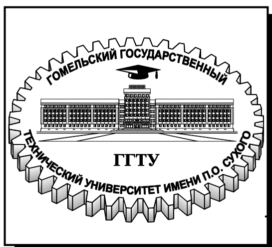

Министерство образования Республики Беларусь

Учреждение образования
«Гомельский государственный технический
университет имени П. О. Сухого»

Кафедра «Промышленная электроника»

Э. М. Виноградов, А. В. Сахарук

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ. УКАЗАТЕЛИ И МАССИВЫ

ПРАКТИКУМ
по выполнению лабораторных работ
для студентов специальности 1-53 01 07
«Информационные технологии и управление
в технических системах»
дневной формы обучения

Гомель 2017

Библиотека ГГТУ им. П.О. Сухого

УДК 004.43(075.8)
ББК 32.973-018.2я73
В49

Рекомендовано научно-методическим советом машиностроительного факультета ГГТУ им. П. О. Сухого (*протокол № 11 от 27.06.2016 г.*)

Рецензент: зав. каф. «Автоматизированный электропривод» ГГТУ им. П. О. Сухого канд. техн. наук, доц *В. С. Захаренко*

Виноградов, Э. М.

В49

Технология разработки программного обеспечения систем управления. Указатели и массивы : практикум по выполнению лаборатор. работ для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» днев. формы обучения / Э. М. Виноградов, А. В. Сахарук. – Гомель : ГГТУ им. П. О. Сухого, 2017. – 57 с. – Систем. требования: PC не ниже Intel Celeron 300 МГц ; 32 Mb RAM ; свободное место на HDD 16 Mb ; Windows 98 и выше ; Adobe Acrobat Reader. – Режим доступа: http://library.gstu.by. – Загл. с титул. экрана.

Содержит четыре лабораторные работы с основными теоретическими сведениями, порядком их выполнения, заданиями для самостоятельной работы и контрольными вопросами.

Для студентов специальности 1-53 01 07 «Информационные технологии и управление в технических системах» дневной формы обучения.

УДК 004.43(075.8)
ББК 32.973-018.2я73

© Учреждение образования «Гомельский государственный технический университет имени П. О. Сухого», 2017

Библиотека ГГТУ им. П.О.Сухого

Лабораторная работа № 1

Компилятор gcc и интегрированная среда Qt Creator

1 Цель работы

Изучить принципы работы с компилятором gcc, сборку однофайловых и многофайловых проектов. Изучить основы работы с интегрированной средой разработки Qt Creator.

2 Основные теоретические сведения

Компиля́ция – это трансляция программы, составленной на исходном языке высокого уровня, в эквивалентную программу на низкоуровневом языке, близком машинному коду (абсолютный код, объектный модуль, иногда язык ассемблера). Входной информацией для компилятора (исходный код) является описание алгоритма или программа на объектно-ориентированном языке, а на выходе компилятора – эквивалентное описание алгоритма на машинно-ориентированном языке (объектный код).

Компоновщик (также редактор связей или ли́нкер, от англ. link editor, linker) — это инструментальная программа, которая производит компоновку («линковку»): принимает на вход один или несколько объектных модулей и собирает из них исполняемый модуль.

GNU Compiler Collection (обычно используется сокращение GCC) – это набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU. GCC является свободным программным обеспечением, распространяется фондом свободного программного обеспечения (FSF) на условиях GNU GPL и GNU LGPL, и является ключевым компонентом GNU toolchain. Он используется как стандартный компилятор для свободных UNIX-подобных операционных систем.

Изначально названный GNU C Compile поддерживал только язык C. Позднее GCC был расширен для компиляции исходных кодов на таких языках программирования, как C++, Objective C, Java, Fortran, Ada и Go.

Программа gcc, запускаемая из командной строки, представляет собой надстройку над группой компиляторов. В зависимости от рас-

Библиотека ГГТУ им. П.О.Сухого

ширений имен файлов, передаваемых в качестве параметров, и дополнительных опций, gcc запускает необходимые препроцессоры, компиляторы, линкеры.

Файлы с расширением .cc рассматриваются, как файлы на языке C++, файлы с расширением .c как программы на языке C, а файлы с расширением .o считаются объектными.

Чтобы откомпилировать исходный код на языке C++, находящийся в файле F.cc, и создать объектный файл F.o, необходимо выполнить команду:

```
gcc -c F.cc
```

Здесь опция –c означает «только компиляция».

Чтобы скомпоновать один или несколько объектных файлов, полученных из исходного кода F1.o, F2.o, ... в единый исполняемый файл F, необходимо ввести команду:

```
gcc -o F F1.o F2.o
```

Опция -o задает имя исполняемого файла.

Опции компиляции

Среди множества опций компиляции и компоновки наиболее часто употребляются следующие:

Опция	Назначение
-c	Эта опция означает, что необходима только компиляция. Из исходных файлов программы создаются объектные файлы в виде name.o. Компоновка не производится.
-Dname=value	Определить имя name в компилируемой программе, как значение value. Эффект такой же, как наличие строки #define name value в начале программы. Часть =value может быть опущена, в этом случае значение по умолчанию равно 1.
-o file-name	Использовать file-name в качестве имени для создаваемого файла.

Библиотека ГГТУ им.П.О.Сухого

-lname	Использовать при компоновке библиотеку libname.so
-Llib-path -Iinclude-path	Добавить к стандартным каталогам поиска библиотек и заголовочных файлов пути lib-path и include-path соответственно.
-g	Поместить в объектный или исполняемый файл отладочную информацию для отладчика gdb. Опция должна быть указана и для компиляции, и для компоновки. В сочетании –g рекомендуется использовать опцию отключения оптимизации –O0 (см. ниже)
-MM	Вывести зависимости от заголовочных файлов, используемых в программе на C или C++, в формате, подходящем для утилиты make. Объектные или исполняемые файлы не создаются.
-pg	Поместить в объектный или исполняемый файл инструкции профилирования для генерации информации, используемой утилитой gprof. Опция должна быть указана и для компиляции, и для компоновки. Собранная с опцией -pg программа при запуске генерирует файл статистики. Программа gprof на основе этого файла создает расшифровку, указывающую время, потраченное на выполнение каждой функции.
-Wall	Вывод сообщений о всех предупреждениях или ошибках, возникающих во время компиляции программы.
-O1 -O2 -O3	Различные уровни оптимизации.
-O0	Не оптимизировать. Если вы используете многочисленные -O опции с номерами или без номеров уровня, действительной является последняя такая опция.
-I	Используется для добавления ваших собственных каталогов для поиска заголовочных файлов в процессе сборки
-L	Передается компоновщику. Используется для добав-

Библиотека БГУИР

	ления ваших собственных каталогов для поиска библиотек в процессе сборки.
-l	Передается компоновщику. Используется для добавления ваших собственных библиотек для поиска в процессе сборки.

Утилита make

Утилита make автоматизирует процесс преобразования файлов из одной формы в другую. Чаще всего это компиляция исходного кода в объектные файлы и последующая компоновка в исполняемые файлы или библиотеки.

Утилита использует специальные make-файлы, в которых указаны зависимости файлов друг от друга и правила для их удовлетворения. На основе информации о времени последнего изменения каждого файла утилита make определяет и запускает необходимые программы.

Утилита используется в следующем виде:

make [-f make-файл] [цель] …]

Файл ищется в текущем каталоге. Если ключ -f не указан, используется имя по умолчанию для make-файла – makefile. Утилита make открывает make-файл, считывает правила и выполняет команды, необходимые для создания указанной цели.

Стандартные цели для сборки дистрибутивов GNU:

- all — выполнить сборку пакета;
- install — установить пакет из дистрибутива (производит копирование исполняемых файлов, библиотек и документации в системные каталоги);
- uninstall — удалить пакет (производит удаление исполняемых файлов и библиотек из системных каталогов);
- clean — очистить дистрибутив (удалить из дистрибутива объектные и исполняемые файлы, созданные в процессе компиляции);
- distclean — очистить все созданные при компиляции файлы и все вспомогательные файлы, созданные утилитой configure в процессе настройки параметров компиляции дистрибутива.

Библиотека ГГТУ им. П.О.Сухого

По умолчанию make использует самую первую цель в make-файле.

Make-файл

Программа make выполняет команды согласно правилам, указанным в специальном файле. Этот файл называется make-файл (makefile, мейкфайл). Как правило, make-файл описывает, каким образом нужно компилировать и компоновать программу. Make-файл состоит из правил и переменных. Правила имеют следующий синтаксис:

```
цель1 цель2 ...: реквизит1 реквизит2 ...
	команда1
	команда2
	...
```

Правило представляет собой набор команд, выполнение которых приведёт к сборке файлов-целей из файлов-реквизитов.

Правило сообщает make, что файлы, получаемые в результате работы команд (цели) являются зависимыми от соответствующих файлов-реквизитов. Make никак не проверяет и не использует содержимое файлов-реквизитов, однако, указание списка файлов-реквизитов требуется только для того, чтобы make убедилась в наличии этих файлов перед началом выполнения команд и для отслеживания зависимостей между файлами.

Обычно цель представляет собой имя файла, который генерируется в результате работы указанных команд. Целью также может служить название некоторого действия, которое будет выполнено в результате выполнения команд (например, цель clean в make-файлах для компиляции программ обычно удаляет все файлы, созданные в процессе компиляции).

Строки, в которых записаны команды, должны начинаться с символа табуляции.

