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

Студентка гр. 0382 _____ Деткова А.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучить работу препроцессора, линковщика, компилятора. Научиться делать сборку программ, путем создания Makefile-ов.

Задание.

Вариант №3.

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 : индекс первого нулевого элемента. (index_first_zero.c)

1 : индекс последнего нулевого элемента. (index_last_zero.c)

2 : Найти сумму модулей элементов массива, расположенных от первого нулевого элемента и до последнего. (sum_between.c)

3 : Найти сумму модулей элементов массива, расположенных до первого нулевого элемента и после последнего. (sum_before_and_after.c)

иначе необходимо вывести строку "Данные некорректны".

Ошибкой в данном задании считается дублирование кода!

Подсказка: функция нахождения модуля числа находится в заголовочном файле stdlib.h стандартной библиотеки языка Си.
При выводе результата, не забудьте символ переноса строки.

Основные теоретические положения.

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

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.

Линковщик (компоновщик) — программа, которая принимает на вход объектные файлы, собирает по ним исполняемый модуль.

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

Makefile - список инструкций для утилиты make, которая позволяет собирать проект сразу целиком.

Выполнение работы.

Решение поставленных задач реализовано, как в лабораторной работе №1, но при этом весь код поделен на функции, из которых по итогу получается сборка программы. Путем написания *Makefile*-а. Каждая функция выделена в отдельный файл:

index_first_zero.h: содержит объявление функции *index_first_zero* и файлы, которые нужно включать в этой функции (*stdio.h*). Тип возвращаемого значения: *int*. Кол-во и тип аргументов: 2 аргумента типа *int.*

index_first_zero.c: определение функции, аргументами которой являются массив и его размер. Функция находит и возвращает индекс первого элемента, равного нулю, если такого элемента нет, то возвращает -1.

index_last_zero.h: содержит объявление функции *index_last_zero* и файлы, которые нужно включать в этой функции (*stdio.h*). Тип возвращаемого значения: *int*. Кол-во и тип аргументов: 2 аргумента типа *int.*

index_last_zero.c: определение функции, аргументами которой являются массив и его размер. Функция находит и возвращает индекс последнего элемента, равного нулю, если такого элемента нет, то возвращает -1.

sum_before_and_after.h: содержит объявление функции *sum_before_and_after.h* и заголовочные файлы, которые нужны для этой функции (*stdio.h, tdlib.h, index_first_zero.h, index_last_zero.h*). Тип возвращаемого значения: *int*. Кол-во и тип аргументов: 2 аргумента типа *int.*

sum_before_and_after.c: определение функции, аргументами которой являются массив и его размер. Функция находит и возвращает сумму модулей элементов до первого нулевого элемента и после последнего нулевого элемента. Если нулей в последовательности нет, то функция возвращает -1.

sum_between.h: содержит объявление функции *sum_between.h* и заголовочные файлы, которые нужны для этой функции (*stdio.h, tdlib.h, index_first_zero.h, index_last_zero.h*). Тип возвращаемого значения: *int*. Кол-во и тип аргументов: 2 аргумента типа *int.*

sum_beftween.c: определение функции, аргументами которой являются массив и его размер. Функция находит и возвращает сумму модулей элементов между первым нулевым элементом и последним нулевым элементом. Если нулей в последовательности нет, то функция возвращает -1.

menu.c: определение главной функции *main*, которая считывает одно из значений и массив чисел. В зависимости от введенного первого значения, путем обработки этого значения оператором множественного выбора switch, функция выводит результат:

0 : индекс первого нулевого элемента. (index_first_zero.c)

1 : индекс последнего нулевого элемента. (index_last_zero.c)

2 : сумма модулей элементов массива, расположенных от первого нулевого элемента и до последнего. (sum_between.c)

3 : сумма модулей элементов массива, расположенных до первого нулевого элемента и после последнего. (sum_before_and_after.c)

иначе выводится строка "Данные некорректны".

Все эти файлы собраны в одном *Makefile*-е (сборка).

Makefile:

чтобы выполнить цель — получить исполняемый файл *menu*, нужно скомпилировать и скомпоновать файлы: *menu.o index_first_zero.o index_last_zero.o sum_between.o sum_before_and_after.o* в исполняемый файл *menu*.

Чтобы получить объектный файл *menu.o* нужны *menu.c index_first_zero.h index_last_zero.h sum_between.h sum_before_and_after.h* и нужно скомпилировать без компоновки *menu.c*.

Чтобы получить объектный файл sum_between.o нужны sum_between.c sum_between.h index_first_zero.h index_last_zero.h и нужно скомпилировать без компоновки sum_between.c.

Чтобы получить объектный файл sum_before_and_after.o нужны *sum_before_and_after.c sum_before_and_after.h index_first_zero.h index_last_zero.h* и нужно скомпилировать без компоновки *sum_before_and_after.c*.

Чтобы получить объектный файл index_first_zero.o нужны *index_first_zero.c index_first_zero.h* и нужно скомпилировать без компоновки index_first_zero.*c*.

