстовому файлу, читающую из файла имена текстовых файлов с сообщениями и выводящую текстовые файлы с сообщениями на экран.

28. Написать программу, принимающую в качестве аргументов командной строки имя текстового файла и параметр, определяющий режим вывода содержимого файла на экран. Программа выводит содержимое файла на экран в зависимости от режима вывода либо в обычном символьном виде, либо в шестнадцатеричном формате (по два знакоместа на символ под старшие и младшие 4 бита). При выводе в шестнадцатеричном формате коды соседних символов разделять пробелами.

Рекомендации по организации кода программы

1. Старайтесь осмысленно использовать директиву *`using`*, определяя при помощи неё использование нужных средств стандартной библиотеки. Например, целесообразно использовать

```
using std::cout;
using std::cin;
using std::endl;
using std::string;
```

вместо использования всего пространства имен стандартной библиотеки.

```
using namespace std;
```

Целесообразно использовать пространство имен для разделения области различных частей проекта, содержащего объявления нескольких классов. Пространство имен позволяет исключить конфликт имен для классов и методов с одинаковыми именами, определенными в различных частях проекта.

2. Каждый блок кода программы должен быть сопровожден соответствующими комментариями. Комментарии к программе должны быть составлены на русском языке. Старайтесь делать код программы читаемым, давать переменным и методам осмысленные имена.

Раздел 2. Файловые потоки ввода-вывода в C++

ЛАБОРАТОРНАЯ РАБОТА №2
Использование структур (Агрегативные типы данных)

Цель работы

1. Изучить правила организации пользовательских (агрегативных) типов данных.
2. Исследовать механизмы организации полей данных в структурах и способы обращения к ним.
3. Научится использовать структуры для разработки проблемно ориентированных типов данных (абстрактных типов данных, АТД).
4. Научиться разрабатывать и использовать хранилища данных, организованные по принципу электронных таблиц.

Практическая работа.
Агрегативный тип данных

Структура является одним из базовых типов для языков C и C++, она относится к категории агрегативных типов данных и используется для создания типов данных, определяемых пользователем. С точки зрения теории информации, структурой называют некоторый набор данных и связи, наложенные на эти данные. Благодаря имеющимся связям между данными они выделяются из внешней информационной среды. Можно сказать, что структура – это способ организации информации. Структуры позволяют описывать категории физического мира терминами языка программирования и являются базой для создания абстрактных типов данных (ADT – Abstract Data Type)

Структура – это совокупность данных разного типа, собранных под одним именем. Структура состоит из набора полей, каждое из которых является экземпляром одного из базовых типов данных. Поля, определяющие состав структуры, в C++ называются ее атрибутами. Существует ряд ограничений, накладываемых на типы полей структур: в языке C полями структур не могут быть функции и экземпляры структур того же типа, что и структура, в которой они объявляются, но могут быть указатели на функции и указатели на структурные переменные того же типа. В *C++* атрибутами структур могут быть и функции, но, как и в *C*, атрибутами не могут быть переменные того же структурного типа.

Структуры, содержащие указатели на структуры того же типа, называются самоссылочными структурами.

Общий вид объявления структур в *C*/*C++*.

```
struct <Имя_составного_типа> {
    <тип_1> <Имя_атрибута_1>;
```

```
    <тип_2> <Имя_атрибута_2>;
       *     *     *     *
    <тип_N> <Имя_атрибута_N>;
};
typedef struct <Имя_составного_типа> {
    <тип_1> <Имя_атрибута_1>;
    <тип_2> <Имя_атрибута_2>;
       *     *     *     *
    <тип_N> <Имя_атрибута_N>;

} <Псевдоним_для_составного_типа>;
```

Объявление структуры является декларацией о том, каким образом должна быть организована информация при создании экземпляров структурных переменных, эта декларация не связана с выделением памяти, память выделяется только при объявлении структурных переменных. Рассмотрим следующие примеры.

Объявление структуры – составного типа данных.

```
struct TPoint2D { /* объявляется структурный тип TPoint2D, */
    float x;      /* содержащий два поля типа float        */
    float y;
};  /*объявление структуры заканчивается пустым оператором */
```

Объявление структуры, совмещенное с объявлением двух структурных переменных.

```
struct TPoint2D { /* объявляется структурный тип TPoint2D, */
    float x;      /* содержащий два поля типа float        */
    float y;
} A, B;
```

Следует обратить внимание на тот факт, что при объявлении структурных переменных в языке *C* необходимо использовать ключевое слово `struct`, являющееся частью имени структурного типа, а в *C++* его можно опускать:

```
struct A { int val; }; /* Объявление структурного типа */
struct A a; /* Объявление переменной структурного типа по правилам
               ANSI C*/
```

Поэтому в ANSI C целесообразно использовать форму объявления структурного типа, совмещенную с определением псевдонима для типа (`typedef`).

```
typedef struct TA { int val; } A; /*Объявление структурного типа*/
A a;                  /* Объявление переменной структурного типа */
```