Рассмотрим несложную программу на языке Си. Пусть программа program состоит из пары файлов кода main.c и lib.c, а также из одного заголовочного файла defines.h, который подключен в обоих файлах кода. Поэтому для program необходимо из пар (main.c defines.h) и (lib.c defines.h) создать объектные файлы main.o и lib.o, а

Библиотека БГТУ им. П.О. Сухого

затем скомпоновать их в program. При сборке вручную требуется исполнить следующие команды:

```
cc -c main.c
cc -c lib.c
cc -o program main.o lib.o
```

Если в процессе разработки программы в файл defines.h будут внесены изменения, потребуется перекомпиляция обоих файлов и линковка, а если изменим lib.c, то повторную компиляцию main.o можно не выполнять.

Таким образом, для каждого файла, который мы должны получить в процессе компиляции, нужно указать, на основе каких файлов и с помощью какой команды он создается. Программа make на основе этих данных выполняет следующее:

– собирает из этой информации правильную последовательность команд для получения требуемых результирующих файлов;
– и инициирует создание требуемого файла только в случае, если такого файла не существует, или он старше, чем файлы, от которых он зависит.

Если при запуске make явно не указать цель, то будет обрабатываться первая цель в make-файле, имя которой не начинается с символа «.».

Для программы program достаточно написать следующий make-файл:

```
program: main.o lib.o
cc -o program main.o lib.o
main.o lib.o: defines.h
```

Стоит отметить ряд особенностей. В имени второй цели указаны два файла и для этой же цели не указана команда компиляции. Кроме того, нигде явно не указана зависимость объектных файлов от «*.c»-файлов. Дело в том, что программа make имеет предопределенные правила для получения файлов с определенными расширениями. Так, для цели-объектного файла (расширение «.o») при обнаружении соответствующего файла с расширением «.c» будет «cc -c» с указанием в параметрах этого «.c»-файла и всех файлов-зависимостей.

Синтаксис для определения переменных:

переменная = значение

Библиотека ГГТУ им. П. О. Сухого

Значением может являться произвольная последовательность символов, включая пробелы и обращения к значениям других переменных. С учетом сказанного, можно модифицировать наш make-файл следующим образом:

```
OBJ = main.o lib.o
program: $(OBJ)
	cc -o program $(OBJ)
$(OBJ): defines.h
```

3 Порядок выполнения работы

3.1 Создание папки для работы

3.1.1. Для выполнения лабораторных работ Вам необходимо создать на компьютере свою рабочую папку. Сделать это можно следующим образом.

На панели TotCom выберите диск F. Затем выберите папку TDS и раскройте ее. Внутри нее создайте новую папку (с помощью клавиши F7) с именем, соответствующим вашей фамилии (буквы обязательно латинские), например: Ivanov.

Примечание. При использовании в именах папок русских букв возможна неправильная работа исследуемых программ.

В дальнейшем Вы будете записывать и хранить в вашей папке все файлы в процессе выполнения лабораторных работ. Полный путь к ней:

F:\TDS\Ivanov

3.1.2. При выполнении лабораторных работ Вы будете создавать много различных файлов. Чтобы было легче ориентироваться в них, желательно для каждой лабораторной работы иметь свою отдельную папку. Допустим, что для выполнения лабораторной работы № 1 мы будем использовать папку с именем Lab1, которую необходимо предварительно создать. С этой целью откройте вашу рабочую папку (с именем, соответствующим вашей фамилии) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab1.

В дальнейшем Вы будете записывать и хранить в этой папке все файлы в процессе выполнения лабораторной работы №1. Полный путь к ней:

F:\TSD\Ivanov\Lab1

Библиотека ГГТУ им. П.О. Сухого

3.2 Сборка простейших однофайловых проектов

В папке Lab1 создадите файл hello.c при помощи простейшего текстового редактора (например, блокнота) со следующим содержанием:

```
#include <stdio.h>

int main (void)
{
	puts("Hello world!");
	getchar();
	return (0);
}
```

Сохраните файл. Затем необходимо запустить Qt 5.5 for Desktop (MinGW 4.9.2 32 bit). В появившемся окне командной строки необходимо перейти на диск E:

```
C:\Qt\Qt5.5.0\5.5\mingw492_32>E:
```

Затем зайти в папку с исходным файлом (hello.c):

```
F:\>cd f:\TSD\Ivanov\Lab1
```

И проверить наличие файла:

```
F: :\TSD\Ivanov\Lab1> dir
```

Если все выполнено правильно, то в списке вы увидите имя вашего файла (hello.c). Теперь, используя компилятор gcc, выполним сборку нашего проекта:

```
gcc -o hello.exe hello.c
```

Если в тексте программы присутствуют ошибки, то в командной строке вы увидите сообщения компилятора. Если все прошло удачно, то в списке файлов появится hello.exe. Запустите его и проверьте правильность выполнения.

Библиотека ГГТУ им. П.О. Сухого

3.3 Сборка многофайловых проектов

Рассмотрим, каким образом через командную строку собирать проекты, которые состоят из двух и более файлов исходного кода. Для этого нам понадобиться все то, что и в предыдущим разделе, а также утилита mingw32-make, которая аналогична утилите make.

Теперь в папке с проектом создадим следующе файлы:

- main.c (исходный текст основной программы)
- libTest.h (заголовочный файл)
- libTest.c (исходный текст библиотеки)
- Makefile (инструкции сборки проекта для утилиты make)

Содержание файлов:

main.c

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

int a;
int b;

int main()
{
 puts ("Start programm");

 scanf("%d",&a);
 scanf("%d",&b);
 int Summa = Sum (a,b);
 printf("Summa=%d\r\n",Summa);

 getchar();
 return(0);
}
```

libTest.h

```
#ifndef LIBTEST_H
#define LIBTEST_H
```


Библиотека ГГТУ им. П.О. Сухого

```
int Sum (int _a,int _b);

#endif
```

libTest.c

```
#include "libTest.h"

int Sum (int _a,int _b)
{
  return (_a+_b);
}
```

Makefile

```
CC=gcc
CFLAGS=-c  -static
LDFLAGS=
SOURCES= main.c libTest.c

OBJECTS=$(SOURCES:.c=.o)
EXECUTABLE= Test.exe

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
  $(CC) $(OBJECTS) -o $@ $(LDFLAGS)

clean:
  del $(OBJECTS) $(EXECUTABLE)
```

Опции:

- CC задает компилятор, который будет использоваться для сборки проекта
- CFLAGS задает опции компилятора
- SOURCES перечень исходных файлов, входящих в проект
- OBJECTS задает имя объектных файлов
- EXECUTABLE имя исполняемого файла, который будет получен после сборки проекта

Библиотека ГГТУ им. П.О. Сухого

Блок «clean:» предназначен для очистки папки нашего проекта от сборочных файлов. Для выполнения данного действия необходимо ввести команду mingw32-make clean.

Итак, после того, как все файлы созданы, можно приступить к процессу сборки. Для этого необходимо ввести команду:

