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 _____ Крючков А.М.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Изучение процесса сборки программ на языке си при помощи утилиты Make.

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.

1 : индекс последнего нечётного элемента. (index_last_odd.c)

2 : Найти сумму модулей элементов массива, расположенных от первого чётного элемента и до последнего нечётного, включая первый и не включая последний. (sum_between_even_odd.c)

3 : Найти сумму модулей элементов массива, расположенных до первого чётного элемента (не включая элемент) и после последнего нечётного (включая элемент). (sum_before_even_and_after_odd.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 программы для передачи ее компилятору.

Команды препроцессора называются директивами и имеют следующий формат:

#ключевое_слово параметры

Основные действия, выполняемые препроцессором:

- Удаление комментариев
- Включение содержимого файлов (*#include*
- Макроподстановка (*#define*)
- Условная компиляция (*#if, #ifdef, #elif, #else, #endif*)

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) 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.

Выполнение работы.

Написание кода производилось на базе системы Linux Ubuntu 20.04 через интегрированную среду разработки Visual Studio Code.

Описание вводимых переменных:

- n — число, номер исполняемой функции. Вводится пользователем
- list - массив типа i*nt* размером 100, предназначенный для хранения целых чисел, введённых пользователем
- listsize — число типа int, хранит в себе количество записанных элементов массива list

Описание созданных файлов:

- sum_before_even_and_after_odd.h – содержит объявление функции sum_before_even_and_after_odd. sum_before_even_and_after_odd.c – включает содержимое файла sum_before_even_and_after_odd.h с помощью команды #include " sum_before_even_and_after_odd.h" и содержит определение функции sum_before_even_and_after_odd.
- sum_between_even_odd.h - содержит объявление функции sum_between_even_odd. sum_between_even_odd.c – включает содержимое файла sum_between_even_odd.h с помощью команды #include "sum_between_even_odd.h" и содержит определение функции sum_between_even_odd.
- index_last_odd.h – содержит объявление функции index_last_odd. index_last_odd.c – включает содержимое файла index_last_odd.h с помощью команды #include "index_last_odd.h" и содержит определение функции index_last_odd.
- index_first_even.h – содержит объявление функции index_first_even. index_first_even.c – включает содержимое файла index_first_even.h с помощью команды #include "index_first_even.h" и содержит определение функции index_first_even.
- menu.c – содержит главную функцию main() и связывает все вышеописанные функции с помощью директивы #include, включая заголовочные файлы.

- Файл Makefile, в котором прописываются команды для компиляции программы. С помощью утилиты make собирается программа, названная menu.

Описание используемых функций:

- index_first_even (на вход массив типа int и размер этого массива типа int) — возвращает индекс первого чётного элемента.
- index_last_odd (на вход массив типа int и размер этого массива типа int) — возвращает индекс последнего нечётного элемента.
- sum_between_even_odd (на вход массив типа int и размер этого массива типа int) — возвращает сумму модулей элементов массива, расположенных от первого чётного элемента и до последнего нечётного, включая первый и не включая последний. .
- sum_before_even_and_after_odd (на вход массив типа int и размер этого массива типа int) — возвращает сумму модулей элементов массива, расположенных до первого чётного элемента (не включая элемент) и после последнего нечётного (включая элемент).

Ввод значений происходит в цикле while, сначала вводится значение n затем заполняется list.

При помощи оператора *switch*, в зависимости от значения переменной *n*, функцией *printf* выводится на консоль:

- 0 : индекс первого чётного элемента. (index_first_even)
- 1 : индекс последнего нечётного элемента. (index_last_odd)
- 2 : сумма модулей элементов массива, расположенных от первого чётного элемента и до последнего нечётного, включая первый и не включая последний. (sum_between_even_odd)
- 3 : сумма модулей элементов массива, расположенных до первого чётного элемента (не включая элемент) и после последнего нечётного (включая элемент). (sum_before_even_and_after_odd)
- Иначе вывводится строка "Данные некорректны".

Разработанный программный код см. в приложении А.

Выводы.

Были изучены условия, циклы, оператор switch, рафота с функциями в языке программирования С.

