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. 0382 _____ Санников В.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2020

Цель работы.

Изучить сборку программ на Си с помощью Makefile.

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 : 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".
```

Основные теоретические положения.

•Функции библиотеки stdio.h:

◦printf()–функция выводит на консоль значине аргумета

◦scanf()–функция ввода данных из консоли

•Функция библиотеки stdlib.h:

◦abs()–функция получения модуля числа

•Циклы:

◦while(){}–каждая итерация проверяет, выполняется ли условие в круглых скобках, если оно верно, то выполняется код в фигурных скобках, а если неверно, то происходит выход из цикла

◦for(){<переменная>; <условие>; <выражение_1>}–код в теле цикла будет исполнятся до тех пор, пока объявленная в цикле переменная будет удовлетворять условию цикла, выражение_1 каким-либо способом меняет значение этой переменной

•Операторы:◦if(){} ... else{}–если выполняется условия, указанное в круглых скобках, то выполняется код в фигурных скобках после if, иначе–в фигурных скобках после else(else не является обязательной частью конструкции)

◦switch(<переменная>){casex:... break; ... default:...break;}–от значения переменной в круглых скобках зависит, какой кейс будет выполняться (например, если переменная имеет значение x–выполнится casex). Если же не будет кейса с таким значением, то выполнится код из блока default

•Функции:◦<тип_функции> имя_функции(<аргумент_1>, ... , <аргумент_n>) {}–при вызове данной функции в главной(main) функции выполняется код в фигурных скобках, а затем возвращает значение оператором return(если тип функции не void)

Makefile:

Компилятор языка C 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. Линковка (Компоновка)Мы уже знаем, что можно скомпилировать каждый исходный файл по отдельности и получить для каждого из них объектный файл. Теперь нам надо 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.3

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. Для этого используются 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 AnyMakefile

Структура make-файла:

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.

Выполнение работы.

Ход работы:

В начале программы подключаем библиотеки:

•stdio.h–используется для подключения ввода-вывода (printf(), scanf())

•stdlib.h–используются для доступа к функции abs(), которая позволяет вычислить модуль числа.

Далее создаем Makefile и прописываем цели с зависимостями. Не забываем про clean.

Делим функции на файлы с разрешением .c и объектные файлы функций с разрешением .h Главный файл menu.c

Затем объявим переменные:

массив numb[100] — будет хранить входной массив целых чисел

prob — переменная типа char, которая будет хранить символ, разделяющий элементы массива

length — переменная типа int, это длинна массива, передается для контроля выхода из цикла.

Для начала считываем данные с помощью оператора scanf. Сначала считывается переменная symbol типа int, определяющая какая из функций (index_first_zero,index_last_zero,sum_between,sum_before_and_after)будет вызвана. Далее с помощью while считываем элементы массива с клавиатуры так, чтобы они были через пробел и не превышали 100 штук.

1) Чтобы найти индекс первого нулевого элемента в массиве воспользуемся циклом for: в его условии зададим, чтобы он перебирал индексы элементов массива и остановился на нулевом элементе. С помощью оператора return возвращаем индекс нужного нам элемента.

2) Для нахождения индекса последнего нулевого элемента массива снова воспользуемся циклом for. В начале функции объявим переменную thelast типа int. Далее перебираем элементы массива с помощью for и присваиваем индекс нулевых элементов переменной i в цикле for с помощью оператора if. С помощью оператора return возвращаем переменную thelast.

3) А теперь найдем сумму модулей элементов массива, которые находятся между первым и последним нулевыми элементами массива. Для

начала объявим целочисленное значение счетчика i и сумматора sum=0. Далее используем цикл for так, чтобы сумма модулей элементов начиналась сразу с первого и до последнего нулевого элемента. Чтобы это реализовать вызовем функцию index_last_zero и index_first_zero(см. Приложение А). Суммируем элементы в sum. Возвращаем в функцию с помощью return.

4) Чтобы найти сумму модулей элементов массива до первого нулевого элемента и после последнего, сначала объявим переменные i, sum1 и sum2 типа int, которые будут выполнять роль сумматоров. С помощью двух циклов for суммируем элементы в sum1(до 0) и sum2(после 0) соответственно. (В условие for опять вызываем функции поиска первого нулевого элемента и последнего. Возвращаем в функцию sum1+sum2.

Чтобы выполнить один из пунктов (1-4), внедряем в нашу программу оператор switch. Если данные введены неправильно, выводим «Данные некорректны».

В конце программы: return 0.

Краткий алгоритм: С помощью команды make собираем наш проект. Далее ввод данных с клавиатуры(scanf) → заходим в оператор switch, выбирается ветка действий в зависимости от введенных данных → срабатывают функции с помощью заголовочных и объектных файлов → получаем ответ, либо сообщение о некорректности введенных данных.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 -21 10 0 -23 -7 -15 -14 8 -9 10 -13 -14 -27 0 -7 12 -18	2	Первый нулевой элемент с индексом 2
2.	1 2 4 -5 0 -56 0 34 3 0	8	Последний нулевой элемент с индексом 8
3.	2 3 0 4 -45 4 0 45 0	98	Сумма модулей элементов между первым нулевым и последним
4.	3 4 5 0 -45 89 2 0 4	13	Сумма модулей элементов перед первым нулевым и после последнего

Выводы.

Были изучены возможности работы с компилятором и прекомпилятором. Была изучена сборка программ на языке Си, посредством создания Makefile.

Разработана программа, выполняющая следующий алгоритм: С помощью команды make собираем наш проект. Далее ввод данных с клавиатуры(scanf) → заходим в оператор switch, выбирается ветка действий в зависимости от введенных данных → срабатывают функции с помощью заголовочных и объектных файлов → получаем ответ, либо сообщение о некорректности введенных данных.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: menu.c