В языке программирования *C++* эта конструкция является избыточной, так как компилятор ведет собственный «реестр» пользовательских типов данных, объявленных в проекте, и добавляет необходимую информацию в объявления переменных. Приведем вариант объявления типа *A* для языка *C++*.

```
struct A { int val; }; /*Объявление структурного типа*/
A a;                   /* Объявление переменной структурного типа */
```

Для доступа к полям структур используются две операции: доступ по имени (`.`) и доступ через указатель (`->`). Рассмотрим следующий пример.

```
typedef struct Point2D { float x; float y;} TPoint2D;
TPoint2D A, B={0F, 0.5F};
```

```
TPoint2D* ptr=&A; /* объявление указателя на структуру          */
A.x=1.15F;        /* обращение к полю структуры по имени       */
A->y=2.34F;       /* обращение к полю структуры через указатель */
```

Механизмы выделения памяти для структур в куче такие же, как и для скалярных типов данных.

```
typedef struct Point3D {float x; float y; float z;} TPoint3D;
TPoint3D* arrayPoint=(TPoint3D*)malloc(sizeof(TPoint3D)*10);
if(arrayPoin==NULL) {
    /* обработка ошибок выделения памяти */
}
arrayPoint[0]->x=0F;
arrayPoint[0]->y=0F;
arrayPoint[0]->z=0F;
    *    *    *
free(arrayPoint); /* освобождение памяти в куче */
```

Язык программирования C++ позволяет объявлять поля структур типа функции, это обусловлено том, что в C++ ключевое слово `struct` является синонимом для ключевого слова `class`. В тоже время, в языке C можно использовать в качестве полей структур указатели на функции, но не сами функции, это связано с механизмами организации исполняемого кода, создаваемого C – компилятором.

Изучение утилиты who

Задания к лабораторной работе

Согласно заданию составить алгоритм и написать программу на языке C++. Программа компилируется и запускается под управлением ОС Linux. Разработанная программа должна содержать встроенную справочную информации, описывающую правила использования, цель назначения и информацию о разработчике. Аргументы запуска программа должна обрабатывать согласно рекомендациям POSIX.

Разрабатываемая программа предназначена для хранения массива структур (записей), она должна поддерживать управление на уровне аргументов командной строки (аргументов запуска).

Поддерживаемые опции запуска:

--help либо *-h* - запуск программы в режиме получения справки. После вывода справочной информации программа завершает работу.

-c [*N*] [*file_name*] - запуск программы в режиме создания электронной таблицы записей, *N* – количество записей, *file_name* – имя текстового файла, в котором будет сохранен массив (таблица) записей.

-r [*N*] [*file_name*] - запуск программы в режиме чтения содержимого текстового файла *file_name*, на экран должны быть выведены не более *N* записей. Следует учесть, что реальное количество записей в файле может не совпадать с заданным значением *N*. Если заданный файл окажется пуст, либо по какой-либо причине программа не сможет его открыть, должно быть выдано соответствующее сообщение.

В случае если программа будет запущена с неопределенными разработчиком аргументами, программа должна выдать соответствующее сообщение и вывести минимальную справку о корректных аргументах запуска. Это так же касается случая, когда программа запускается без аргументов.

Варианты заданий

Варианты заданий работы

1. Самолеты

Наименование типа	Фамилия конструктора	Год выпуска	Количество кресел	Грузоподъемность, т

2. Расчет движения

Наименование воздушной линии	Тип самолета	Количество рейсов	Налет, тыс. км	Пассажиро-оборот, человеко-км

3. Перевозки

Тип самолета	Номер борта	Количество рейсов	Налет в часах	Налет, тыс. км

4. Расписание

Номер рейса	Наименование рейса	Тип самолета	Стоимость билета	Протяженность линии, км

5. Аэропорт

Наименование	Площадь здания	Этажность здания	Год сооружения	Оценочная стоимость, млн.руб

6. Ремонт аэродромных сооружений

Наименование	Шифр	Вид ремонта	Сметная стоимость ремонта	Наименование подрядчика

7. Кассы авиабилетов

Номер кассы	Ф.И.О. кассира	Количество проданных билетов	Суммарная выручка	Дата продаж

8. Технические характеристики парка используемых ПК

Тип процессора	Тактовая частота	Емкость ОП, МБ	Емкость ЖМН, ГБ	Тип монитора

9. Города

Наименование	Количество жителей	Площадь, кв.км	Год основания	Количество школ

10. Московские мосты

Наименование	Высота	Ширина	Количество опор	Протяженность

11. Линии московского метро

Наименование	Район линии	Год пуска	Протяженность, км	Количество поездов на линии

12. Легковые автомобили

Марка	Год выпуска	Пробег	Изготовитель	Цена

13. Учет продажи программных продуктов

Наименование	Фирма изготовитель	Стоимость	Вид лицензии	Имеющееся количество экземпляров

14.Учет абонентов городской телефонной станции

Ф.И.О. абонента	Номер телефона	Год подключения	Вид тарифа	Плата за месяц

15. Детские сады

Наименование детского сада	Номер сада	Количество детей	Район города	Сумма платежа за месяц

16. Учет сотрудников (для отдела кадров)

