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: «Создание make-файла»

Студент гр. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2019

Цель работы.

Изучить структуру make-файла.

Задание.

Вариант лабораторной работы №1.

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.

Реализуйте функцию-меню, на вход которой подается одно из значений 0, 1, 2, 3 и массив целых чисел размера не больше 20. Числа разделены пробелами. Строка заканчивается символом перевода строки.

В зависимости от значения, функция должна выводить следующее:

0 – индекс первого отрицательного элемента; (*index_first_negative.c*)

1 – индекс последнего отрицательного элемента; (*index_last_negative.c*)

2 – найти произведение элементов массива, расположенных от первого отрицательного элемента (включая элемент) и до последнего отрицательного (не включая элемент); (*multi_between_negative.c*)

3 – найти произведение элементов массива, расположенных до первого отрицательного элемента (не включая элемент) и после последнего отрицательного (включая элемент); (*multi_before_and_after_negative.c*)

иначе необходимо вывести строку "*Данные некорректны*".

Основные теоретические положения.

Препроцессор - это программа, которая подготавливает код программы для передачи ее компилятору.

Основные действия, выполняемые препроцессором:

- удаление комментариев;
- включение содержимого файлов *(#include*);
- макроподстановка *(#define*);
- условная компиляция (*#if, #ifdef, #elif, #else, #endif*).

#include – директива, которая подключает другой исходный файл, имя которого указывается после директивы. Синтаксис директивы:

```
#include <название_файла>
```

#define – директива, определяющая идентификатор и последовательность символов, которой будет замещаться данный идентификатор при его обнаружении в тексте программы. Синтаксис директивы:

```
#define <имя_макроса> <последовательность_символов>
```

Директивы условной компиляции (*#if, #ifdef, #elif, #else, #endif*) допускают возможность выборочной компиляции кода. Это может быть использовано для настройки кода под определенную платформу, внедрения отладочного кода или проверки на повторное включение файла.

Линковка (Компоновка) – процесс объединения объектных файлов в единый модуль.

Make – утилита для сборки проекта. Makefile - список инструкций для утилиты make, которая позволяет собрать проект сразу целиком в исполняемый модуль.

Любой make-файл состоит из:

1) списка целей;
2) зависимостей этих целей;
3) команд, которые требуется выполнить, чтобы достичь эту цель.

Пример синтаксиса:

```
<цель>: <зависимости>
    <команда>
```

Для того, чтобы выполнить цель, необходимо выполнение блока зависимостей. Зависимостью может быть какой-либо файл или команда. Зависимость может быть целью. Первая цель в файле является целью по умолчанию.

Выполнение работы.

В программе использовались следующие переменные:

1. Целого типа (integer):

a. *array[20]* – массив целых чисел с размерностью 20;
b. *function* – переменная, которая определяет последующие действия программы в операторе switch;
c. *number* – определяет количество элементов в массиве, которое заранее не известно;
d. *i* – локальная переменная-счётчик;
e. *index_last_negative* – переменная, которая определяет последний отрицательный элемент массива;
f. *multi_between_negative* – переменная, которая определяет произведение элементов массива, расположенных между крайними отрицательными числами (но не включая первый отрицательный элемент массива);
g. *multi_before_and_after_negative* – переменная, которая определяет произведение элементов массива, расположенных не между крайними отрицательными числами (но включая последний отрицательный элемент массива).

2. Символьного типа (char):
   a. *symbol* – используется для завершения ввода элементов массива после прочтения символа «*n*».

В программе реализованы следующие функции:

1. Функция *int array_*create*(int *array)* принимает на вход незаполненный массив. Функция считывает элементы массива со строки до тех пор, пока не считает символ переноса строки (не более 20 по условию), и считает количество записанных элементов массива. Функция заполняет массив и возвращает количество элементов.
2. Функция *int index_first_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит первый отрицательный элемент массива. Функция возвращает индекс первого отрицательного элемента массива.

3. Функция *int index_last_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит последний отрицательный элемент массива. Функция возвращает индекс последнего отрицательного элемента массива.
4. Функция *int multi_between_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит произведение элементов массива между первым отрицательным элементом массива (не включая его) и последним отрицательным элементом массива (включая его). Функция возвращает данное произведение элементов массива.
5. Функция *int multi_before_and_after_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит произведение элементов массива от первого до первого отрицательного элемента массива (не включая его) и от последнего отрицательного элемента массива (включая его) до последнего. Функция возвращает данное произведение элементов массива.

В программе были реализованы следующие модули:

1. *index_first_negative.c* – модуль, содержащий функцию *int index_first_negative_function(int *array, int *number).*
2. *index_last_negative.c* – модуль, содержащий функцию *int index_last_negative_function(int *array, int *number).*
3. *multi_between_negative_function.c* – модуль, содержащий функцию *int multi_between_negative_function(int *array, int *number).*
4. *multi_ before_and_after _negative_function.c* – модуль, содержащий функцию *int multi_ before_and_after _negative_function(int *array, int *number).*
5. menu.c – основной модуль, содержащий функцию *int array_*create*(int *array)* и тело функции *main*.

Разработанный программный код см. в приложении А.

Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 1 2 3 -4 5 -6 7 -8 9 10	3	Тест пройден
2.	0 1 -2 3 -4 5	1	Тест пройден
3.	0 1 2 3 4 5 6 7 8 9 -10 11 12 13 14 15 16 17 18 19 20	9	Тест пройден
4.	1 1 -2 3 -4 5 6 7 -8 9 10	7	Тест пройден
5.	1 1 -2 3 -4 5	3	Тест пройден
6.	2 0 2 3 -4 -1 6 7 -8 9 10	168	Тест пройден
7.	2 1 2 3 -4 5 6 7 -8 9 10	-840	Тест пройден
8.	2 1 -2 3 -4 5	-6	Тест пройден
9.	2 -5 5 -5 5 -5 5 -5	-15625	Тест пройден
10.	3 0 2 3 -4 5 6 7 -8 9 10	0	Тест пройден
11.	3 1 -2 3 -4 5	-20	Тест пройден
12.	3 -5 5 -5 5 -5 5 -5	-5	Тест пройден
13.	5 1 2 3 -4 5 6 7 -8 9 10	Данные некорректны	Тест пройден
14.	8 -1 -2 -3 -4 -5	Данные некорректны	Тест пройден
15.	00009 -9 -9 -9 -9	Данные некорректны	Тест пройден

Выводы.

В ходе лабораторной работы были изучены: структура make-файла, основные директивы препроцессора.

Были разработаны модули программы. Исполняемый модуль считывает с клавиатуры исходные данные, формирует массив из исходных данных, выполняет определённые операции над этими данными, согласно условию, и выводит значение результатов обработки. Для обработки команд пользователя использовались условные операторы *if-else*, операторы циклов *for*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include <stdio.h>
#include "index_first_negative.h"
#include "index_last_negative.h"
#include "multi_between_negative.h"
#include "multi_before_and_after_negative.h"

int array_create(int *array){
    char symbol;
    for(int i=0; i<20; i++){
        scanf("%i%c", &array[i], &symbol);
        if(symbol == '\n'){
            i++;
            return i;
        }
    }
}

int main(){
    int function;
    scanf("%i", &function);
    int array[20];
    int number=array_create(array);
    switch (function){
        case 0:
            printf("%i\n",index_first_negative_function(array,
&number));
            break;
        case 1:
            printf("%i\n",index_last_negative_function(array,
&number));
            break;
        case 2:
            printf("%i\n",multi_between_negative_function(array,
&number));
            break;
        case 3:
            printf("%i\n",
multi_before_and_after_negative_function(array, &number));
            break;
        default:
            printf("Данные некорректны\n");
    }
}
```

Название файла: index_first_negative.c

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

int index_first_negative_function(int *array, int *number){
    for (int i=0; i<*number; i++){
        if (array[i]<0){
```

```
                return i;
            }
        }
    }
```

Название файла: index_first_negative.h

```
#pragma once
int index_first_negative_function(int *array, int *number);
```

Название файла: index_last_negative.c

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

int index_last_negative_function(int *array, int *number){
    int index_last_negative=-1;
    for (int i=0; i<*number; i++){
        if (array[i]<0){
            index_last_negative=i;
        }
    }
    return index_last_negative;
}
```

Название файла: index_last_negative.h

```
#pragma once
int index_last_negative_function(int *array, int *number);
```

Название файла: multi_between_negative.c

```
#include <stdio.h>
#include "index_first_negative.h"
#include "index_last_negative.h"
#include "multi_between_negative_function.h"

int multi_between_negative_function(int *array, int *number){
    int multi_between_negative=1;
    for(int     i=index_first_negative_function(array,     number);
i<index_last_negative_function(array, number); i++){
        multi_between_negative=multi_between_negative*array[i];
    }
    return multi_between_negative;
}
```

Название файла: multi_between_negative.h

```
#pragma once
int multi_between_negative_function(int *array, int *number);
```

Название файла: multi_before_and_after_negative.c

```
#include<stdio.h>
#include "index_first_negative.h"
#include "index_last_negative.h"
#include "multi_before_and_after_negative.h"
```

```
     int    multi_before_and_after_negative_function(int    *array,    int
*number){
          int multi_before_and_after_negative=1;
          for(int  i=0;  i<index_first_negative_function(array,  number);
i++){

multi_before_and_after_negative=multi_before_and_after_negative*array[i];
          }
          for(int i=index_last_negative_function(array, number); i<*number;
i++){

multi_before_and_after_negative=multi_before_and_after_negative*array[i];
          }
          return multi_before_and_after_negative;
     }
```

Название файла: multi_before_and_after_negative.h

```
     #pragma once
     int multi_before_and_after_negative_function(int *array, int
*number);
```

Название файла: Makefile

```
     all:     menu.o     index_first_negative.o     index_last_negative.o
multi_between_negative.o multi_before_and_after_negative.o
          gcc    menu.o    index_first_negative.o    index_last_negative.o
multi_between_negative.o multi_before_and_after_negative.o -o menu

     menu.o:    menu.c    index_first_negative.h    index_last_negative.h
multi_between_negative.h multi_before_and_after_negative.h
          gcc -c -std=c99 menu.c

     index_first_negative.o:                        index_first_negative.c
index_first_negative.h
          gcc -c -std=c99 index_first_negative.c

     index_last_negative.o: index_last_negative.c index_last_negative.h
          gcc -c -std=c99 index_last_negative.c

     multi_between_negative.o:                   multi_between_negative.c
index_first_negative.h index_last_negative.h multi_between_negative.h
          gcc -c -std=c99 multi_between_negative.c

     multi_before_and_after_negative.o:
multi_before_and_after_negative.c                  index_first_negative.h
index_last_negative.h multi_before_and_after_negative.h
          gcc -c -std=c99 multi_before_and_after_negative.c
```