```
mingw32-make
```

Так как MakeFile не указан в параметрах запуска утилиты, то она проведет поиска данного файла в текущей директории. И если данный файл присутствует, то по нему произведет сборку проекта.

Если в проекте нет ошибок, то в папке с проектом появятся следующие файлы:

- main.o
- libTest.o
- Test.exe

Запустите программу и проверьте ее работу.

3.4 Интегрированная среда разработки Qt Creator

Интегрированная среда Qt Creator позволяет разрабатывать и отлаживать программы на языках C и C++, как с использованием графических библиотек Qt, так и без них.

Запустим данную среду через соответствующий ярлык на рабочем столе и попробуем реализовать и отладить программу, описанную в предыдущем пункте.

Для создания нового проекта нажмем кнопку «Новый проект» и в появившемся диалоге выберем Проект без использования Qt -> Простой проект на C.

Библиотека ГГТУ им. П.О. Сухого

Рисунок 1 – Вид стартового окна Qt Creator

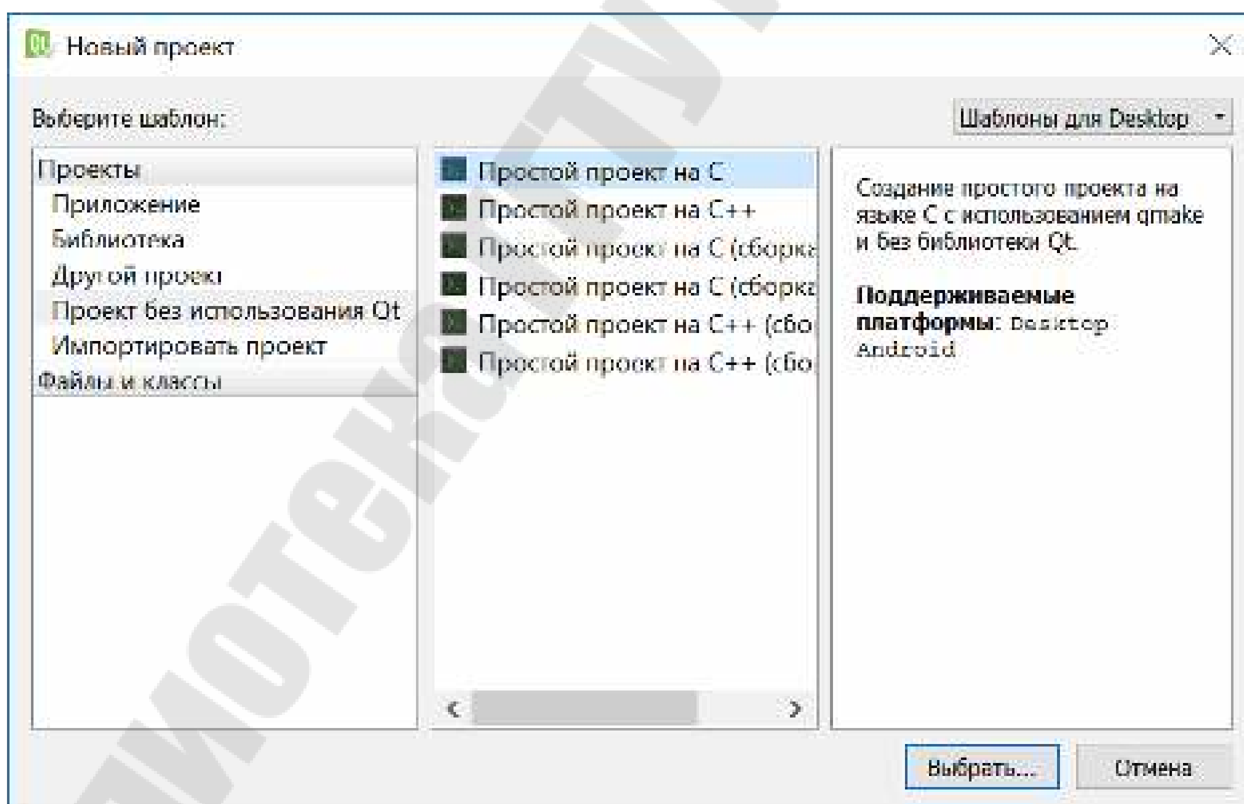

Рисунок 2 – Диалог выбора типа проекта

Библиотека БГТУ им. П.О. Сухого

После появится запрос с именем проекта и его расположением. Укажем имя проекта TestQtCreator и папку с нашими проектами.

После этого появится диалог с выбором комплекта сборки. Так как в списке только один вариант, то нажмем кнопку далее.

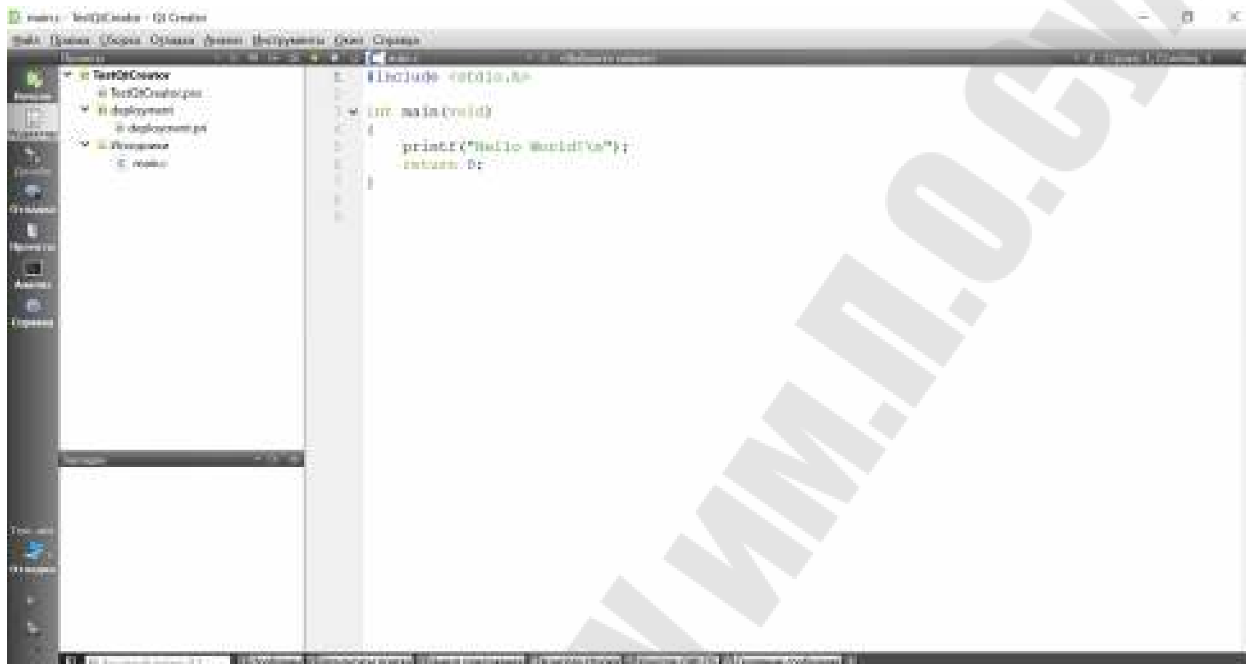

Рисунок 3 – Внещний вид основного окна после создания проекта

В левом верхнем углу располагается область менеджера проектов. В нем отображена структура нашего проекта. По умолчанию он состоит из следующих файлов:

- TestQtCreator.pro (Основной файл проекта)
- deployment.pri (Дополнительный файл проекта)
- main.c (Основной файл исходного кода проекта)

Затем нам необходимо создать библиотеку, состоящую из файлов:

- libTest.h (заголовочный файл)
- libTest.c (исходный текст библиотеки)

Для этого необходимо нажать правой кнопкой мыши по названию нашего проекта, затем в появившемся меню выбрать пункт «Добавить новый …». В появившемся диалоге создания нового файла необходимо последовательно выбрать C++ Source File (для libTest.c) и C++ Header File (для libTest.h).

Библиотека ГГТУ им.П.О.Сухого

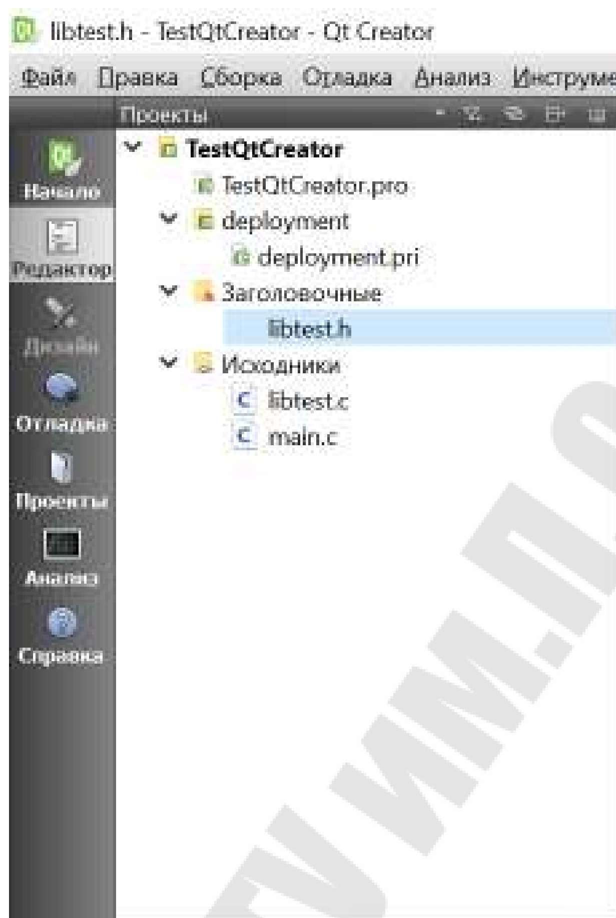

Рисунок 4 – Структура проекта после создания файлов библиотеки

Теперь из предыдущего пункта перенесем текст программы в соответствующие файлы. И запустим сборку проекта. Для этого необходимо нажать сочетание кнопок CTRL + R или слева внизу выбрать соответствующую кнопку. Если проект не содержит ошибок, то появится основное окно программы, такое же, как и в предыдущем пункте.

3.6 Задание для самостоятельной работы

Необходимо разработать программу, для вычисления значений следующих функций:

- f1(x) = 2x-5
- f2(x) = 2x^2 + 6x – 10
- f3(x) = x^2/(2x^3-6x)

Библиотека ГГТУ им.П.О.Сухого

Вычисления функций должны располагаться в библиотеке. Прототипы всех функций описаны в заголовочном файле. Также программа должна иметь текстовое меню с возможностью выбора вычисления одной из трех функций, а также вводом значения X с клавиатуры.

4 Содержание отчета

Наименование и цель работы. Краткая теоретическая часть, которая применялась вами при выполнении работы. Комментарии и снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

Контрольные вопросы

1. Что такое компиляция?
2. Что такое компоновка?
3. В чем отличия файла с исходными кодами от заголовочного?
4. Из каких основных файлов состоит проект на Qt Creator?
5. Как создать файлы пользовательской библиотеки в Qt Creator?
6. Что такое MakeFile и для чего он предназначен?

Лабораторная работа № 2

Работа с динамической памятью и указателями

1 Цель работы

Изучить принципы работы с динамической памятью. Изучить указатели и их практическое применение при разработке программного обеспечения

2 Основные теоретические сведения

Библиотека ГГТУ им. П.О.Сухого

Указатель – это переменная, содержащая адрес какой-либо переменной. Указатели широко применяются в языке С отчасти потому, что в некоторых случаях без них просто не обойтись, а отчасти потому, что программы с ними обычно короче и эффективнее. Указатели и массивы тесно связаны друг с другом. В данной лабораторной работе мы рассмотрим эту зависимость и покажем, как ею пользоваться.

Указатели и адреса

Начнем с того, что рассмотрим упрощенную схему организации памяти. Память типичного компьютера представляет собой массив последовательно пронумерованных (проадресованных) ячеек. Применительно к любому компьютеру верны следующие утверждения: однобайтовая ячейка может хранить значение типа char, двухбайтовые ячейки могут рассматриваться как целое типа short, а четырехбайтовые – как целые типа long. Указатель - это группа ячеек (как правило, две или четыре), в которых может храниться адрес. Так, если переменная с имеет тип char, а р – указатель на с, то это записывается следующим образом:

```
p = &c;
```

Унарный оператор & выдает адрес объекта, так что инструкция присваивает переменной р адрес ячейки, где находится переменная с. Оператор & применяется только к объектам, расположенным в памяти: к переменным и элементам массивов. Его операндом не может быть ни выражение, ни константа, ни регистровая переменная.

Унарный оператор * есть оператор косвенного доступа. Примененный к указателю он выдает объект, на который данный указатель указывает. Предположим, что x и y имеют тип int, а ip – укаэатель на int.

