МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Программирование»
Тема: Создание make-файла

Студент гр. 7381 _____ Лукашев Р.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург
2017

Цель работы.

Ознакомиться с операционной системой Linux.

Ознакомиться с системой контроля версий Git.

Ознакомиться с основами создания и сборки программ на Си. Научиться пользоваться утилитой make.

Создать проект, описанный в задании.

Основные теоретические положения.

Файл исходного кода на языке С имеет расширение .c. Чтобы скомпилировать такой файл необходимо использовать команду

gcc «name».c

Результатом выполнения такой команды будет исполняемый файл с именем a.out в текущей директории, запустить который можно таким образом:

./a.out

Изменение имени исполняемого файла можно произвести с помощью ключа -o компилятора gcc с указанием нового названия, например:

gcc main.c -o main

Исполняемый файл в таком случае будет называться main.

Функция в языке С - подпрограмма (т.е. фрагмент программного кода), которую можно вызвать по имени из другого места программы.

Определение функции выглядит следующим образом:

<тип_возвращаемого_значения> имя_функции (список_параметров_функции)
{
<Код_который_исполняется_в_функции>
}

Объявление функции сообщает, аргументы каких типов и в каком количестве функция ожидает и какой возвращает результат. Это объявление, называемое прототипом функции, должно быть согласовано с определением и всеми вызовами функции. Если определение функции или вызов не соответствует своему прототипу, возникает ошибка. Обычно, объявления функций выносят в специальные заголовочные файлы, имеющие расширение .h.

Объявление функции:

<тип_возвращаемого_значения> имя_функции (список_параметров_функции);

Если функция не возвращает значения, то <тип_возвращаемого_значения> указывается void.

Если функция не принимает аргументов, то список_параметров_функции никак не указывается.

Выполнение любой программы на языке С начинается с выполнения функции **main** (говорят также, что main - точка входа в программу). Каждая программа обязательно должна содержать функцию **main.**

Для возврата значения из функции служит оператор return. При его вызове выполнение функции немедленно завершается и функция возвращает значение, переданное return.

Значение, которое возвращает функция main, получает та среда, в которой была запущена программа. По соглашению, при корректном завершении программы, функция main должна вернуть значение 0. Ненулевое значение будет расценено операционной системой как код ошибки.

Многие функции содержатся в различных библиотеках функций, и для их использования надо явно указать заголовочный файл, в котором они объявлены. Например, для функции printf, как и для многих других функций ввода/вывода, это заголовочный файл **stdio.h** (**st**an**d**ard **i**nput/**o**utput).
#include <stdio.h> подключает нужный нам заголовочный файл **stdio.h**.
Для стандартных библиотек имя заголовочного файла следует писать в треугольных скобках. Если написать имя заголовочного файла в двойных кавычка, например, *#include "header_name.h"*, компилятор будет искать этот файл в директории с исходным кодом.

Препроцессор - это программа, которая подготавливает код программы для передачи ее компилятору.
Команды препроцессора называются директивами и имеют следующий формат:
#ключевое_слово параметры

Основные действия, выполняемые препроцессором:

- Удаление комментариев
- Включение содержимого файлов (*#include*)
- Макроподстановка (*#define*)
- Условная компиляция (*#if, #ifdef, #elif, #else, #endif*)

Компиляция - процесс преобразования программы с исходного языка высокого уровня в эквивалентную программу на языке более низкого уровня (в частности, машинном языке).

Компилятор - программа, которая осуществляет компиляцию. Большая часть компиляторов преобразует программу в машинный код, который может быть выполнен непосредственно процессором. Этот код различается между операционными системами и архитектурами. Однако, в некоторых языках программирования программы преобразуются не в машинный, а в код на более низкоуровневом языке, но подлежащий дальнейшей интерпретации (байт-код). Это позволяет избавиться от архитектурной зависимости, но влечет за собой некоторые потери в производительности.

Компилятор языка С принимает исходный текст программы, а результатом является объектный модуль. Он содержит в себе подготовленный код, который может быть объединён с другими объектными модулями при помощи линковщика для получения готового исполняемого модуля.