```
#include <stdio.h>
#include <stdlib.h>
#include "index_first_zero.h"
#include "index_last_zero.h"
#include "sum_between.h"
#include "sum_before_and_after.h"

int main(){
    int numb[100], length=0, symbol;
    char prob = ' ';

    scanf("%d", &symbol);

    while (length < 100 && prob == ' '){
        scanf("%d%c", &numb[length], &prob);
        length++;
    }
    switch(symbol){
case 0: printf("%d", index_first_zero(numb,length));
    break;
case 1: printf("%d", index_last_zero(numb,length));
    break;
case 2: printf("%d", sum_between(numb,length));
    break;
case 3: printf("%d", sum_before_and_after(numb,length));
    break;
default:printf ("Данные некорректны");
    break;
}
    return 0;
}
```

Название файла: index_first_zero.c

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

int index_first_zero(int fnumb[], int length){
    int i;
    for (i=0; i<length && fnumb[i]!=0; i++){
    }
    return i;
}
```

Название файла: index_first_zero.h

```
int index_first_zero(int fnumb[], int length);
```

Название файла: index_last_zero.c

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

int index_last_zero(int fnumb[], int length){
    int i, thelast=0;
    for (i=0; i<length; i++){
        if (fnumb[i]==0) thelast = i;
    }
    return thelast;
}
```

Название файла: index_last_zero.h

```
int index_last_zero(int fnumb[], int length);
```

Название файла: sum_between.c

```
#include <stdio.h>
#include <stdlib.h>
#include "sum_between.h"
#include "index_first_zero.h"
#include "index_last_zero.h"

int sum_between(int fnumb[], int length){
    int i,sum=0;
        for (i = index_first_zero(fnumb, length) + 1; i <
index_last_zero(fnumb, length); i++){
    sum+=abs(fnumb[i]);
    }
    return sum;
}
```

Название файла: sum_between.h

```
int sum_between(int fnumb[], int length);
```

Название файла: sum_before_and_after.c

```
#include <stdio.h>
#include <stdlib.h>
#include "sum_before_and_after.h"
#include "index_first_zero.h"
#include "index_last_zero.h"

int sum_before_and_after(int fnumb[], int length){
    int i, sum1=0, sum2=0;
    for (i=0; i < index_first_zero(fnumb, length); i++){
      sum1+=abs(fnumb[i]);
    }
```

```
        for (i = index_last_zero(fnumb, length) + 1; i < length; i++)
{
          sum2+=abs(fnumb[i]);
        }
        return sum1 + sum2;
    }
```

Название файла: sum_before_and_after.h

```
int sum_before_and_after(int fnumb[], int length);
```

Название файла: Makefile

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