```
int x = 1, y = 2, z[10];


int *ip;          /* ip - указатель на int */


ip = &x;          /* теперь ip указывает на x */
y = *ip;          /* y теперь равен 1 */
```


Библиотека ГГТУ им.П.О.Сухого

```
 *ip = 0;           /* x теперь равен 0 */
 ip = &z[0];      /* ip теперь указывает на z[0] */
```

Объявления x, y и z нам уже знакомы. Объявление указателя ip int *ip; мы стремились сделать мнемоничным - оно гласит: "выражение *ip имеет тип int". Синтаксис объявления переменной "подстраивается" под синтаксис выражений, в которых эта переменная может встретиться. Указанный принцип применим и в объявлениях функций. Например, запись

double *dp, atof (char *);

означает, что выражения *dp и atof(s) имеют тип double, а аргумент функции atof есть указатель на char.

Вы, наверное, заметили, что указателю разрешено указывать только на объекты определенного типа.

Если ip указывает на x целочисленного типа, то *ip можно использовать в любом месте, где допустимо применение x, например, *ip = *ip + 10 увеличивает *ip на 10.

Унарные операторы * и & имеют более высокий приоритет, чем арифметические операторы, так что присваивание y = *ip + 1 берет то, на что указывает ip, и добавляет к нему 1, а результат присваивает переменной y. Аналогично *ip += 1 увеличивает на единицу то, на что указывает ip. Те же действия выполняют ++*ip и (*ip)++.

В последней записи скобки необходимы, поскольку если их не будет, увеличится значение самого указателя, а не то, на что он указывает. Это обусловлено тем, что унарные операторы * и ++ имеют одинаковый приоритет и порядок выполнения справа налево.

И наконец, так как указатели сами являются переменными, в тексте они могут встречаться и без оператора косвенного доступа. Например, если iq есть другой указатель на int, то iq = ip копирует содержимое ip в iq, чтобы ip и iq указывали на один и тот же объект.

Указатели и аргументы функций

Поскольку в языке C функции в качестве своих аргументов получают значения параметров, нет прямой возможности, находясь в вызванной функции, изменить переменную вызывающей функции. В программе сортировки нам понадобилась функция swap, меняющая местами два неупорядоченных элемента. Однако недостаточно написать

swap(a, b);   где функция swap определена следующим образом:

Библиотека ГГТУ им.П.О.Сухого

```
void swap(int x, int y)        /* НЕВЕРНО */
{
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

Поскольку функция swap получает лишь копии переменных a и b, она не может повлиять на переменные a и b той программы, которая к ней обратилась. Чтобы получить желаемый эффект, вызывающей программе надо передать указатели на те значения, которые должны быть изменены: swap(&a, &b);

Так как оператор & получает адрес переменной, &a есть указатель на a. В самой же функции swap параметры должны быть объявлены как указатели, при этом доступ к значениям параметров будет осуществляться косвенно.

```
void swap(int *px, int *py)       /* перестановка *px и *py */
{
   int temp;
   temp = *px;
   *px = *py;
   *py = temp;
}
```

Аргументы-указатели позволяют функции осуществлять доступ к объектам вызвавшей ее программы и дают возможность изменить эти объекты.

Рассмотрим, например, функцию getint, которая осуществляет ввод в свободном формате одного целого числа и его перевод из текстового представления в значение типа int. Функция getint должна возвращать значение полученного числа или сигнализировать значением EOF о конце файла, если входной поток исчерпан. Эти значения должны возвращаться по разным каналам, так как нельзя рассчитывать на то, что полученное в результате перевода число никогда не совпадет с EOF.

Одно из решений состоит в том, чтобы getint выдавала характеристику состояния файла (исчерпан или не исчерпан) в качестве результата, а значение самого числа помещала согласно указателю, пе-

Библиотека ГГТУ им. П. О. Сухого

реданному ей в виде аргумента. Показанный ниже цикл заполняет некоторый массив целыми числами, полученными с помощью getint.

```
int n, array[SIZE], getint (int *);
for (n = 0; n < SIZE && getint (&array[n]) != EOF; n++) ;
```

Результат каждого очередного обращения к getint посылается в array[n], и n увеличивается на единицу. Заметим, и это существенно, что функции getint передается адрес элемента array[n]. Если этого не сделать, у getint не будет способа вернуть в вызывающую программу переведенное целое число.

В предлагаемом нами варианте функция getint возвращает EOF по концу файла: нуль, если следующие вводимые символы не представляют собою числа, и положительное значение, если введенные символы представляют собой число.

```
#include <ctype.h>
int getch (void);
void ungetch (int);
    /* getint: читает следующее целое из ввода в *pn */
int getint(int *pn)
{
   int c, sign;
   while (isspace(c = getch()))


      ; /* пропуск символов-разделителей */





   if(!isdigit(c) && c != EOF && c != '+' && c != '-') {


      ungetch (c); /* не число */
      return 0;
   }
```


Библиотека ГГТУ им. П.О.Сухого

```
   sign =(c =='-') ? -1 : 1;

   if (c == '+' || c == '-')

      c = getch();
   for (*pn = 0; isdigit(c); c = getch())

      *pn = 10 * *pn + (c -'0');     *pn *= sign;     if (c != EOF)

      ungetch(c);
   return c;
}
```

Везде в getint под *pn подразумевается обычная переменная типа int. Функция ungetch вместе с getch включена в программу, чтобы обеспечить возможность отослать назад лишний прочитанный символ.

Указатели и массивы

В языке С существует связь между указателями и массивами, и связь эта настолько тесная, что эти средства лучше рассматривать вместе. Любой доступ к элементу массива, осуществляемый операцией индексирования, может быть выполнен с помощью указателя. Вариант с указателями в общем случае работает быстрее, но разобраться в нем, особенно непосвященному, довольно трудно. Объявление
int a[10]; определяет массив a размера 10, т. е. блок из 10 последовательных объектов с именами a[0], a[1], ..., a[9].

Запись a[i] отсылает нас к i-му элементу массива. Если pa есть указатель на int, т. е. объявлен как int *pa; то в результате присваивания pa = &a[0]; pa будет указывать на нулевой элемент a, иначе говоря, pa будет содержать адрес элемента a[0]. Теперь присваивание x = *pa; будет копировать содержимое a[0] в x.

Библиотека ГГТУ им.П.О.Сухого

Если pa указывает на некоторый элемент массива, то pa+1 по определению указывает на следующий элемент, pa+i - на i-й элемент после pa, а pa-i - на i-й элемент перед pa. Таким образом, если pa указывает на a[0], то *(pa+1) есть содержимое a[1], a+i - адрес a[i], а *(pa+i) - содержимое a[i].

Сделанные замечания верны безотносительно к типу и размеру элементов массива a. Смысл слов "добавить 1 к указателю", как и смысл любой арифметики с указателями, состоит в том, чтобы pa+1 указывал на следующий объект, а pa+i - на i-й после pa.

Между индексированием и арифметикой с указателями существует очень тесная связь. По определению значение переменной или выражения типа массив есть адрес нулевого элемента массива. После присваивания pa = &a[0]; pa и a имеют одно и то же значение. Поскольку имя массива является синонимом расположения его начального элемента, присваивание pa=&a[0] можно также записать в следующем виде: pa = a;

Еще более удивительно (по крайней мере на первый взгляд) то, что a[i] можно записать как *(a+i). Вычисляя a[i], Си сразу преобразует его в *(a+i); указанные две формы записи эквивалентны. Из этого следует, что полученные в результате применения оператора & записи &a[i] и a+i также будут эквивалентными, т. е. и в том и в другом случае это адрес i-го элемента после a. С другой стороны, если pa - указатель, то его можно использовать с индексом, т. е. запись pa[i] эквивалентна записи *(pa+i). Короче говоря, элемент массива можно изображать как в виде указателя со смещением, так и в виде имени массива с индексом.

Между именем массива и указателем, выступающим в роли имени массива, существует одно различие. Указатель – это переменная, поэтому можно написать pa=a или pa++. Но имя массива не является переменной, и записи вроде a=pa или a++ не допускаются.

Если имя массива передается функции, то последняя получает в качестве аргумента адрес его начального элемента. Внутри вызываемой функции этот аргумент является локальной переменной, содержащей адрес. Мы можем воспользоваться отмеченным фактом и написать еще одну версию функции strlen, вычисляющей длину строки.

```
/* strlen: возвращает длину строки */
int strlen(char *s)
```


Библиотека БГУИР

```
{
   int n;
   for (n = 0; *s != '\0'; s++)
      n++;
   return n;
}
```

Так как переменная s – указатель, к ней применима операция инкремента. s++ не оказывает никакого влияния на строку символов функции, которая обратилась к strlen. Просто увеличивается на 1 некоторая копия указателя, находящаяся в личном пользовании функции strlen. Это значит, что все вызовы, такие как:

```
strlen("Здравствуй, мир"); /* строковая константа */
strlen(array);            /* char array[100]; */
strlen(ptr);             /* char *ptr; */ правомерны.
```

Формальные параметры char s[]; и char *s; в определении функции эквивалентны. Мы отдаем предпочтение последнему варианту, поскольку он более явно сообщает, что s есть указатель. Если функции в качестве аргумента передается имя массива, то она может рассматривать его так, как ей удобно – либо как имя массива, либо как указатель, и поступать с ним соответственно. Она может даже использовать оба вида записи, если это покажется уместным и понятным.

