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

ОТЧЕТ

по лабораторной работе №2
по дисциплине «Программирование на Си»
Тема: Сборка программ в Си

Студент гр. 0382 _____ Азаров М.С.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Изучить процесс сборки программ, написанных на языке С на примере использования make-файлов.

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 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.

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*)

#***include***

Препроцессор обрабатывает содержимое указанного файла и включает содержимое на место директивы. Включаемые таким образом файлы называются заголовочными и обычно содержат объявления функций, глобальных переменных, определения типов данных и другое.
Директива может иметь вид #*include* "...." либо #*include* <...>. Для <...> поиск файла осуществляется среди файлов стандартной библиотеки, а для "..." - в текущей директории.

#***define***

Позволяет определить макросы или макроопределения. Имена их принято писать в верхнем регистре через нижние подчеркивания, если это требуется:
#define SIZE 10
Такое макроопределение приведет к тому, что везде, где в коде будет использовано *SIZE*, на этапе работы препроцессора это значение будет заменено на 10. Макросы отличаются только наличием параметров:

*#define MUL_2(x) x*2*

Таким образом, каждый макрос *MUL_2* в коде будет преобразован в выражение *x*2,* где x - его аргумент.

Следует обратить особое внимание, что *define* выполняет просто подстановку идентификатора (без каких-то дополнительных преобразований), что иногда может приводить к ошибкам, которые трудно найти.

#if, #ifdef, #elif, #else, #endif

Директивы условной компиляции допускают возможность выборочной компиляции кода. Это может быть использовано для настройки кода под определенную платформу, внедрения отладочного кода или проверки на повторное включение файла.

##

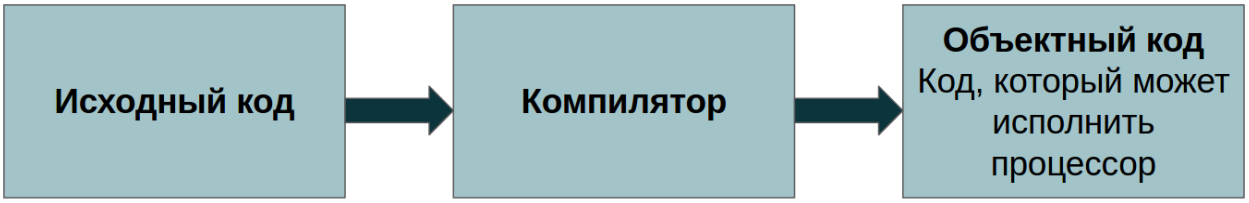
Оператор ## используется для объединения двух лексем, что может быть полезным.

Компиляция

Немного терминологии

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

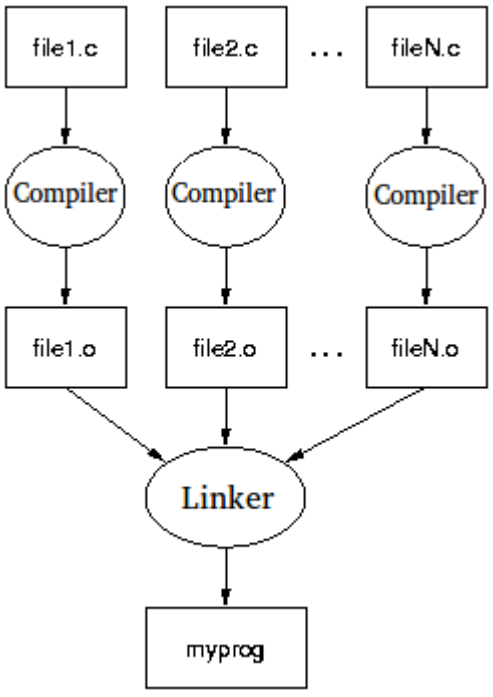
различается между операционными системами и архитектурами. Однако, в некоторых языках программирования программы преобразуются не в машинный, а в код на более низкоуровневом языке, но подлежащий дальнейшей интерпретации (байт-код). Это позволяет избавиться от архитектурной зависимости, но влечет за собой некоторые потери в производительности.