Разработана программа, которая собирается из нескольких файлов с помощью Makefile и утилиты make, выполняющая считывание исходных с помощью функции scanf() и цикла *while(){}* в переменную *n* и массив *list[100],* условием которого было равенство переменной *c*, хранящей код символа между числами, коду символа пробела, написаны функции для обработки входных результатов, подробное описание которых приведено в разделе «выполнение работы», с помощью оператора *switch(){}* и функции *printf()* реализован вывод реузльтата определённой функции в зависимости от входного управляющего значения *option*:

- если *n = 0* — выводится результат функции *int index_first_even()*;
- если *n = 1* — выводится результат функции *int index_last_odd()*;
- если *n = 2* — выводится результат функции *int sum_between_even_odd()*;
- если *n = 3* — выводится результат функции *int sum_before_even_and_after_odd()*;

Если значение *n* не соответствует ни одному из перечисленных — выводится строка «Данные некорректны».

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: menu.c

```
#include <stdio.h>
#include <stdlib.h>
#include "index_first_even.h"
#include "index_last_odd.h"
#include "sum_between_even_odd.h"
#include "sum_before_even_and_after_odd.h"
int main(){
int n = -1, list[100]; // n - значение для функции, list - список для вводимых символов
int listsize = 0; // насколько массив заполнен

char sym = ' ';
while(listsize<100 && sym == ' '){ //Ввод
if (n == -1) scanf("%d%c", &n, &sym);
else scanf("%d%c", &list[listsize++], &sym);
}

switch (n)
{
case 0:
printf("%d\n", index_first_even(list, listsize));
break;
case 1:
printf("%d\n", index_last_odd(list, listsize));
break;
case 2:
printf("%d\n", sum_between_even_odd(list, listsize));
break;
case 3:
printf("%d\n", sum_before_even_and_after_odd(list, listsize));
break;
default:
printf("Данные некорректны\n");
break;
}
return 0;
}
```

Название файла: sum_before_even_and_after_odd.c

```
#include <stdlib.h>
```

```
#include "index_first_even.h"
#include "index_last_odd.h"
#include "sum_between_even_odd.h"
int sum_before_even_and_after_odd(int list[], int listsize){
int sum = 0;

for (int i = 0; i < listsize; i++) sum += abs(list[i]);

return sum - sum_between_even_odd(list, listsize);
}
```

Название файла: sum_before_even_and_after_odd.h

```
int sum_before_even_and_after_odd(int list[], int listsize);
```

Название файла: index_first_even.c

```
int index_first_even(int list[], int listsize){

for (int i = 0; i < listsize; i++){
if(list[i]%2==0) {
return i;
}
}
}
```

Название файла: index_first_even.h

```
int index_first_even(int list[], int listsize);
```

Название файла: sum_between_even_odd.c

```
#include <stdlib.h>

int index_last_odd(int list[], int listsize){

for (int i = listsize - 1; i >= 0; i--){

if(abs(list[i])%2==1) {

return i;

}

}

}
```

Название файла: sum_between_even_odd.h

```
int sum_between_even_odd(int list[], int listsize);
```

Название файла: index_last_odd.c

```
#include <stdlib.h>

int index_last_odd(int list[], int listsize){
for (int i = listsize - 1; i >= 0; i--){
if(abs(list[i])%2==1) {
return i;
```

```
}
}
}
```

Название файла: index_last_odd.h

```
int index_last_odd(int list[], int listsize);
```

Название файла: Makefile.c

```
all:	menu.o	index_first_even.o	index_last_odd.o
sum_between_even_odd.o sum_before_even_and_after_odd.o

	gcc	menu.o	index_first_even.o	index_last_odd.o
sum_between_even_odd.o sum_before_even_and_after_odd.o -o menu
menu.o:	menu.c	index_first_even.h	index_last_odd.h
sum_between_even_odd.h sum_before_even_and_after_odd.h
	gcc -c menu.c
index_first_even.o: index_first_even.c index_first_even.h
	gcc -c index_first_even.c
index_last_odd.o: index_last_odd.c index_last_odd.h
	gcc -c index_last_odd.c
sum_between_even_odd.o:	sum_between_even_odd.c
sum_between_even_odd.h index_first_even.h index_last_odd.h
	gcc -c sum_between_even_odd.c
sum_before_even_and_after_odd.o:	sum_before_even_and_after_odd.c
sum_before_even_and_after_odd.h index_first_even.h index_last_odd.h
	gcc -c sum_before_even_and_after_odd.c
clean:
	rm *.o menu
```