Функции можно передать часть массива, для этого аргумент должен указывать на начало подмассива. Например, если a – массив, то в записях

f(&a[2]) или f(a+2)

функции f передается адрес подмассива, начинающегося с элемента a[2]. Внутри функции f описание параметров может выглядеть как f(int arr[]) {...} или f(int *arr) {...} .

Следовательно, для f тот факт, что параметр указывает на часть массива, а не на весь массив, не имеет значения.

Если есть уверенность, что элементы массива существуют, то возможно индексирование и в "обратную" сторону по отношению к нулевому элементу; выражения p[-1], p[-2] и т.д. не противоречат синтаксису языка и обращаются к элементам, стоящим непосредст-

Библиотека ГГТУ им. П.О.Сухого

венно перед p[0]. Разумеется, нельзя "выходить" за границы массива и тем самым обращаться к несуществующим объектам.

3 Порядок выполнения работы

3.1 Создание папки для работы

При выполнении лабораторной работы № 2 вы будете использовать папку с именем Lab2, которую необходимо предварительно создать.

С этой целью откройте вашу рабочую папку (с вашей фамилией) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab2.

В дальнейшем вы будете записывать и хранить в этой папке все файлы при выполнении лабораторной работы № 2. Полный путь к этой папке будет такой:

F:\TDS\Ivanov\Lab2

3.2 Указатели на простые типы данных

Рассмотрим простейшую программу с указателем на переменную типа int:

```
#include <iostream>

using namespace std;

int main()
{
   cout << "Start programm" << endl;

   int a =0;
   int *b=&a;


   cout << "sizeof(a)=" << sizeof(a) << endl;
   cout << "sizeof(b)=" << sizeof(b) << endl;

   cout << "a=" << a << endl;
```


Библиотека ГГТУ им. П.О.Сухого

```
    cout << "b=" << b << endl;
    cout << "*b=" << *b << endl;

    cout << "Increment a" << endl;
    a++;

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "*b=" << *b << endl;

    cout << "Increment *b" << endl;
    (*b)++;

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "*b=" << *b << endl;

    cout << "Increment b" << endl;
    b++;

    cout << "a=" << a << endl;
    cout << "b=" << b << endl;
    cout << "*b=" << *b << endl;

    return 0;
}
```

Сначала подключаем библиотеку iostream для использования потокового ввода/вывода. Опция using namespace std указывает компилятору что по умолчанию используются функции из пространства имен std. Если данная опция не будет указана, то команда вывода информации через cout будет выглядеть следующим образом:

```
std::cout << "a=" << a << std::endl;
```

Таким образом, когда в программе используются функции только из пространства std, наиболее удобно применять данную опцию для уменьшения исходного кода.

Библиотека ГГТУ им.П.О.Сухого

Объявляем переменную a типа int и указатель b на тип int. Переменной a присвоим значение 0, а указателю b – адрес переменной a:

```
int a =0;
int *b=&a;
```

С помощью функции sizeof() определим объем, который занимают в памяти наша переменная и указатель. Так как компилятор предназначен для 32-х битной платформы, тип int, а также указатель на него будут занимать в оперативной памяти объем равный 4 байта.

```
cout << "sizeof(a)=" << sizeof(a) << endl;
cout << "sizeof(b)=" << sizeof(b) << endl;
```

Теперь проведем несколько опытов с переменной и указателем. Для начала выведем на экран значение нашей переменной, значение указателя, а также значение объекта, на который указывает указатель:

```
cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "*b=" << *b << endl;
```

В первом и третьем случае на экране будет отображено значение нашей переменной, которое равно 0. А вот во втором случае будет выведен адрес, который в памяти занимает наша переменная.

Теперь проведем инкремент переменной a и снова выведем значения на экран:

```
a++;

cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "*b=" << *b << endl;
```

Значение переменной a после операции инкремента увеличится на 1. А адрес, находящийся в указателе, останется неизменный. Теперь проведем инкремент объекта, на который указывает b:

Библиотека ГГТУ им. П. О. Сухого

```
(*b)++;
cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "*b=" << *b << endl;
```

На экране мы увидим те же изменения, как и в первом случае – значение переменной увеличится на 1, а адрес останется неизменным. А вот теперь выполним инкремент самого указателя:

```
b++;
cout << "a=" << a << endl;
cout << "b=" << b << endl;
cout << "*b=" << *b << endl;
```

После выполнения данных действий переменная a останется неизменной, а вот адрес сместиться вправо на 4 байта, и значение объекта, на который указывает указатель b, не будет совпадать со значением переменной a.

3.3 Указатель как параметр функции

Указатели можно применять в качестве входного параметра функции. Рассмотрим небольшую программу:

```
#include <iostream>

using namespace std;
int a;

int pointers(int *b,int c);

int main()
{
   cout << "Start programm" << endl;
   a=5;
   int e=8;
   cout << "a=" << a << endl;
   cout << "e=" << e << endl;
```


Библиотека ГГТУ им.П.О.Сухого

```
   int d = pointers(&a,e);

   cout << "a=" << a << endl;
   cout << "e=" << e << endl;
   cout << "d=" << d << endl;

   return 0;
}

int pointers(int *b,int c)
{
   a++;
   (*b)++;

   int output;

   output = c + 10 ;

   return output;
}
```

Вначале объявим глобальную переменную a, которая будет доступна всем функциям в текущем файле исходных кодов. Затем присвоим ей значение 5, а также объявим локальную переменную e для функции main.

Пользовательская функция pointers в качестве входных параметров принимает указатель на тип int, а также переменную того же типа.

```
int pointers(int *b, int c);
```

В данной функции сначала производится инкремент глобальной переменной, а затем инкремент объекта, на который указывает указатель b, а также возвращает c + 10.

После выполнения программы значение a увеличиться на 2, так как дважды производится операция инкремента. Первый раз как инкремент глобальной переменной в функции, а второй раз как инкремент объекта, на который ссылается указатель.

3.4 Указатель на массив

Библиотека ГГТУ им.П.О.Сухого

Работа с массивом через указатель заключается в том, что объявляется указатель на тип элемента массива. Затем ему присваивается адрес первого элемента, а затем за счет инкремента адреса выполняется перебор элементов.

```
#include <iostream>

using namespace std;

void MasIncrement(int *input,int count);

int main()
{
   cout << "Start program" << endl;

   int Mas[10] = {0,1,2,3,4,5,6,7,8,9};

   cout<<"Massive:"<< endl;
   for (int i=0;i<10;i++) cout<<"Mas["<<i<<"]="<<Mas[i]<<" ";
   cout<< endl;

   MasIncrement(&Mas[0],10);

   cout<<"Massive New:"<< endl;
   for (int i=0;i<10;i++) cout<<"Mas["<<i<<"]="<<Mas[i]<<" ";
   cout<< endl;

   return 0;
}

void MasIncrement(int *input,int count)
{
   for (int i=0;i<count;i++) (*(input+i))++;
}
```

В функции MasIncrement() выполняется последовательный перебор элементов массива и инкремент каждого элемента.

Библиотека ГГТУ им. П.О.Сухого

3.5 Приведение типа указателя

Приведение указателей к типам char * или void * - это гарантированное преобразование. При приведении обратно будет получен тот же указатель. При приведении к другим типам указателей и обратно обратный указатель может быть отличным от исходного.

Рассмотрим пример программы, в которой мы выполним сериализацию и десериализацию пользовательской структуры.

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

using namespace std;

#pragma pack(push, 1)
struct TestUserStruct
{
   int a;
   int b;
   float c;
};
#pragma pack(pop)

int main()
{
   cout << "Start programm" << endl;
   TestUserStruct A;

   A.a=5;
   A.b=21;
   A.c=7.5;

   cout<<"A.a="<<A.a<<" A.b="<<A.b<<" A.c="<<A.c<<endl;
   cout<<"sizeof(A)="<<sizeof(A)<<endl;

   char Mas[sizeof(A)];

   //Сериализация сруктуры A
```


Библиотека ГГТУ им.П.О.Сухого

```
  char *ptr = (char*)&A;
  for (int i=0;i<sizeof(A);i++) Mas[i]=*(ptr+i);

  cout<<"Massiv:"<<endl;
  for (int i=0;i<sizeof(A);i++) printf ("Mas[%d]=%d ",i,Mas[i]);