Компилятор языка 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.

Работа компоновщика заключается в том, чтобы в каждом модуле определить и связать ссылки на неопределённые имена.



Сборка. Make

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. Достаточно пересобрать измененную часть

Пример:

```
all: hello
hello: main.o f1.o f2.o
	gcc main.o f1.o f2.o -o hello
main.o: main.c
	gcc -c main.c
f1.o: f1.c
	gcc -c f1.c
f2.o: f2.c
	gcc -c f2.c
clean:
	rm -rf *.o hello
```

Таким образом, что ты выполнить цель all, требуется выполнить цель hello

Для выполнения цели hello, а именно вызова gcc для объектных файлов, требуется что бы были выполнены цели соответствующие этим объектным файлам.

Для выполнения цели для каждого объектного файла требуется скомпилировать исходный код.

Комментарии и переменные

Часто бывает необходимо изменить какие-то параметры сборки. Это может стать проблемой, если придется все изменять вручную. Что бы избежать этого, полезно использовать переменные. Для этого достаточно присвоить им значения до момента их использования и в месте использования обратиться к ним как $(*VARNAME*). Имена переменных принято писать в верхнем регистре.

```
# It is comment line. using CC variable you can easily switch compiler to compile the project
CC=gcc
# You can change the flags for the compilation in one line
CFLAGS=-c -Wall

all: hello
hello: main.o f1.o f2.o
	$(CC) main.o f1.o f2.o -o hello
main.o: main.c
	$(CC) $(CFLAGS) main.c
f1.o: f1.c
	$(CC) $(CFLAGS) f1.c
f2.o: f2.c
	$(CC) $(CFLAGS) f2.c
clean:
	rm -rf *.o hello
```

Выполнение работы.

Программа из лабораторной работы №1 была разделена на несколько файлов, а именно:

index_first_negative.c — описание функции **idx_frst_otr** , которая находит индекс первого отрицательного элемента входного массива и возвращает это значение вместо себя.

index_first_negative.h — заголовочный файл , объявление функции **idx_frst_otr.**

index_last_negative.c — описание функции **idx_last_otr** , которая находит индекс последнего отрицательного элемента входного массива и возвращает это значение вместо себя.

index_last_negative.h — заголовочный файл , объявление функции **idx_last_otr.**

multi_between_negative.c — описание функции **mult_btwn_otr** , которая считает произведение элементов массива, расположенных от первого отрицательного элемента (включая элемент) и до последнего отрицательного (не включая элемент) и возвращает это значение вместо себя.

multi_between_negative.h - заголовочный файл , объявление функции **mult_btwn_otr.**

multi_before_and_after_negative.c — описание функции **mult_bef_and_af_otr** , которая считает произведение элементов массива, расположенных до первого отрицательного элемента (не включая элемент) и после последнего отрицательного (включая элемент) и возвращает это значение вместо себя.

multi_before_and_after_negative.h - заголовочный файл , объявление функции **mult_bef_and_af_otr.**

menu.c - описание основной функции **main.**

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

Разработанный программный код и содержание файлов см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 -4 5 7 -5 4	0	Программа работает правильно
2.	1 -3 5 -6 83 40 59 4	2	Программа работает правильно
3.	2 5 3 -6 3 5 -1 -6 9	90	Программа работает правильно
4.	2 84 54 9 3 -3 -90 32	-3	Программа работает правильно
5.	3 4 4 6 -1 4 9 -3	-288	Программа работает правильно
6.	3 4 -4 1	-16	Программа работает правильно
7.	5 9 39 -4 9	Данные некорректны	Программа работает правильно
8.	0 34 6 4 35 6	Данные некорректны	Программа работает правильно

Выводы.

Был изучен процесс сборки программ, написанных на языке С на примере использования make-файлов.

