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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Программирование»
Тема: Сборка программ в Си

Студент гр. 1304 _____ Павлов Д.Р.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2021

Цель работы.

Изучить и научиться управлять основными командами для работы с makefiles.

Задание.

В текущей директории создайте проект с make-файлом. Главная цель должна приводить к сборке проекта. Файл, который **реализует главную функцию**, должен называться menu.c; **исполняемый файл** - menu. Определение каждой функции должно быть расположено в **отдельном файле,** название файлов указано в скобках около описания каждой функции.

Реализуйте функцию-меню, на вход которой подается одно из **значений** 0, 1, 2, 3 и **массив** целых чисел **размера не больше** 100. Числа разделены пробелами. Строка заканчивается символом перевода строки.
В зависимости от **значения**, функция должна выводить следующее:
0 : максимальное число в массиве. (max.c)

1 : минимальное число в массиве. (min.c)

2 : разницу между максимальным и минимальным элементом. (diff.c)

3 : сумму элементов массива, расположенных до минимального элемента. (sum.c)

иначе необходимо вывести строку "Данные некорректны".

Выполнение работы.

II вариант

Для выполнения данной лабораторной работы мы будем использовать код, который был написан ранее для Лабораторной Работы 1 и поэтому, дабы сэкономить время, я не буду углублять в то, как именно работает та или иная функция. На этот раз нам предстоит разбить наши использованные ранее функции на отдельные файлы. Для этого мы сначала создаем файлы при помощи команды touch в терминале Linux (*все действия производятся по пути нашей папке(src)), в которых у нас будет хранится определение каждой функции (touch min.c; touch max.c; touch diff.c; touch sum.c), потом создаем основной файл, который будет их всех задействовать(touch menu.c). После этого мы по аналогии создаем файлы, которые будут служить нам для объявления каждой из функций (touch min.h; touch max.h; touch diff.h; touch sum.h). Вскоре мы уже создаем при помощи той же команды touch, файл под название Makefile, в котором у нас будут хранится инструкции по сборке. Далее мы, естественно, убираем из нашей основной программы(main.c) все функции, которые я пометил как *Option Files и добавляем дополнительные директивы с названием наших файлов, объявляющих функции(#include „min.h“; #include „max.h“; #include „diff.h“; #include „sum.h“).

Теперь приступим в написанию наших функций. Для этого мы так же заходим в файлы при помощи команды текстового редактора (vi min.c ~ nano min.c), куда мы пока что вставим код с нашими функциями. Далее мы для каждой функции пишем директивы с прототипом функций, соотвествующие названиям этих самых функции(т.е. для файла min.c мы пишем #include „min.h“). Важно отметить, что для файлов diff.c и sum.c мы так же пишем и другие директивы(для diff.c мы дополнительно пишем #include „min.h“ и #include „max“, а для sum.c - #include „min.h“), потому что в коде этих функций мы так же используем первые две функции.

Далее приступим к написанию файлов, объявляющих каждую из функций (Единственное их различие будет лишь последняя строка). Для этого мы сначала объявляем в каждом файле директиву stdio(#include <stdio.h>), далее

вводим #define MAX_BUFFER 101 и наконец пишем int <название файла с описанием функции>();.

Теперь приступим к написанию файла, в котором хранятся все инструкции по сборке — Makefile. Для этого мы пишем сначала первое правило, с названием цели all, зависящее от всех целей которые мы напишем далее. При этой правиле будет выполняться следующая инструкция: „gcc main.o min.o max.o diff.o sum.o -o menu“,где menu станет исполняемым файлом. таким образом это правило будет выполнятся только после компилирования всех остальных файлов.

```
all: menu.o min.o max.o diff.o sum.o
	gcc menu.o min.o max.o diff.o sum.o -o menu
```

Далее мы пишем наши наши остальные правила, цели которых — скомпелированные файлы наших исходных файлов. Для main.o зависящие файлы будут menu.c, min.h, max.h, diff.h, sum.h, поскольку имеено они нужны для выполнения компеляции main.c. Далее мы пишем инструкцию: „gcc -c main.c“, которая компелирует нашу программу.

```
menu.o: menu.c min.h max.h diff.h sum.h
	gcc -c menu.c
```

Далее мы по такому же принципу пишем и другие правила для функций, где цель — скомпелированный файл, а зависящие файлы - <название файлы>.c <название файлы.h>, а инуструкция которая будет совершать компеляцию — gcc -c <название файла>.c. Единственное различие — для файлов diff.o и sum.o в качестве зависящих файлов мы им добавляем min.h, и max.h только для diff.o.

```
min.o: min.c min.h
```

```
	gcc -c min.c

max.o: max.c max.h
	gcc -c max.c

diff.o: diff.c diff.h max.h min.h
	gcc -c diff.c

sum.o: sum.c sum.h min.h
gcc -c sum.c
```

Далее мы можем начинать процесс сборки и компелирования, а затем и линковки. Для этого мы сначала пишем make, дабы все файлы скомпелировались, а потом пишем ./menu

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 10 23 535 67 45 9 64 11 53 37	535	
2.	5 364 584 657 865 346	Данные некорректны	
3.	2 10 15 15 15 15 10 5 10 10 10 10 10 10	10	
...			

Выводы.

Я иисследовал и изучил основные понятия и команды для работы с makefile.

Так же в этой лабораторной работе был исследован процесс сборки и компелирования, процесс линковки.

В этой лабораторной работе мы разраработали ряд программ, на базе функций из прошлой лабораторной работы. Для их сборки мы использовали терминал linux, работали с директориями и писали инструкции.