  //Десериализация структуры B
  TestUserStruct B;

  char *e = (char*)&B;
  for (int i=0;i<sizeof(B);i++)
  {
    *(e+i)=Mas[i];
  }

  cout<<endl;
  cout<<"B.a="<<B.a<<" B.b="<<B.b<<" B.c="<<B.c<<endl;

  return 0;
}
```

В данной программе создается пользовательская структура, состоящая из переменных простых типов данных. Далее производится ее сериализация в массив Mas. Затем производиться десериализация данного массива во вторую копию структуры B. Если все выполнено правильно, то структуры A и B будут содержать идентичные данные.

3.6 Задания для самостоятельной работы

Задание 1.
Реализовать в пользовательской библиотеке набор функций, которые осуществляют основные действия над массивами. А именно ввод, вывод, сортировка по возрастанию, сортировка по убыванию. Массив должен передаваться через указатели. В программе должно присутствовать меню, с возможностью выбора необходимого действия.

Задание 2.
Реализовать программу, которая производит сериализацию заданной структуры, сохранение ее в файл. А также загрузку массива

Библиотека ГГТУ им.П.О.Сухого

данных из файла и десериализацию. В программе должно присутствовать меню, в котором выбирается необходимое действие: загрузка информации из файла либо сохранение.

4 Содержание отчета

Наименование и цель работы. Краткая теоретическая часть, которая применялась вами при выполнении работы. Комментарии и снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

Контрольные вопросы

1. Что такое указатель?
2. Какие унарные операции существуют над указателями?
3. Как применить указатель в качестве входного параметра функции?
4. Какие операции над указателями возможны?

Лабораторная работа № 3

Использование классов и встроенных методов

1 Цель работы

Изучить принципы использования классов, работу с методами классов. А также технологии доступа к объекту через указатель и динамическое выделение памяти.

2 Основные теоретические сведения

Центральным понятием ООП является класс. Класс используется для описания типа, на основе которого создаются объекты (переменные типа класс).

Библиотека ГГТУ им. П. О. Сухого

Класс, как и любой тип данных, характеризуется множеством значений, которые могут принимать объекты класса, и множеством функций, задающих операции над объектами. Пример объявления класса:

```
class  Имя_класса  { определение_членов_класса };
```

Члены класса можно разделить на информационные члены и функции-члены (методы) класса. Информационные члены описывают внутреннюю структуру информации, хранящейся в объекте, который создается на основе класса. Методы класса описывают алгоритмы обработки этой информации.

Данные, хранящиеся в информационных членах, описывают состояние объекта, созданного на основе класса. Состояние объекта изменяется на основе изменения хранящихся данных с помощью методов класса. Алгоритмы, заложенные в реализации методов класса, определяют поведение объекта, то есть реагирование объекта на поступающие внешние воздействия в виде входных данных.

Принцип инкапсуляции обеспечивается вводом в класс областей доступа:

− private (закрытый, доступный только собственным методам)
− public (открытый, доступный любым функциям)
− protected (защищенный, доступный только собственным методам и методам производных классов)

Члены класса, находящиеся в закрытой области (private), недоступны для использования со стороны внешнего кода. Напротив, члены класса, находящиеся в открытой секции (public), доступны для использования со стороны внешнего кода. При описании класса каждый член класса помещается в одну из перечисленных выше областей доступа следующим образом:

```
class  Имя_класса  {
private:
определение_закрытых_членов_класса
public:
            определение_открытых_членов_класса
      protected:
            определение_защищенных_членов_класса
```

```
 …
};
```

Порядок следования областей доступа и их количество в классе произвольны.

Служебное слово, определяющее первую область доступа, может отсутствовать. По умолчанию эта область считается private.

В закрытую (private) область обычно помещаются информационные члены, а в открытую (public) область – методы класса, реализующие интерфейс объектов класса с внешней средой. Если какой-либо метод имеет вспомогательное значение для других методов класса, являясь подпрограммой для них, то его также следует поместить в закрытую область. Это обеспечивает логическую целостность информации.

После описания класса его имя можно использовать для описания объектов этого типа.

Доступ к информационным членам и методам объекта, описанным в открытой секции, осуществляется через объект или ссылку на объект с помощью операции выбора члена класса ‘.’.

Если работа с объектом выполняется с помощью указателя на объект, то доступ к соответствующим членам класса осуществляется на основе указателя на член класса‘->’:

```
class X {  public:
            char c;
int f(){…}
};

int main () {
X x1;
X & x2 = x1; X * p = & x1; int i, j, k; x1.c = ‘*’; i = x1.f(); x1.c = '+'; j = x2.f(); x1.c = ‘#’; k = p -> f();
…
}
```

Объекты класса можно определять совместно с описанием класса:

```
class Y {…} y1, y2;
```

Библиотека ГГТУ им.П.О.Сухого

Но правило «хорошего тона» требует описывать структуру класса в .h файле, а непосредственно тело методов – в файле .cpp.

Каждый метод класса должен быть определен в программе. Определить метод класса можно либо непосредственно в классе (если тело метода не слишком сложно и громоздко), либо вынести определение вне класса, а в классе только объявить соответствующий метод, указав его прототип.

При определении метода класса вне класса для указания области видимости соответствующего имени метода используется операции ‘::’:

Пример:

```
class x {       int ia1;    public:
     x(){ia1 = 0;}       int func1();
};

int x::func1(){ … return ia1; }
```

Это позволяет повысить наглядность текста, особенно, в случае значительного объема кода в методах. При определении метода вне класса с использованием операции ‘::’ прототипы объявления и определения функции должны совпадать.

В классах С++ неявно введен специальный указатель this – указатель на текущий объект. Каждый метод класса при обращении к нему получает данный указатель в качестве неявного параметра. Через него методы класса могут получить доступ к другим членам класса.

Указатель this можно рассматривать как локальную константу, имеющую тип X*, если X – имя описываемого класса. Нет необходимости использовать его явно. Он используется явно, например, в том случае, когда выходным значением для метода является текущий объект.

Данный указатель, как и другие указатели, может быть разыменован.

При передаче возвращаемого значения метода класса в виде ссылки на текущий объект используется разыменованный указатель this, так как ссылка, как уже было указано, инициализируется непосредственным значением.

Библиотека ГГТУ им.П.О.Сухого

Пример:

```
class x {
  . . .   public:
     x& f(. . .){
 . . .
 return *this;
 }
};
```

3 Порядок выполнения работы

3.1 Создание папки для работы

При выполнении лабораторной работы № 3 вы будете использовать папку с именем Lab3, которую необходимо предварительно создать.

С этой целью откройте вашу рабочую папку (с вашей фамилией) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab3.

В дальнейшем вы будете записывать и хранить в этой папке все файлы при выполнении лабораторной работы № 3. Полный путь к этой папке будет такой:

F:\TDS\Ivanov\Lab3

3.2 Статическое объявление объекта

Рассмотрим простейший пример класса. Который имеет две переменные целого типа в качестве информационного члена класса и 4 метода для работы с ними.

Файл main.cpp

```
#include <iostream>
#include "libclass01.h"

using namespace std;
```


Библиотека ГГТУ им.П.О.Сухого

```
int main()
{
   cout << "Start programm" << endl;

   Numbers ObjectNumbers;

   int A;
   cout <<"A=";
   cin >> A;
   ObjectNumbers.setA(A);

   int B;
   cout << "B=";
   cin >> B;
   ObjectNumbers.setB(B);

   cout << "Sum=" << ObjectNumbers.getSum() << endl;
   cout << "Difference=" << ObjectNumbers.getDifference() << endl;
   return 0;
}
```

Файл libclass01.h

```
#ifndef LIBCLASS01
#define LIBCLASS01

class Numbers
{
private:
   int A;
   int B;

public:
   void setA (int input);
   void setB (int input);
   int getSum(void);
   int getDifference(void);
```

Библиотека ГГТУ им. П.О. Сухого

```
};

#endif // LIBCLASS01
```

Файл libclass01.cpp

```
#include "libclass01.h"

void Numbers::setA (int input)
{
   A=input;
}

void Numbers::setB (int input)
{
   B=input;
}

int Numbers::getSum(void)
{
   int output = A+B;
   return output;
}

int Numbers::getDifference(void)
{
   int output = A-B;
   return output;
}
```

Переменные int A и int B являются информационными членами класса. Методы setA и setB предназначены для задания значения соответствующим информационным членам класса.

Методы getSum и getDifference производят обработку информационных членов класса. Они возвращают сумму (A+B) и разность (A-B) соответственно.

3.3 Динамическое объявление объекта

Библиотека ГГТУ им.П.О.Сухого

Рассмотрим небольшую программу, которая имеет класс для обработки массива целых чисел:

Файл main.cpp

```
#include <iostream>
#include "libclass02.h"

using namespace std;

int main()
{
   cout << "Start programm" << endl;



   int Count=0;
   cout << "Count=";
   cin >> Count;

   Array *Massive = new Array[Count];

   int A[Count];

   for (int i=0;i<Count;i++)
   {
      cout <<"A["<< i <<"]=";
      cin >>A[i];
   }

   Massive->setArray(A,Count);
   cout << "Source array:" <<endl;
   Massive->printArray();
   Massive->workArray();
   cout << "New array:" <<endl;
   Massive->printArray();

   delete Massive;
```


Библиотека ГГТУ им.П.О.Сухого

```
    return 0;
}
```

Файл libclass02.h

```
#ifndef LIBCLASS02
#define LIBCLASS02

#include <iostream>

class Array
{
private:
    int *Mas=0;
    int Count_=0;
    int indexOf (int input);

public:
    void setArray(int *input,int Count);
    void printArray(void);
    void workArray(void);

};

#endif // LIBCLASS02
```

Файл libclass02.cpp

```
#include "libclass02.h"

int Array::indexOf (int input)
{
    if ((Count_!=0)&&(Mas!=0))
    {
        for (int i=0;i<Count_;i++)
            if (Mas[i]==input)
            {
                return i;
            }
```


Библиотека ГГТУ им.П.О.Сухого