Ф.И.О.	Табельный номер	Дата рождения	Оклад, тыс.руб.	Стаж работы

17. Ведомость зарплаты за текущий месяц

Ф.И.О.	Номер отдела	Табельный номер	Количество рабочих дней	Размер зарплаты

18. Музеи

Наименование	Назначение	Адрес	Часы работы	Стоимость билета

19. Экскурсии

Наименование	Страна	Стоимость	Продолжительность	Вид транспорта

20. Киноафиша

Наименование кинотеатра	Название картина	Время сеансов	Стоимость билетов	Адрес

21. Книга-почтой

Наименование книги	Ф.И.О. автора	Номер по каталогу	Издательство	Стоимость книги

22. БД квартир в агентстве недвижимости

Адрес	Площадь, кв.м.	Сторона света	Стоимость 1 кв.м.	Этаж	Количество комнат

23. Система учета заказов для магазинов

Номер магазина	Наименование товара	Артикул товара	Цена единицы товара	Количество товара / Размер заказанной партии

24. Телевизоры на складе магазина

Наименование	Фирма-изготовитель	Стоимость	Размер экрана	Количество на складе

25. Холодильники на складе магазина

Наименование	Фирма-изготовитель	Стоимость	Емкость холодильной камеры	Емкость морозильной камеры	Количество на складе

Раздел 4. Основы объектно-ориентированного программирования (ООП)

ЛАБОРАТОРНАЯ РАБОТА №3
Проектирование и использование классов

Цель работы

1. Изучить правила организации классов, как пользовательских (агрегативных) типов данных.
2. Исследовать сокрытия данных как один из базисных принципов объектно-ориентированного программирования.
3. Научится правильно проектировать и использовать интерфейс класса (функции-члены, методы).

4. Используя разработанные классы, создать прототип базы данных, организованной по принципу электронных таблиц (массив элементов класса).

Задания к лабораторной работе

Согласно заданию составить алгоритм и написать программу на языке C++. Программа компилируется и запускается под управлением ОС Linux. Разработанная программа должна содержать встроенную справочную информации, описывающую правила использования, цель назначения и информацию о разработчике. Аргументы запуска программа должна обрабатывать согласно рекомендациям POSIX.

Разрабатываемая программа предназначена для хранения массива экземпляров класса. Созданная электронная таблица, массив экземпляров класса должен сохраняться в бинарном файле. Для получения информации об объеме записей, в файл так же должен быть записан так называемый дескриптор электронной таблицы. Дескриптор, как минимум, должен содержать информации о количестве данных, которые записаны в указанный файл.

Перечень атрибут класса (членов-данных) определяется исходя их задания во второй лабораторной работе. Созданная программа должна поддерживать управление на уровне аргументов командной строки (аргументов запуска).

Поддерживаемые опции запуска:

--help либо *-h* - запуск программы в режиме получения справки. После вывода справочной информации программа завершает работу.

-c [*N*] [*file_name*] - запуск программы в режиме создания электронной таблицы записей, *N* – количество записей, *file_name* – имя бинарного файла, в котором будет сохранен массив (таблица) записей.

-r [*N*] [*file_name*] - запуск программы в режиме чтения содержимого бинарного файла *file_name*, на экран должны быть выведены не более *N* записей. Следует учесть, что реальное количество записей в файле может не совпадать с заданным значением *N*. Если заданный файл окажется пуст, либо по какой-либо причине программа не сможет его открыть, должно быть выдано соответствующее сообщение.

В случае, если программа будет запущена с неопределенными разработчиком аргументами, программа должна выдать соответствующее сообщение и вывести минимальную справку о корректных аргументах запуска. Это так же касается случая, когда программа запускается без аргументов.

ЛАБОРАТОРНАЯ РАБОТА №4
Наследование, перегрузка операций

Цель работы

1. Изучить механизмы наследования при создании иерархии классов.
2. Исследовать правила перегрузки операций.

3. Научится использовать структуры для разработки проблемно ориентированных типов данных (абстрактных типов данных, АТД).
4. Научиться разрабатывать и использовать хранилища данных, организованные по принципу электронных таблиц.

Задания к лабораторной работе

Согласно заданию составить алгоритм и написать программу на языке C++. Программа компилируется и запускается под управлением ОС Linux.

В случае если студент разрабатывает программу, работающую под управлением операционной системы MS Windows, необходимо обеспечить максимальную переносимость кода. То есть не использовать не стандартные Microsoft-расширения языка *C++*, привязанные к среде разработки MS Visual Studio.

Разработанная программа должна содержать встроенную справочную информации, описывающую правила использования, цель назначения и информацию о разработчике. Аргументы запуска программа должна обрабатывать согласно рекомендациям POSIX.

Создать пользовательский класс TCharArray (массив), используемый для хранения элементов типа char. В данном классе должен быть реализован метод at для доступа к элементу символьного массива с проверкой корректности значения индекса элемента массива. Кроме того, необходимо перегрузить операцию [ ] для доступа к элементам массива. Используя класс TCharArray в качестве родительского, создать производный от него пользовательский класс String, используемый для хранения символьных строк. Для данного класса перегрузить следующие операции: ‘+’, ‘==’, ‘>’, ‘<’, ‘!=’

Разрабатываемая программа предназначена для хранения массива экземпляров класса. Перечень атрибут класса (членов-данных) определяется исходя их задания во второй лабораторной работе. Созданная программа должна поддерживать управление на уровне аргументов командной строки (аргументов запуска).