Чтобы получить объектный файл *index_last_zero.o* нужны index_last_zero.c index_last_zero.h и нужно скомпилировать без компоновки index_last_zero.*c*.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 2 3 4 5 3 4 2 -7 6 3	Данные некорректны	В массиве нет нулей — ошибка.
2.	0 3 9 8 0 4 3 2 6 9 0	3	Индекс первого нулевого элемента массива.
3.	1 2 3 4 5 6 5 3 8 -2 7 6	Данные некорректны	В массиве нет нулей — ошибка.
4.	1 2 3 4 0 -4 5 0 3 4 0 3	9	Индекс последнего нулевого элемнта.
5.	2 5 5 4 3 6 7 -1 6 4	Данные некорректны	В массиве нет нулей — ошибка.
6.	2 2 3 4 5 0 4 -1 2 3	0	Индекс первого и последнего нулевого элемента совпадают, сумма между ними равна 0.
7.	2 3 4 0 1 -1 2 0 8 7 6 5	4	Сумма моделей элементов между первым и последним нулевыми значениями.
8.	3 2 3 4 5 -9 7 6 5	Данные некорректны	В массиве нет нулей — ошибка.
9.	3 1 2 -1 3 0 6 5 4	22	Индекс первого и последнего нулевого элемента совпадают, сумма до и после нулевых равна сумме всех элементов.
10.	3 4 2 -1 0 6 5 4 3 0 -1 1	7	Сумма модулей элементов, лежащих до первого и после последнего нулевого элемента.
11.	6 9 8 75 0 7 5 4 -1 0 8	Данные некорректны	Введенное значение не равно 0, 1, 2 или 3.

Выводы.

Были изучены работа препроцессора, линковщика и компилятора. Опытным путем исследовано создание Makefile-а (сборка программы).

Разработана программа, выполняющая считывание с клавиатуры исходных данных и команды пользователя. Программа находила в зависимости от введенной пользователем команды индекс первого нулевого элемента, индекс последнего нулевого элемента, сумму модулей элементов между последним и первым нулевым элементами или сумму модулей элементов до первого и после последнего нулевого элемента. В процессе решения задачи весь код был разделен на функции, каждая из функций была или объявлением, или определением функции. Все файлы были собраны в общий Makefile.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файлов: Makefile, menu.c, index_first_zero.h, index_first_zero.c, index_last_zero.h, index_last_zero.c, sum_between.h, sum_between.c, sum_before_and_after.h, sum_before_and_after.c.

Makefile:

```
all: menu
menu: menu.o index_first_zero.o index_last_zero.o sum_between.o sum_before_and_after.o
	gcc menu.o index_first_zero.o index_last_zero.o sum_between.o sum_before_and_after.o -o menu
menu.o: menu.c index_first_zero.h index_last_zero.h sum_between.h sum_before_and_after.h
	gcc -c menu.c
sum_between.o: sum_between.c sum_between.h index_first_zero.h index_last_zero.h
	gcc -c sum_between.c
sum_before_and_after.o: sum_before_and_after.c sum_before_and_after.h index_first_zero.h index_last_zero.h
	gcc -c sum_before_and_after.c
index_first_zero.o: index_first_zero.c index_first_zero.h
	gcc -c index_first_zero.c
index_last_zero.o: index_last_zero.c index_last_zero.h
	gcc -c index_last_zero.c
clean:
	rm -rf *.o menu
```

menu.c:

```
#include "index_first_zero.h"
#include "index_last_zero.h"
#include "sum_between.h"
#include "sum_before_and_after.h"

int main()
{
    int a, arr[100], arr_size, res;
    char sym = ' ';
    scanf("%d", &a);
    for (arr_size = 0; arr_size < 100 && sym == ' '; arr_size ++)
    {
         scanf("%d%c", &arr[arr_size], &sym);
    }
    switch (a)
    {case 0:
        res = index_first_zero(arr, arr_size);
        if (res != -1)
            printf("%d\n", res);
        else
            printf("Данные некорректны\n");
        break;
```

```
     case 1:
        res = index_last_zero(arr, arr_size);
        if (res != 100)
            printf("%d\n", res);
        else
            printf("Данные некорректны\n");
        break;
     case 2:
        res = sum_between(arr, arr_size);
        if (res != -1)
            printf("%d\n", res);
        else
```

index_first_zero.h:

```
#include <stdio.h>
int index_first_zero(int arr[], int n);
```

index_first_zero.c:

```
#include "index_first_zero.h"
int index_first_zero(int arr[], int n)
{
    int index, index_first = -1;
    for (index = 0; index < n; index ++)
    {
        if (arr[index] == 0 && index_first == -1)
            index_first = index;
    }
    return index_first;
}
```

index_last_zero.h:

```
#include <stdio.h>
int index_last_zero(int arr[], int n);
```

index_last_zero.c:

```
#include "index_last_zero.h"
int index_last_zero(int arr[], int n)
{
    int index, index_last = 100;
    for (index = 0; index < n; index ++)
    {
        if (arr[index] == 0)
            index_last = index;
    }
    return index_last;
}
```

sum_between.h:

```
#include <stdio.h>
#include <stdlib.h>
#include "index_first_zero.h"
#include "index_last_zero.h"
int sum_between(int arr[], int n);
```

sum_between.c:

```
#include "sum_between.h"
int sum_between(int arr[], int n)
{
    int ind_first, ind_last, sum = 0, item;
    ind_first = index_first_zero(arr, n);
    ind_last = index_last_zero(arr, n);
    if (ind_first == -1 && ind_last == 100)
        return -1;
    for (item = ind_first; item <= ind_last; item ++)
        sum = sum + abs(arr[item]);
    return sum;
}
```

sum_before_and_after.h:

```
#include <stdio.h>
#include <stdlib.h>
#include "index_first_zero.h"
#include "index_last_zero.h"
int sum_before_and_after(int arr[], int n);
```

sum_before_and_after.c:

```
#include "sum_before_and_after.h"
int sum_before_and_after(int arr[], int n)
{
    int ind_first, ind_last, sum = 0, item;
    ind_first = index_first_zero(arr, n);
    ind_last = index_last_zero(arr, n);
    if (ind_first == -1 && ind_last == 100)
        return -1;
    for (item = 0; item <= ind_first; item ++)
    {
        sum = sum + abs(arr[item]);
    }
    for (item = ind_last; item < n; item ++)
        sum = sum + abs(arr[item]);
    return sum;
}
```