```
   } else return -1;

   return -1;
}

void Array::setArray(int *input, int Count)
{
   Mas = new int[Count];
   Count_=Count;
   for (int i=0;i<Count;i++) Mas[i] = input[i];
}

void Array::printArray(void)
{
   if (Count_!=0)
      for (int i=0;i<Count_;i++)
      {
         std::cout << "Mas[" << i <<"]=" << Mas[i] << std::endl;
      }
}

void Array::workArray(void)
{
   int Pos = 1;
   while (Pos>0)
   {
      Pos = indexOf(0);
      Mas[Pos]=500;
   }
}
```

В данном случае информационным членом класса у нас является массив, который динамически формируется в оперативной памяти. Метод setArray в качестве входных параметров принимает указатель на созданный в основной программе массив, а также количество элементов в данном массиве. При его вызове в памяти формируется динамический массив. Количество элементов данного массива совпадает с количеством элементов массива, на который ссылается указатель. Затем данные из внешнего массива переносятся во внутренний.

Библиотека ГГТУ им.П.О.Сухого

Метод workArray производит обработку внутреннего массива. Каждый элемент массив, который равен 0, принимает значение 500. Внутренний метод indexOf осуществляет поиск элементов с заданным значением, и возвращает номер элемента. Если таковой элемент не найден, то возвращается -1.

Метод printArray осуществляет вывод текущих значений внутреннего массива на экран.

После выполнения необходимых действий объект удаляется из оперативной памяти.

3.4 Задание для самостоятельной работы

Разработать класс, который производит обработку двумерного массива. Дана квадратная матрица n*n (размеры матрицы вводятся с клавиатуры). Она заполняется значениями с клавиатуры. Затем с помощью метода setMatrix передается в класс. С помощью метода printMatrix, данная матрица выводится на экран в виде таблицы. Метод workMatrix производит обработку двумерного массива по заданному алгоритму (алгоритм зависит от варианта).

Объект должен быть создан динамически и после выполнения его необходимо удалить из оперативной памяти.

Пояснение:

Квадратная матрица делиться на 4 части. С помощью двух диагоналей: главной и побочной, как показано на рисунке 1.

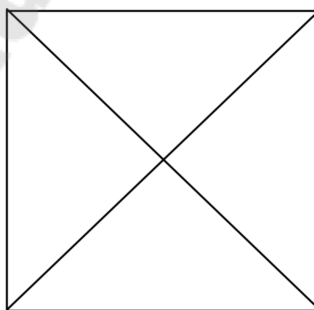

Рисунок 1 – Деление матрицы на 4 части с помощью диагоналей

В результате такого деления образуется 4 треугольника. Выбор треугольника зависит от порядкового номера студента по журналу. Необходимо взять ваш текущий номер и получить остаток от деления

Библиотека ГГТУ им. П.О.Сухого

на 4. Например, номер по журналу 13, разделим его 4 и возьмем остаток от деления, равный 1.

Таблица 1 – Выбор треугольника

Остаток от деления	Треугольник
0	Левый
1	Правый
2	Верхний
3	Нижний

Затем в выбранном треугольнике необходимо обработать все элементы, лежащие в нем и на его границах. Программу оптимизировать по выполнению. Т.е. необходимо перебрать в цикле только те элементы, которые принадлежат вашему треугольнику и его границам.

Алгоритм обработки выбирается в зависимости от порядкового номера студента по журналу по таблице 2.

Библиотека ГГТУ им. П.О. Сухого

Таблица 2 – Выбор алгоритма обработки треугольника

№	Алгоритм обработки
1	Отсортировать элементы по возрастанию
2	Отсортировать элементы по убыванию
3	Определить минимальный элемент
4	Определить максимальный элемент
5	Определить количество отрицательных элементов
6	Определить количество положительных элементов
7	Определить количество нулевых элементов
8	Посчитать сумму минимального и максимального элемента
9	Посчитать произведение максимального и минимального элемента
10	Найти остаток от деления максимального на минимальный элемент
11	Посчитать среднее арифметическое положительных элементов
12	Посчитать среднее арифметическое отрицательных элементов
13	Посчитать произведение положительных элементов
14	Определить максимальный элемент, принадлежащий промежутку [-10,5]
15	Определить минимальный элемент, принадлежащий промежутку [-100,50]
16	Определить количество элементов, кратных 5 и принадлежащих промежутку [4,13]
17	Поменять местами максимальный и минимальный элементы
18	Определить упорядочены ли элементы, если да, то каким образом
19	Заменить максимальный элемент средним арифметическим всех элементов

4 Содержание отчета

Наименование и цель работы. Краткая теоретическая часть, которая применялась вами при выполнении работы. Комментарии и снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

Библиотека ГГТУ им. П.О. Сухого

Контрольные вопросы

1. Что такое класс?
2. Что такое объект?
3. Как задаются права доступа к членам и методам класса?
4. В чем отличие статического и динамического создания объекта?

Лабораторная работа № 4

Использование методов классов: конструктора, деструктора и дружественной функции

1 Цель работы

Изучить принципы использования конструктора и деструктора класса, а также технологию применения дружественной (friend) функции.

2 Основные теоретические сведения

Конструкторы и деструкторы являются специальными методами класса. Конструкторы вызываются при создании объектов класса и отведении памяти под них. Деструкторы вызываются при уничтожении объектов и освобождении отведенной для них памяти.

В большинстве случаев конструкторы и деструкторы вызываются автоматически (неявно) соответственно при описании объекта (в момент отведения памяти под него) и при уничтожении объекта. Конструктор (как и деструктор) может вызываться и явно, например, при создании объекта в динамической области памяти с помощью операции new.

Так как конструкторы и деструкторы неявно входят в интерфейс объекта, их следует располагать в открытой области класса.

Отличия и особенности описания конструктора от обычной функции:

Библиотека ГГТУ им. П.О. Сухого

– Имя конструктора совпадает с именем класса
– При описании конструктора не указывается тип возвращаемого значения

1. Конструкторы можно классифицировать разными способами:

по наличию параметров:
– без параметров,
– с параметрами;

2. по количеству и типу параметров:

– конструктор по умолчанию,
– конструктор преобразования,
– конструктор копирования,
– конструктор с двумя и более параметрами.

В классе может быть несколько конструкторов. В соответствии с правилами языка C++ все они имеют одно имя, совпадающее с именем класса, что является одним из проявлений статического полиморфизма. Компилятор выбирает тот конструктор, который в зависимости от ситуации, в которой происходит создание объекта, удовлетворяет ей по количеству и типам параметров. Естественным ограничением является то, что в классе не может быть двух конструкторов с одинаковым набором параметров.

Деструкторы применяются для корректного уничтожения объектов. Часто процесс уничтожения объектов включает в себя действия по освобождению выделенной для них по операциям new памяти.

Имя деструктора: ~имя_класса

– У деструкторов нет параметров и возвращаемого значения.
– В отличие от конструкторов деструктор в классе может быть только один.

Конструктор по умолчанию

Конструктор без параметров называется конструктором по умолчанию. Если для создания объекта не требуется каких-либо параметров, то используется конструктор по умолчанию. При описании таких объектов после имени класса указывается только идентифика-

Библиотека ГГТУ им. П.О. Сухого

тор переменной. Роль конструктора по умолчанию может играть конструктор, у которого все параметры имеют априорные значения.

Конструктор преобразования и конструкторы с двумя и более параметрами

Если для создания объекта необходимы параметры, то они указываются в круглых скобках после идентификатора переменной. Указываемые параметры являются параметрами конструктора класса. Если у конструктора имеется ровно один входной параметр, который не представляет собой ссылку на свой собственный класс, то соответствующий конструктор называется конструктором преобразования. Этот конструктор называется так в связи с тем, что в результате его работы на основе объекта одного типа создается объект другого типа (типа описываемого класса).

Конструктор копирования

При создании объекта его информационные члены могут быть проинициализированы значениями полей другого объекта этого же типа, то есть объект создается как копия другого объекта.

Для такого создания объекта используется конструктор копирования.

Инициализация может быть выполнена аналогично инициализации переменных встроенных типов с использованием операции присваивания совместно с объявлением объекта.

Если класс не предусматривает создания внутренних динамических структур, например, массивов, создаваемых с использованием операции new, то в конструкторе копирования достаточно предусмотреть поверхностное копирование, то есть почленное копирование информационных членов класса.

Конструктор копирования, осуществляющий поверхностное копирование, можно явно не описывать, он генерируется автоматически.

Если же в классе предусмотрено создание внутренних динамических структур, использование только поверхностного копирования будет ошибочным, так как информационные члены-указатели, находящиеся в разных объектах, будут иметь одинаковые значения и указывать на одну и ту же размещенную в динамической памяти структуру. Автоматически сгенерированный конструктора копирования в

Библиотека БГТУ им. П.О. Сухого

данном классе не позволит корректно создавать объекты такого типа на основе других объектов этого же типа.

В подобных случаях необходимо глубокое копирование, осуществляющее не только копирование информационных членов объекта, но и самих динамических структур. При этом, естественно, информационные члены-указатели в создаваемом объекте должны не механически копироваться из объекта-инициализатора, а указывать на вновь созданные динамические структуры нового объекта.

Автоматическая генерация конструкторов и деструкторов

Автоматически могут генерироваться только конструкторы по умолчанию, конструкторы копирования и деструкторы.

Если в классе явно не описано ни одного конструктора, то автоматически генерируется конструктор по умолчанию с пустым телом.

Если в классе явно описан хотя бы один конструктор, например, конструктор копирования, то конструктор по умолчанию не будет автоматически генерироваться, даже, если он необходим в соответствии с постановкой задачи.

В случае отсутствия в классе явно описанного конструктора копирования он всегда генерируется автоматически и обеспечивает поверхностное копирование.

Если в классе не описан деструктор, то всегда автоматически генерируется деструктор, который не производит никаких действий.

Таким образом, даже если в классе не описаны конструкторы и деструктор, они все равно неявно присутствуют в нем.

Дружественная функция

Иногда требуется, чтобы функция – не член класса, имела доступ к скрытым членам класса. Основная причина использования таких функций состоит в том, что некоторые функции нуждаются в привилегированном доступе более, чем к одному классу. Такие функции получили название дружественных.

Для того, чтобы функция, не член класса, имела доступ к private-членам класса, необходимо в определение класса поместить объявление этой дружественной функции, используя ключевое слово friend. Объявление дружественной функции начинается с ключевого слова friend и должно находиться только в определении класса.

Библиотека ГГТУ им. П.О. Сухого

```
void func(){...}
class A{
...
friend                                   void func();
};
```

Дружественная функция, хотя и объявляется внутри класса, функцией-членом не является. Поэтому не имеет значения, в какой части тела класса (private, public) она объявлена.

Функция-член одного класса может быть дружественной для другого класса.