Поддерживаемые опции запуска:

--help либо *-h* - запуск программы в режиме получения справки. После вывода справочной информации программа завершает работу.

-c [*N*] [*file_name*] - запуск программы в режиме создания электронной таблицы записей, *N* – количество записей, *file_name* – имя бинарного файла, в котором будет сохранен массив (таблица) записей.

-r [*N*] [*file_name*] - запуск программы в режиме чтения содержимого бинарного файла *file_name*, на экран должны быть выведены не более *N* записей. Следует учесть, что реальное количество записей в файле может не совпадать с заданным значением *N*. Если заданный файл окажется пуст, либо по какой-либо причине программа не сможет его открыть, должно быть выдано соответствующее сообщение.

В случае, если программа будет запущена с неопределенными разработчиком аргументами, программа должна выдать соответствующее

сообщение и вывести минимальную справку о корректных аргументах запуска. Это так же касается случая, когда программа запускается без аргументов.

ЛАБОРАТОРНАЯ РАБОТА №5
Виртуальные функции. Полиморфизм

Цель работы

1. Изучить механизмы вызова методов производного класса через указатель на балловый класс.
2. Исследовать внутреннюю организацию классов при использовании наследования.
3. Научится использовать механизмы наследования для проектирования информационных систем (проблемно-ориентированного программного обеспечения).
4. Научиться разрабатывать и использовать хранилища данных, организованные по принципу электронных таблиц.

Задания к лабораторной работе

Согласно заданию составить алгоритм и написать программу на языке C++. Программа компилируется и запускается под управлением ОС Linux.

В случае если студент разрабатывает программу, работающую под управлением операционной системы MS Windows, необходимо обеспечить максимальную переносимость кода. То есть не использовать не стандартные Microsoft-расширения языка *C*++, привязанные к среде разработки MS Visual Studio.

Разработанная программа должна содержать встроенную справочную информации, описывающую правила использования, цель назначения и информацию о разработчике. Аргументы запуска программа должна обрабатывать согласно рекомендациям POSIX.

Варианты индивидуальных заданий.

Вариант №1. Журнал аудита. Разработать иерархию классов, которая позволит смоделировать работу службы аудита.			
Базовый класс	Событие		
Производные классы	Уведомление	Предупреждение	Ошибка

Вариант №2. Журнал авиа диспетчерской службы. Разработать иерархию классов, которая позволит смоделировать работу авиа диспетчерской службы.				
Базовый класс	Летательный аппарат (ЛА)			
Производные классы	Пилотируемый летательный аппарат (ПЛА)		Не пилотируемый летательный аппарат (НПЛА)	
Производные классы	Самолёт	Вертолёт	Метеозонд	Летательный аппарат-робот

Вариант №3. Система учета подвижного состава автотранспортного предприятия.

Разработать иерархию классов, которая позволит смоделировать работу системы учета подвижного состава автотранспортного предприятия.				
Базовый класс	Дородно-транспортное средство (ДТС)			
Производные классы	Пассажирский автобус	микроавтобус	Передвижная ремонтная мастерская	Тягач

Вариант №4. Система учета сотрудников компании. Разработать иерархию классов, которая позволит смоделировать работу системы учета сотрудников IT – компании.				
Базовый класс	Категории работников			
Человек	Стажер (студент)	Программист	Тестировщик ПО	Руководитель отдела

Вариант №5. Система учета . Разработать иерархию классов, которая позволит смоделировать работу системы планирования распорядка дня.				
Базовый класс	Событие			
Производные классы	Встреча	Напомнить	День рождения	Заметка

Вариант №6. Система учета кадров ВУЗа. Разработать иерархию классов, которая позволит смоделировать работу отдела кадров ВУЗа						
Базовый класс	Человек (Person)					
Производные классы	Сотрудник			Студент		
Производные классы	Преподавательский состав	Учебно-вспомогательный состав	Административно-хозяйственные службы	Бакалавриат	Магистратура	Аспирантура

Вариант №7. Система учета кадров ВУЗа. Разработать иерархию классов, которая позволит смоделировать работу отдела кадров ВУЗа						
Базовый класс	Элемент цепи					
Производные классы	Пассивный электрический элемент			Активный электрический элемент		
Производные классы	сопротивление	Источник питания	Проводник (соединительный элемент)	Емкость (конденсатор)	Индуктивность	Триод (биполярный транзистор)

Приложение 1.

Этапы проектирование программного обеспечения (ПО) при структурном подходе.

Программное средство (ПС) – это совокупность программ определенного назначения, пригодных для исполнения на ЭВМ. Отдельная программа может быть как частью существующего ПС, так и самостоятельным ПС.

Программист должен создавать правильно, корректно работающие программы.