Можно скомпилировать каждый исходный файл по отдельности и получить для каждого из них объектный файл. Теперь необходимо получить по

ним исполняемый файл. Эту задачу решает линковщик (компоновщик) - он принимает на вход один или несколько объектных файлов и собирает по ним исполняемый модуль.

Работа компоновщика заключается в том, чтобы в каждом модуле определить и связать ссылки на неопределённые имена.

Сборка проекта - это процесс получения исполняемого файла из исходного кода.

Сборка проекта вручную может стать довольно утомительным занятием, особенно, если исходных файлов больше одного и требуется задавать некоторые параметры компиляции/линковки. В этих случаях используется Makefile - список инструкций для утилиты make, которая позволяет собирать проект сразу целиком.

Если запустить утилиту make, то она попытается найти файл с именем Makefile в текущей директории и выполнить из него инструкции.
Если требуется задать какой-то конкретный Makefile, это можно сделать с помощью ключа -f:

make -f AnyMakefileName

Любой make-файл состоит из:

- списка целей
- зависимостей этих целей
- команд, которые требуется выполнить, чтобы достичь эту цель

цель: зависимости
[tab] команда

Для сборки проекта обычно используется цель all, которая находится самой первой и является целью по умолчанию. (фактически, первая цель в файле и является целью по-умолчанию)

Также, рекомендуется создание цели clean, которая используется для очистки всех результатов сборки проекта

Использование нескольких целей и их зависимостей особенно полезно в больших проектах, так как при изменении одного файла не потребуется пересобирать весь проект целиком. Достаточно пересобрать измененную часть.

Часто бывает необходимо изменить какие-то параметры сборки. Это может стать проблемой, если придется все изменять вручную. Что бы избежать этого, полезно использовать переменные. Для этого достаточно присвоить им

значения до момента их использования и в месте использования обратиться к ним как $(VARNAME). Имена переменных принято писать в верхнем регистре, пробельный символ перед знаком '=' не допускается.

Ход работы.

Был создан проект, состоящий из пяти файлов: main.c, print_str.c, get_name.c, print_str.h, get_name.h.

- Файл get_name.c содержит описание функции, которая считывает из входного потока имя пользователя и возвращает его.
- Файл get_name.h содержит прототип функции, которая считывает из входного потока имя пользователя и возвращает его.
- Файл print_str.c содержит описание функции, которая принимает в качестве аргумента строку и выводит её (функция ничего не возвращает).
- Файл print_str.h содержит прототип функции, которая принимает в качестве аргумента строку и выводит её (функция ничего не возвращает).
- Файл main.c содержит главную функцию, которая вызывает функцию из файла get_name.h, добавляет к результату выполнения функции строку “Hello, ” и передает полученную строку в функцию вывода строки из print_str.h.

После создания проекта, для него был написан Makefile.

Выводы.

Были освоены операционная система Linux и система контроля версий Git. Были освоены принципы создания исходного кода на языке C и процесс сборки программ вручную и с использованием утилиты make. На основе полученных знаний был создан проект.

Исходный код проекта.

- Файл «get_name.c»:

```
#include "get_name.h"
#include <stdlib.h>
#include <stdio.h>

char* get_name(){
   char* name = (char*)malloc(80*sizeof(char));
   int i = 0;
   char ch;
   while ((ch = getchar()) != '\n')
   {
     name[i] = ch;
     i++;
   }
   name[i] = '\0';
   return name;
}
```

- Файл «get_name.h»:

```
char* get_name();
```

- Файл «print_str.c»:

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

void print_str(char* str){
 puts(str);
}
```

- Файл «print_str.h»:

```
void print_str(char* str);
```

- Файл «main.c»:

```
#include <string.h>
#include <stdlib.h>
#include "get_name.h"
#include "print_str.h"

int main(){
   char hello[90] = "Hello, ";
   char* result;
   result = get_name();
```

```
    print_str(strncat(hello, result, 80));
    free(result);
    return 0;
}
```

- Файл «Makefile»:

```
all: main.o get_name.o print_str.o
	gcc main.o get_name.o print_str.o
get_name.o: get_name.c get_name.h
	gcc -c get_name.c
print_str.o: print_str.h print_str.c
	gcc -c print_str.c
main.o: main.c
	gcc -c main.c
```