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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Программирование Си»
Тема: Сборка программ в Си.

Студентка гр. 0382 _____ Кривенцова Л.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучить сборку программ на языке Си, посредством создания Makefile.

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.

Напишите программу, выделив каждую подзадачу в отдельную функцию.

Реализуйте программу, 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)

1 : индекс последнего нулевого элемента. (index_last_zero)

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

3 : Найти сумму модулей элементов массива, расположенных до первого нулевого элемента и после последнего. (sum_before_and_after) иначе необходимо вывести строку "Данные некорректны".

Основные теоретические положения.

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.

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.

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 решения:

Используется стандартная библиотека языка си и её заголовочный файл stdlib.h (для вычисления модуля)числа.

На вход программы задаётся символ(command=*getchar()*), обозначающий комманду, которую выбирает пользователь. Далее с помощью цикла вводится массив и считывается его длина в переменную *length*. При помощи оператора *switch*, в котором описаны 5 случаев (4 из которых — команда, заданная символом, и *default*, в случае если данные были введены некорректно), и в зависимости от *command* вызывается одна из четырех заданных функций.

Чтобы найти индекс первого нуля массива, в функции void *index_first_zero* организован цикл *for*, перестающий читать массив при встрече первого нуля. Тогда индекс i цикла и является индексом первого нуля — он печатается с помощью *printf*.

Чтобы найти индекс последнего нуля массива, в функции *void index_last_zero* организован цикл *for*, читающий массив от начала до конца, и перезаписывающий в переменную *last* индекс каждого последующего встреченного нуля. Конечное значение *last* печатается с помощью *printf*.

Чтобы посчитать сумму модулей элементов массива, находящихся между первым и последним нулями, в функции *void sum_between* организовано несколько циклов. Первый

цикл *for* просто читает массив с начала до первого нуля. Следующий цикл *while* читает массив с остановленного места до конца. В нём исполняется вложенный цикл *for*, который записывает в переменную *sum2* сумму модулей всех читающихся в массиве элементов. Следует проверка, если цикл *for* закончился из-за условия (встречен ноль), а не из-за окончания массива, то в переменную *sum1*, где хранится интересующая нас сумма, записывается значение переменной *sum2*, вычисленное в цикле *for*. После окончания цикла *while* печатается значение переменной *sum1*.

Чтобы посчитать сумму модулей элементов массива, находящихся до первого и после последнего нулей, в функции *void sum_before_and_after* организовано два цикла *for*. Первый цикл прибавляет к переменной *sum1* модуль значения каждого элемента массива, пока не встречен 0. Второй цикл выполняется с момента первого нуля и до конца массива. Он прибавляет к переменной *sum2* модуль значения каждого элемента массива, но если встречен 0, он обнуляет эту переменную *sum2*. После окончания второго цикла печатается сумма переменных *sum1+sum2.*

Переменные и константы:

1.Глобальная константа, задана через #define.

#define max 100 — максимальная длина массива по условию задачи.

2. Главной функции main():

char command; - переменная, в которую считывается номер команды.

int arr[max]; - массив, в который записываются числа (входные данные) для решения задачи.

int length=0; - переменная, в которую считается фактическая длинна введённого массива. Используется как

счётчик, применяемая в цикле для считывания элементов массива.

char sym = ' '; вспомогательная символьная переменная для считывания элементов массива с клавиатуры через пробел.

3.Переменные функции void index_first_zero(int num[max],int length):

int i; - переменная счётчик для цикла.

4.Переменные функции void index_last_zero(int num[max], int length):

int i; - переменная счётчик для цикла.

int last=0; - переменная, хранящая индекс последнего встретившегося нуля.

5.Переменные функции void sum_between(int num[max], int length):

int i; - переменная счётчик для цикла.

int sum1=0; -переменная, в которой сохраняется сумма модулей элементов от первого, до последнего встретившегося нуля.

int sum2=0; - переменная, в которой записывается сумма модулей элементов от первого нуля, до окончания цикла.

6.Переменные функции void sum_before_and_after(int num[max], int length);

int i; - переменная счётчик для цикла.

int sum1=0; - переменная, в которой сохраняется сумма модулей элементов от начала массива, до первого нуля.

int sum2=0; - переменная, в которой сохраняется сумма модулей элементов от последнего нуля, до конца массива.