Правильность или корректность – это свойство соответствия проверяемого объекта некоторому эталонному объекту или совокупности формализованных эталонных характеристик и правил. Корректность программы при проектировании наиболее полно определяется степенью соответствия предъявляемым к ней формализованным требованиям – программной спецификации.

Рассмотрим задание на написание программы, принимающей в качестве аргументов запуска простое арифметическое выражение, состоящее из двух операндов в формате с плавающей точкой и знака математической операции, стоящей между ними. Операнды должны быть разделены пробельными символами. Программа должна реализовывать инфиксный калькулятор, вычисление значения выражения вида "операнд_1 операция опернд_2". В данном случае корректность работы программы подразумевает получение правильного результата вычисления математического выражения и, возможно, выдачу сообщений об ошибке в случае передаче программе неверных параметров. После вывода результата в стандартный поток вывода программа должна завершить свою работу. Предположим, что созданный исполняемый модуль будет именован CALC.EXE, в этом случае запуск программы из командной строки

"C:\>CALC.EXE 4 + 5" [Enter] (на Microsoft-платформе),

"$./CALC 4 + 5" [Enter] (на Linux-платформе),

должен привести к выводу сообщения-результата, соответствующего значению 9, а запуск

"C:\>CALC.EXE 4 + 9 > RES.TXT" [Enter]

должен привести к созданию текстового файла, содержащего символ '9' и символ конца файла *EOF*.

При отсутствии полностью формализованной спецификации требований в качестве эталона, которому должна соответствовать программа и результата ее функционирования, иногда используются неформализованные представления разработчика, пользователя или заказчика программ. Однако понятие корректности программ по отношению к запросам пользователя или заказчика сопряжено с неопределенностью самого этого эталона, которому должна соответствовать программа. Вследствие этого понятие корректности программ становится субъективным и его невозможно определить количественно. Неопределенность программных спецификаций уменьшается в

процессе разработки программ путем уточнения требований по согласованию между разработчиком и заказчиком.

Однако внешне правильная и корректно работающая программа может содержать в себе скрытые, трудно выявляемые ошибки, связанные с ошибками проектирования. В частности, в языке Си существует много так называемых узких мест, связанных с функциями обработки строк. Большинство программистов знает о возможных ошибках, связанных с передачей функции *printf*( ) строк - символьных массивов. По вине программиста переданная строка может не содержать так называемого нуль-терминатора, нулевого байта, символизирующего конец строки, что может привести к срыву стека исполняемой программы. Однако при дальнейшем анализе функций обработки строк мы видим целый куст проблем-ошибок, связанных с обработкой таких незавершенных строк. Даже базовая функция *strlen*() не страхует нас от получения неверного результат при передаче ей слишком длинных строк в случае если в ее реализации нет проверки состояния флага переполнения разрядной сетки.

Аналогичная ситуация связана с обработкой исключений в *C*++. Если программист не реализует программные средства обработки исключений, за него эту работу выполняет среда исполнения его программ, однако наиболее часто заданная по умолчанию обработка исключений заключается в аварийном завершении работы программы с выдачей соответствующего сообщения.

В случае если ошибка в работе программы проявляется в ходе ее эксплуатации, то возможны два пути ее локализации:

1. ошибка в работе программы проявляется лишь при определенных, крайне редко используемых условиях, но исправление ее невозможно в силу сложности ПС, в этом случае ошибка документируется;
2. если это возможно, то ошибка в работе программы должна быть изучена и устранена.

Таким образом, микропроектирование ПО возлагает на программиста-разработчика ответственность за знание особенностей программно-аппаратной платформы. Именно поэтому необходимо осознанно формировать культуру программирования уже на начальном этапе обучения. Начинающий программист должен прочувствовать важность грамотной разработки даже малых программ, наличие четкой структуры приложения, ясно отражающей его спецификацию.

Рассмотрим жизненный цикл ПО, он включает в себя следующие основные этапы:

1) системный анализ, в ходе которого определяется потребность в ПО, его назначении и основных функциональных характеристиках, оцениваются затраты и возможная эффективность применения такого комплекса;
2) проектирование ПО, включающее в себя разработку структуры комплекса и его компонент, программирование модулей и ряд этапов отладки, а также испытание и внедрение для регулярной эксплуатации созданной версии ПО;

3) эксплуатацию ПО, заключающуюся в исполнении, функционировании программ на ЭВМ для обработки информации и получения результатов, являющихся целью создания ПО, а также в обеспечении достоверности и надежности выдаваемых данных;
4) сопровождение ПО, состоящее в эксплуатационном обслуживании, развитии функциональных возможностей и повышении эксплуатационных характеристик ПО, тиражировании и переносе ПО на различные виды вычислительных средств.

Следствием данного жизненного цикла ПО является каскадная схема разработки ПО, определяющая следующую последовательность действий:

1. постановка задачи;
2. анализ требований и определение спецификаций, примерно 40% времени, затрачиваемого на разработку ПО;
3. проектирование,40%;
4. реализация, 5%;
5. тестирование, 15%;
6. модификация.

Этапы со второго по пятый касаются непосредственно реализации ПО, шестой – процесса сопровождения ПО. Первый этап касается непосредственно разработки технического задания (ТЗ), являющегося для команды разработчиков основным документом, описывающим конечных результат их труда.