```
class A
{...
 int func();
};
class B
{...
 friend          int A          ::          func();
};
```

Функция-член func() класса A является дружественной для класса B.

Если все функции-члены одного класса являются дружественными функциями второго класса, то можно объявить дружественный класс:

```
friend class ИмяКласса;
class A                                          {
...
};
class B
{...
 friend                                     class A;
};
```

3 Порядок выполнения работы

3.1 Создание папки для работы

Библиотека ГГТУ им. П.О. Сухого

При выполнении лабораторной работы № 4 вы будете использовать папку с именем Lab4, которую необходимо предварительно создать.

С этой целью откройте вашу рабочую папку (с вашей фамилией) и создайте в ней (с помощью клавиши F7) новую папку с именем Lab4.

В дальнейшем вы будете записывать и хранить в этой папке все файлы при выполнении лабораторной работы № 4. Полный путь к этой папке будет такой:

F:\TDS\Ivanov\Lab4

3.2 Применение конструктора и деструктора

Рассмотрим применение конструктора и деструктора, а также технологию полиморфизма.

Пример:

libconstructor.h

```
#ifndef LIBCONSTRUCTOR
#define LIBCONSTRUCTOR
#include <string>
#include <iostream>



class Constructor
{
public:
   Constructor();
   Constructor(std::string input);
   ~Constructor();
private:

};

#endif // LIBCONSTRUCTOR
```

libconstructor.cpp

Библиотека ГГТУ им.П.О.Сухого

```
#include "libconstructor.h"

Constructor::Constructor()
{
   std::cout << "The object is created(1)" << std::endl;
}

Constructor::Constructor(std::string input)
{
   std::cout << "The object is created(2)" << std::endl;
}

Constructor::~Constructor()
{
   std::cout << "Object deleted" << std::endl;
}
```

main.cpp

```
#include <iostream>
#include <clocale>
#include "libconstructor.h"

using namespace std;

int main()
{
   Constructor *Constr;

   cout << "1: Start programm" << endl;
   cout << "2: Create an object..." << endl;
   Constr = new Constructor();
   cout << "3: Remove the object..." << endl;
   delete Constr;

   cout << "4: Create an object..." << endl;
   Constr = new Constructor("Test");
   cout << "5: Remove the object..." << endl;
```


Библиотека ГГТУ им.П.О.Сухого

```
   delete Constr;

   return 0;
}
```

Основным классом программы является Constructor. В своем составе он имеет два конструктора и один деструктор:

1. Constructor();
2. Constructor(std::string input);
3. ~Constructor();

Первый конструктор является конструктором по умолчанию. Он не имеет входных параметров. Он вызывается, когда при создании класса не задается ни одного входного параметра. Второй конструктор является конструктором с параметром, и вызывается когда при создании класса указывается входной параметр типа std::string.

При использовании первого конструктора на экране должна появится надпись "The object is created(1)", а при вызове второго появится надпись "The object is created(2)".

3.3 Использование дружественной (friend) функции

Как уже было сказано, дружественная функция – это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях private или protected.

Рассмотрим пример:

libvect.h

```
#ifndef LIBVECT_H
#define LIBVECT_H

class vect
{
public:
   vect(unsigned int n);
   void add(unsigned int input);
   unsigned int gatValue (unsigned char number);

   friend unsigned int getMaxValue(vect *input);
```


Библиотека ГГТУ им. П.О. Сухого

```
private:
   unsigned int *vactor_;
   unsigned char count=0;
};

unsigned int getMaxValue(vect *input);

#endif // LIBVECT_H
```

libvect.cpp

```
#include "libvect.h"

using namespace std;

vect::vect(unsigned int n)
{
   vactor_ = new unsigned int [n];
}

void vect::add(unsigned int input)
{
   vactor_[count] = input;
   count++;
}

unsigned int vect::gatValue (unsigned char number)
{
   return vactor_[number];
}

unsigned int getMaxValue(vect *input)
{
   unsigned int Max=input->vactor_[0];
   for (unsigned int i=1;i<input->count;i++)
      if (input->vactor_[i]>Max) Max = input->vactor_[i];

   return Max;
```


Библиотека ГГТУ им.П.О.Сухого

```
}
```

main.cpp

```
#include <iostream>
#include "libvect.h"
using namespace std;


int main()
{
   unsigned int n;
   cout<<"Start programm"<<endl;
   cout<<"Enter the number of elements "<<endl;
   cin>>n;

   vect *vector = new vect(n);

   cout<<"n="<<n<<endl;

   for (unsigned int i=0;i<n;i++)
   {
      cout<<"vector["<<i<<"]=";
      unsigned int buf;
      cin>>buf;
      vector->add(buf);
   }

   cout<<"vector =";

   for (unsigned int i=0;i<n;i++)
   {
      cout<<" "<<vector->gatValue(i);
   }
   cout<<endl;

   cout<<"Maximum element="<<getMaxValue(vector)<<endl;
   return 0;
}
```

Библиотека ГГТУ им.П.О.Сухого

В данной программе дружественной функцией является getMaxValue. Она получает доступ к закрытым членам класса через указатель и производит их обработку.

3.4 Задание для самостоятельной работы

Разработать класс, который в качестве основного члена данных содержит строку (массив типа char) в области private. Класс должен иметь методы для заполнения строки, вывода ее на экран, вставки и удаления символа. С помощью дружественной функции реализовать обработку данного массива в соответствии с индивидуальным заданием согласно таблицы 1.

Таблица 1 – Выбор алгоритма обработки строки

№	Алгоритм обработки
1	Определить количество заданных символов в строке
2	Определить количество букв в строке
3	Определить количество запятых в строке
4	Определить количество точек в строке
5	Определить количество слов в первом предложении
6	Определить количество слов во всей строке
7	Определить количество слов в последнем предложении
8	Определить количество слов в предложении, длина которых меньше заданной
9	Определить количество слов в предложении, длина которых больше заданной
10	Определить количество предложений в строке
11	Заменить все запятые в строке точками
12	Определить количество двоеточий в строке
13	Заменить первое наибольшее слово заданным
14	Заменить первое наименьшее слово заданным
15	Заменить последнее наибольшее слово заданным
16	Заменить последнее наименьшее слово заданным
17	Удалить все слова, начинающиеся с заданной буквы
18	Удалить все слова короче заданной длины
19	Заменить первое наибольшее слово последним наименьшим

Библиотека ГГТУ им. П.О. Сухого

4 Содержание отчета

Наименование и цель работы. Краткая теоретическая часть, которая применялась вами при выполнении работы. Комментарии и снимки экрана для каждого пункта лабораторной работы, которые отражали бы весь ход выполнения. Вывод о полученных навыках и знаний после выполнения работы.

Контрольные вопросы

1. Что такое конструктор?
2. Что такое деструктор?
3. Какие виды конструкторов существуют и в чем их отличия?

Литература

1. Лафоре, Роберт. Объектно-ориентированное программирование в C++ / Роберт Лафоре – СПб.: Питер, 2014.
2. Боровский, А.Н. Qt4.7: практическое программирование на C++ / А.Н. Боровский – СПб.: БВХ-Петербург, 2012.
3. Лаптев, В.В. C++ объектно-ориентированное программирование / В.В. Лаптев – СПб.: Питер, 2008.

Библиотека ГГТУ им. П.О.Сухого

Содержание

Лабораторная работа № 1
Компилятор gcc и интегрированная среда Qt Creator ………… 3
Лабораторная работа № 2
Работа с динамической памятью и указателями ……………….... 17
Лабораторная работа № 3
Использование классов и встроенных методов …………………. 33
Лабораторная работа № 4
Использование методов класса: конструктора, деструктора и дружественной функции ...……………………………….……… 45

Литература ………………………………………………….... 56

Библиотека ГГТУ им. П.О. Сухого

Виноградов Эдуард Михайлович
Сахарук Андрей Владимирович

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ СИСТЕМ УПРАВЛЕНИЯ. УКАЗАТЕЛИ И МАССИВЫ

Практикум
по выполнению лабораторных работ
для студентов специальности 1-53 01 07
«Информационные технологии и управление
в технических системах»
дневной формы обучения

Подписано к размещению в электронную библиотеку
ГГТУ им. П. О. Сухого в качестве электронного
учебно-методического документа 12.05.17.
Рег. № 93.
http://www.gstu.by

Библиотека ГГТУ им. П.О.Сухого