Функции:

1. **main().**

Функция осуществляет ввод команды и массива с клавиатуры, в зависимости от введённой команды вызывает

нужную функцию. Возвращает 0 при корректной работе. Не имеет аргументов.

2. **int index_first_zero(int num[max],int length).**

Функция выполняет поиск первого нуля в массиве и возвращает его индекс. На вход подаются два аргумента:

int num[max]; - массив, заданный в функции main() передается для обработки (поиска индекса элемента).

int length; - длина массива, посчитанная в функции main() передается для удобства выхода из цикла.

3. **int index_last_zero(int num[max], int length).**

Функция выполняет поиск последнего нуля в массиве и возвращает его индекс. На вход подаются два аргумента:

int num[max]; - массив, заданный в функции main() передается для обработки (поиска индекса элемента).

int length; - длина массива, посчитанная в функции main() передается для удобства выхода из цикла.

4. **void sum_between(int num[max], int length).**

Функция считает сумму модулей элементов массива, располагающихся между первым и последним нулями. Ничего не возвращает в качестве значения. На вход подаются два аргумента: int num[max]; - массив, заданный в функции main() передается для обработки (поиска индекса элемента).

int length; - длина массива, посчитанная в функции main() передается для удобства выхода из цикла.

5. **void sum_before_and_after(int num[max], int length).**

Функция считает сумму модулей элементов массива, располагающихся до первого нуля и после последнего. Ничего не возвращает в качестве значения. На вход подаются два аргумента: int num[max]; - массив, заданный в функции main() передается для

обработки (поиска индекса элемента).

int length; - длина массива, посчитанная в функции main() передается для удобства выхода из цикла.

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 -21 10 0 -23 -7 -15 -14 8 -9 10 -13 -14 -27 0 -7 12 - 18	2	Последний элемент массива равный нулю, имеет индекс два.
2.	2 4 -5 0 7 -2 -9 1 0 3 6	19	Сумма модулей элементов массива, стоящих между первым и последним нулём равна 19.
3.	3 91 -12 0 88 45 6 43 20 0 16 39	158	Сумма модулей элементов массива, стоящих до первого и после последнего нулей равна 158.

Выводы.

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

Разработана программа, выполняющая считывание с клавиатуры исходных данных и команды пользователя. Для обработки команд пользователя использовались оператор множественного выбора switch. Для обработки команд пользователя также использовались условные операторы if-else и циклы while, for. Программа была разбита на отдельные функции и их объявления, так же был создан Makefile.

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"
#define max 100
int main ()
{
char command;
int arr[max],length=0;
char sym = ' ';

command = getchar ();
while (length < max && sym == ' ')
    {
    scanf ("%d%c", &arr[length++], &sym);
    }
switch(command){
case '0': printf ("\n%d", index_first_zero(arr,length));
    break;
case '1': printf ("\n%d", index_last_zero(arr,length));
    break;
case '2': sum_between(arr,length);
    break;
case '3': sum_before_and_after(arr,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"
#define max 100
int index_first_zero(int num[max],int length)
{
int i;
for (i = 0; i< length && num[i]!=0; i++)
    {
    }
return i;
}
```

Название файла: index_first_zero.h

```
#define max 100
int index_first_zero(int num[max],int length);
```

Название файла: index_last_zero.c

```
#include <stdio.h>
#include <stdlib.h>
#include "index_last_zero.h"
#define max 100
int index_last_zero(int num[max], int length)
{
int i,last=0;
for (i = 0; i < length; i++)
    {if (num[i]==0) last=i;
    }
return last;
}
```

Название файла: index_last_zero.h

```
#define max 100
int index_last_zero(int num[max],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"
#define max 100
void sum_before_and_after(int num[max], int length)
{
int i,sum=0;
for (i = 0;i < index_first_zero(num, length); i++){
sum+=abs(num[i]);
}
for (i =1 + index_last_zero(num, length); i < length; i++){
sum+=abs(num[i]);
}
printf ("\n%d", sum);
}
```

Название файла: sum_before_and_after.h

```
#define max 100
void sum_before_and_after(int num[max], 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"
#define max 100
void sum_between(int num[max], int length)
{
int i=0,sum=0;
for (i = 1 + index_first_zero(num,length); i <
index_last_zero(num,length); i++){
sum+=abs(num[i]);
}
printf ("\n%d", sum);
}
```

Название файла: sum_between.h

```
#define max 100
void sum_between(int num[max], 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
```