При циклической схеме разработки ПО происходит последовательное, циклическое повторение этих этапов. При возврате на первый этап происходит уточнение ТЗ, на этапе тестирования выявляются ошибки проектирования и реализации, которые в свою очередь влекут за собой внесение изменений в ТЗ и очередной проход по этапам проектирования и реализации.

При реализации относительно небольших программ, на этапе проектирования программист должен ответить на следующий ряд вопросов:

1. Что необходимо сделать?
2. Каким образом это необходимо сделать?
3. Почему именно так он должен сделать и каковы альтернативы?
4. Что произойдет, если он сделает именно так?

Исходя из вышеизложенного, на этапе проектирования начинающему программисту рекомендуется следующий сценарий работы:

1. Анализ задачи (задания к лабораторной работе).
2. Описание входного потока данных, например, аргументы, переданные программе через командную строку, текстовый файл и т.д. На данном этапе необходимо явно выделить ”первичный”[1] и ”вторичный” потоки данных, если таковые имеются. Например, ”первичный” поток данных может передаваться программе при запуске, а вторичный поток данных определяется для программы при выполнении заложенного в нее алгоритма, набора инструкций.

3. Описание выходного потока данных, порождаемого программой, например, сообщение, выводимое на экран, или создаваемый текстовый файл, или значение, возвращаемое программой операционной системе при завершении работы. На данном этапе также явно выделить ”первичный”, основной, поток данных, порождаемый программно, и вторичный поток, например поток сообщений ошибок *STDERR*, либо поток диалогов с пользователем.
4. Составить схему, отображающую используемые программой информационные потоки и объекты, участвующие в организации информационного обмена с указанием условий, при которых данные потоки порождается.
5. Описание возможных, допускаемых программой "ошибок"[2], правил, по которым выявляются "ошибки". Оформление описания данных ошибок как отдельного файла протокола, поставляемого с программой.
6. Описание правил обработки программой "ошибок", как во входном потоке, так и в выходном потоке данных. То есть если предыдущий этап рассматривать как описание того, что может случиться, то на данном этапе необходимо описать, как данные события-ошибки необходимо обрабатывать.
7. Описание этапов последовательного, пошагового выполнения программы. Выделение функциональных блоков и описание их в виде отдельных модулей, функций.
8. Описание пошагового сценария работы программы с организацией ветвления при реализации логики работы программы как в штатном, нормальном режиме работы, так и в случае обработки ошибок. Составление обобщенной блок схемы функционирования приложения.
9. Составить размеченный граф состояний, соответствующий пошаговой работе программы. Каждой вершине графа поставить в соответствие целочисленное значение.
10. Исходя из номеров вершин в графе, описывающем ПО, определить значения, возвращаемые функциями-модулями с учетом обработки ошибок и досрочного завершения работы программы.
11. Перейти от линейной структуры пошагового исполнения к циклической, а внутри цикла процедуру множественного выбора-ветвления.

Распишем процесс создания программы переименования фалов на основании данного сценария проектирования ПС.

Пункт 1.

Задание: разработать программу, принимающую через аргументы командной строки два имени файлов файл_1 и файл_2, разделенные пробелом, осуществляющую переименование-перенос файла_1 в файл_2. Программа должна поддерживать опцию получения справки и корректно отрабатывать попытку запуска с неправильными параметрами.

Изучив справочник по параметрам командной строки мы находим существующую программу-прототип *RENAME*, являющуюся исполняемым файлом. Вот содержимое встроенной справки о данной программе.

RENAME
Изменение имени отдельного файла. Команда *rename* перечисленными ниже параметрами доступна только при использовании консоли восстановления.
Команда *rename* другими параметрами доступна из командной строки.

rename [*диск*:][*путь*] *имя_файла1 имя_файла2*

-или-

ren [*диск*:][*путь*] *имя_файла1 имя_файла2*

Параметры

[*диск*:][*путь*] *имя_файла1*

Имя и размещение файла, который требуется переименовать. Использование подстановочных символов не допускается.

имя_файла2

Новое имя файла. При переименовании не могут быть заданы новый диск или каталог.

Примечания

- Переименование файлов

Допускается переименование всех файлов, соответствующих заданному имени файла. Команду *rename* нельзя использовать для переименования файлов на разных дисках или для их перемещения в другой каталог.

- Использование подстановочных знаков при переименовании

Подстановочные знаки (* и ?) могут быть использованы в параметрах, задающих имена. Если они использованы в параметре *имя_файла_2*, то символы, замещаемые символами подстановки, будут теми же, что и в параметре *имя_файла_1*.

- Команда переименования не будет работать, если *имя_файла_2* уже существует.

Если имя файла, задаваемое параметром *имя_файла_2*, уже существует, команда *rename* выведет на экран следующее сообщение:

```
Дублирование имени файла или файл не найден
```

Пункт 2.

Первичный поток данных, обрабатываемых программой, представляет собой аргументы, переданные программе при запуске, аргументы функции *main*( ). Вторичным, порождаемым под управлением первичного, потоком является поток данных, содержащихся в файле_1, файле источнике информации.