Изучено как происходит процесс компиляции и линковки с использованием компилятора gcc . Заданная программа была разделена на несколько файлов , изучены структура и правила составления make-файлов а также написан make-файл для сборки заданной программы.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: **index_first_negative.c**

```
int idx_frst_otr(int arr[],int len){
	int i, bol = 0,res;

	for (i = 0; (i < len)&&(bol == 0);++i)
		if (arr[i] < 0){
			res = i;
			bol = 1;
		}
	if (bol == 1) return res;
	else return -1;
}
```

Название файла: **index_first_negative.h**

```
int idx_frst_otr(int arr[],int len);
```

Название файла: **index_last_negative.c**

```
int idx_last_otr(int arr[],int len){
	int i,bol = 0,res;

	for (i = 0; i < len ;++i)
		if (arr[i] < 0){
```

```
                res = i;
               bol = 1;
          }
    if (bol == 1) return res;
      else return -1;

      }
```

Название файла: **index_last_negative.h**

```
int idx_last_otr(int arr[],int len);
```

Название файла: **multi_between_negative.c**

```
#include "index_first_negative.h"
#include "index_last_negative.h"


int mult_btwn_otr(int arr[],int len,int *err){
        int from,to,i,res;
        res = 1;
        *err = 0;
        from = idx_frst_otr(arr,len);
        to = idx_last_otr(arr,len);

        if ((to == -1)||(from == -1)){
                *err = 1;
                return 0;
```

```
        }
    else {
            for (i = from; i < to; ++i)
                res = res * arr[i];
            return res;
        }
}
```

Название файла: **multi_between_negative.h**

```
int mult_btwn_otr(int arr[],int len,int *err);
```

Название файла: **multi_before_and_after_negative.c**

```
#include "index_first_negative.h"
#include "index_last_negative.h"


int mult_bef_and_af_otr(int arr[],int len,int *err){
        int from,to,i,res;
        res = 1;
        *err = 0;
        to = idx_frst_otr(arr,len);
    from = idx_last_otr(arr,len);

        if ((to == -1)||(from == -1)){
            *err = 1;
            return 0;
```

```
        }
        else{
        for (i = 0; i < to; ++i)
                        res = res * arr[i];
                for (i = from; i < len; ++i)
                        res = res * arr[i];
                return res;
        }
}
```

Название файла: **multi_before_and_after_negative.h**

```
int mult_bef_and_af_otr(int arr[],int len,int *err);
```

Название файла: **menu.c**

```
#include "index_first_negative.h"
#include "index_last_negative.h"
#include "multi_between_negative.h"
#include "multi_before_and_after_negative.h"

#include <stdio.h>

#define L 20


int getstr(int arr[]){
```

```
    char ch = 0;
    int i ;

    for (i = 0; ch != '\n'; i++){
        scanf("%i",&arr[i]);
         ch = getchar();
    }

    return i;

}


int main(){

    int flag,len,i,res,err;
    int arr[L];

    scanf("%d",&flag);

    len = getstr(arr);

    switch (flag){
        case 0: res = idx_frst_otr(arr,len);
            if (res == -1) printf("Данные некорректны\n");
            else printf("%i\n",res);
            break;
```

```
            case 1: res = idx_last_otr(arr,len);
                    if (res == -1) printf("Данные некорректны\n");
                   else printf("%i\n",res);
                    break;

            case 2: res = mult_btwn_otr(arr,len,&err);
               if (err == 1) printf("Данные некорректны\n");
               else printf("%i\n",res);
                    break;

            case 3: res = mult_bef_and_af_otr(arr,len,&err);
               if (err == 1) printf("Данные некорректны\n");
               else printf("%i\n",res);
                    break;

            default: printf("Данные некорректны\n");
        }

        return 0;
}
```

Название файла: **Makefile**

```
all : menu

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

```
menu.o : menu.c multi_before_and_after_negative.h multi_between_negative.h index_last_negative.h index_first_negative.h
	gcc -c  menu.c

multi_before_and_after_negative.o: multi_before_and_after_negative.c index_last_negative.h index_first_negative.h
	gcc -c multi_before_and_after_negative.c

multi_between_negative.o: multi_between_negative.c index_last_negative.h index_first_negative.h
	gcc -c multi_between_negative.c

clean:
	rm *.o menu
```