Причем в первичном потоке могут содержаться опции запуска, инициирующие вывод справки о данной программы, такие как *-h* и *--help* (стандарт *POSIX*).

Пункт 3.

Выходной поток данных работы программы может быть представлен:

1. Созданным программой переименованным файлом в случае успешного запуска программы.
2. Выводом в стандартный поток вывода, либо в стандартный поток ошибок сообщения об ошибке.
3. Выводом в стандартный поток вывода справочной информации об использовании данной программы.

При работе программы возможны следующие "ошибки":

1. Программа запущена с неверным числом параметров.
2. Программе передано имя несуществующего файла источника данных, либо у текущего пользователя нет прав для его обработки: чтение, переименование, удаление после создания копии с новым именем.
3. У пользователя, с чьими правами запущена программа, нет прав на создание нового файла, либо модификации существующего, в случае если происходи перезаписывание файла-приемника.

Пункт 4. (схема информационных потоков)

Участники информационного взаимодействия в данной программе: Пользователь, Программа, Операционная система, Файлы на физическом носителе.

В качестве “Пользователя” данного ПС может выступать человек, оператор ПК, либо другая программа, в случае, если проектируемое ПС планируется использовать в качестве программного сервиса (программы, работающей под управлением другой программы).

Состав участников информационно цепи:

”Источник Информации” - ”Приемник Информации” - зависит от конкретного этапа работы программы. Можно выделить следующие шаги-этапы в работе программы:

1. Запуск программы. Пользователь (ИИ) → Программа (ПИ).
2. Досрочное завершение работы программы с выдачей соответствующего сообщения. Программа (ИИ) → Пользователь, другая Программа (ПИ).
3. Процесс создания копии файла под новым именем. Файлы на физическом носителе (ИИ) → Программа (ПИ) → Файлы на физическом носителе (ПИ).

Получаем следующую иерархию участников информационного обмена: Пользователь <=> Программа <=> Операционная система <=> Файл.

Пункт 5. (описание возможных "ошибок" при работе программы**)**

При работе программы возможны следующие "ошибки":

1. программа запущена с недопустимым числом параметров, то есть число параметров запуска не равно 1 — запуск без параметров, либо 2 — запуск с опцией получение справки о программе, либо 3 — запуск в ”штатном” режиме;

2. программе передано имя несуществующего файла источника данных, либо у текущего пользователя нет прав для его обработки: чтение, переименование, удаление после создания копии с новым именем;
3. у пользователя, с чьими правами запущена программа, нет прав на создание нового файла, либо модификации существующего, в случае если происходи перезаписывание файла-приемника;
4. ошибки в файловом потоке, в том числе при обработке слишком длинных файлов, либо при нарушении целостности структуры файла, отсутствие символа *EOF*.

Пункт 6. (правила обработки ошибок**)**

В случае запуска программы с недопустимым числом параметров программа должна выдать соответствующее сообщение в стандартный поток ошибок *STDERR*: “Слишком большое число аргументов, переданных программе” и завершить свою работу с возвратом операционной системе значения 1 (`exit(1);` либо *`return 1;`*).

В случае отказа в доступе к файлу источнику данных программа выдает соответствующее сообщение в стандартный поток ошибок *STDERR:* ”у текущего пользователя нет прав чтения файла <указывается имя файла>” и завершить свою работу с возвратом операционной системе значения 2 (`exit(2);` либо *`return 2;`*).

В случает отказа со стороны операционной системы в праве на создание файл приемник информации программа выдает соответствующее сообщение в стандартный поток ошибок *STDERR:* ”у текущего пользователя нет прав создания файла в указанной директории” и завершить свою работу с возвратом операционной системе значения 3 (`exit(3);` либо *`return 3;`*).

В случае если программа создала копию файла под новым именем, но у пользователя, с чьими правами запущена данная программа, нет прав на удаление файла источника данных, либо для него выставлен атрибут ”Read Only” (только для чтения), программа должна удалить, если это возможно, созданную под новым именем копию файла и в стандартный поток ошибок *STDERR* выдать сообщение: ”ошибка при переименовании — перемещении файла”, и завершить свою работу с возвратом операционной системе значения 4 (`exit(4);` либо *`return 4;`*).

Пункт 7. (Описание этапов последовательного, пошагового выполнения программы. Выделение функциональных блоков и описание их в виде отдельных модулей, функций**)**

1. чтение количества аргументов функции *`main(int argc, char** argv)`* , значения переменной *`argc`*.
2. если количество аргументов больше трех, то досрочное завершение работы программы с выдачей соответствующего сообщения.

3. если количество аргументов равно одному — программа запущена без параметров (*argc==1*) программа выводит справочную информацию о себе, содержащую справку о формате запуска.
4. если количество аргументов равно двум и опция запуска содержит ключ *"-h" "--help"*, то программа запущена в режиме вывода расширенной справки о ее использовании, включающую контактную информацию о разработчике.
   ```
   if((argc==2)&&((!strcmp(argv[1],"-h"))
       ||(!strcmp(argv[1],"—help")))
   ```
   После вывода соответствующей информации программа завершает свою работу.
5. *if(argc==3)* полагаем, что программа запущена в штатном режиме переименования файла. Интерпретируем значения строк *argv*[1] и *argv*[2] как соответствующие исходное старое и новое имена файла. Организуем процедуры открытия файлов в соответствующих режимах.

Пункт 8. (Описание пошагового сценария работы программы с организацией ветвления при реализации логики работы программы как в штатном, нормальном режиме работы, так и в случае обработки ошибок. Составление обобщенной блок схемы функционирования приложения**)**

Пункт 9. (размеченный граф состояний, соответствующий пошаговой работе программы**)**

Пункт 10. (Исходя из номеров вершин в графе, описывающем ПО, определить значения, возвращаемые функциями-модулями с учетом обработки ошибок и досрочного завершения работы программы**)**

Пункт 11. (Перейти от линейной структуры пошагового исполнения к циклической, а внутри цикла процедуру множественного выбора-ветвления.)

```
while(step!=0)
  {
     switch(step)
       {
          case 1: … ; step=2; break;
          case 2: … ; step=3; break;
          case 3: … ; step=4; break;
          case 4: … ; step=0; break;
          default: perror("Error!");/* обработка внути-
                                       программных ошибок */
}
```

Примечания

1. Термины ”первичный” и ”вторичный” потоки данных введены автором для описания порядка их обработки, в порядке создания-порождения.

2. В данном случае под понятием "ошибка" подразумевается не сбой в работе программы, а нарушение нормального режима работы программы, приводящего к реализации главной функции программы, ради которой она создана, в следствии передаче программе заведомо неверных данных. Это не ошибка исполнения программы, а несоблюдение правил диалога между пользователем и программой, либо между взаимодействующими программами. Данные "ошибки" можно описать на этапе проектирования и в состав ПО ввести средства их обработки, как средства обработки внештатных ситуаций.

Приложение 2.

Условное обозначение элементов блок-схемы

Данный раздел составлен на основании ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения.

Символ обозначения	Использование
Символы данных	
	Символ данных. Символ отображает данные, носитель данных не определен.
	Запоминаемые данные. Символ отображает хранимые данные в виде, пригодном для обработки, носитель данных не определен.
	Специфические символы данных Оперативное запоминающее устройство Символ отображает данные, хранящиеся в оперативном запоминающем устройстве.
	Запоминающее устройство с последовательным доступом Символ отображает данные, хранящиеся в запоминающем устройстве с последовательным доступом (магнитная лента, кассета с магнитной лентой, магнитофонная кассета).
	Запоминающее устройство с прямым доступом Символ отображает данные, хранящиеся в запоминающем устройстве с прямым доступом (магнитный диск, магнитный барабан, гибкий магнитный диск).
	Документ Символ отображает данные, представленные на носителе в удобочитаемой форме (машинограмма, документ для оптического или магнитного считывания, микрофильм, рулон ленты с итоговыми данными, бланки ввода данных).
	Ручной ввод. Символ отображает данные, вводимые вручную во время обработки с устройств любого типа (клавиатура, переключатели, кнопки, световое перо, полоски со штриховым кодом).
	Карта Символ отображает данные, представленные на носителе в виде карты (перфокарты, магнитные карты, карты со считываемыми метками, карты с отрывным ярлыком, карты со сканируемыми метками).
	Дисплей Символ отображает данные, представленные в человекочитаемой форме на носителе в виде отображающего устройства (экран для визуального наблюдения, индикаторы ввода информации).
Символы процесса	
	Процесс Символ отображает функцию обработки данных любого вида (выполнение определенной операции или группы операций, приводящее к изменению значения, формы или размещения информации или к определению, по которому из нескольких направлений потока следует двигаться).
	Ручная операция Символ отображает любой процесс, выполняемый человеком.
	Подготовка Символ отображает модификацию команды или группы команд с целью воздействия на некоторую последующую функцию (установка переключателя, модификация индексного регистра или инициализация программы).
	Решение Символ отображает решение или функцию переключательного типа, имеющую

	один вход и ряд альтернативных выходов, один и только один из которых может быть активизирован после вычисления условий, определенных внутри этого символа. Соответствующие результаты вычисления могут быть записаны по соседству с линиями, отображающими эти пути
	Параллельные действия Символ отображает синхронизацию двух или более параллельных операций.
	Граница цикла Символ, состоящий из двух частей, отображает начало и конец цикла. Обе части символа имеют один и тот же идентификатор. Условия для инициализации, приращения, завершения и т.д. помещаются внутри символа в начале или в конце в зависимости от расположения операции, проверяющей условие.
Символы линий	
	Линия Символ отображает поток данных или управления.
	Передача управления Символ отображает непосредственную передачу управления от одного процесса к другому, иногда с возможностью прямого возвращения к инициирующему процессу после того, как инициированный процесс завершит свои функции. Тип передачи управления должен быть назван внутри символа (например, запрос, вызов, событие).
	Пунктирная линия Символ отображает альтернативную связь между двумя или более символами. Кроме того, символ используют для обведения аннотированного участка.