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

ОТЧЕТ по лабораторной работе №2 по дисциплине «Программирование Си»

Тема: Сборка программ в Си.

Студент гр. 0382 _____ Ильин Д.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2020

Цель работы.

Сборка нескольких файлов при помощи “make”.

Задание.

Вариант 1.

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

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

Команды препроцессора называются директивами и имеют следующий формат:

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.

##

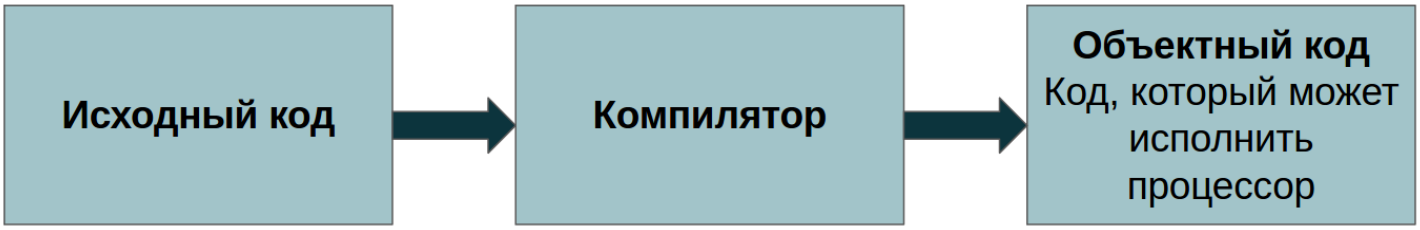
Оператор ## используется для объединения двух лексем, что может быть полезным.

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 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. 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.

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 была разбита на отдельные функции и их объявления, так же был создан Makefile.

Переменные:

- command- команда, в соответствии с которой программа должна обработать поступающие данные
- return_command- то, что возвращает функция, соответствующая определённой команде
- length- количество поступивших чисел, которые надо обработать
- mass- массив чисел, которые надо обработать
- read- считываемые данные(поэлементно)
- elem- переменная, пи помощи которой заполняется «mass»
- index- индекс элемента «mass», при помощи которого происходит выполнение соответствующих функций
- first- индекс первого отриц. числа в «mass»
- last- индекс последнего отриц. числа в «mass»
- between- произведение элементов массива, расположенных от первого отрицательного элемента (включая элемент) и до последнего отрицательного (не включая элемент)
- before_and_after- произведение элементов массива, расположенных до первого отрицательного элемента (не включая элемент) и после последнего отрицательного (включая элемент) **Функции:**

main – считывает входящие данные, при помощи функции getchar()

Сначала считывает первый элемент, по условию задачи- это и есть команда, далее считывает каждый элемент по отдельности и создаёт массив целых чисел.

Алгоритм создания массива целых чисел:

Пока считываемый элемент не \n берётся следующий элемент из ввода, если он не пробел, то элемент массива соответствующего индекса умножается на 10 и к нему прибавляется считанное число (изначально массив наполнен нулями), если же элемент- пробел, то индекс увеличивается на 1.

Далее при помощи оператора switch вызывается функция соответствующая поданной команде, результат которой печатается.

index_first_negative- принимает на вход массив, в котором ищет первый отрицательный элемент при помощи цикла for.

index_last_negative- принимает на вход массив, в котором ищет последний отрицательный элемент при помощи цикла for.

multi_between_negative- принимает на вход массив, перемножает элементы массива, расположенные от первого отрицательного элемента (включая элемент) и до последнего отрицательного (не включая элемент)

multi_before_and_after_negative- принимает на вход массив и его длину, перемножает элементы массива, расположенные до первого отрицательного элемента (не включая элемент) и от последнего отрицательного (включая элемент)

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица *1* – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 -5 -3 -5 -8 3 -9 -3	0	Первый отрицательный элемент массива, имеет индекс ноль.
2.	2 124 513 -2 2 2 2 -14 512	-16	Произведение элементов массива, стоящих между первым(включая) и последним(не включая) отрицательным равна -16.
3.	2 51 -2 -2 2 2 -2 13256 234	16	Произведение элементов массива, стоящих между первым(включая) и последним(не включая) отрицательным равна 16.
4.	3 11 -623 -512 -25 125 54 3 2 -5 1 -1 11 121	-14641	Произведение элементов массива, стоящих до первого(не включая) и после последнего(включая) отрицательного равна 14641.

Выводы.

Были изучены возможности работыс компилятором и прекомпилятором.

Были изучены варианты рабыты с “make”.

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

int main() {
int f, a, length, t;
a = getchar();
length = 0;
char k = getchar();
int mass[20] = {0};
f = -1;

while (k != '\n'){
k = getchar();

if (k == '-'){
while ((k != ' ') && (k != '\n')){
if (k == '-'){
k = getchar();
}
else {
t = k - '0';
mass[length] = mass[length]*10 - t;
k = getchar();
}
```

```
}
}
else {
while ((k != ' ') && (k != '\n')){
t = k - '0';
mass[length] = mass[length]*10 + t;
k = getchar();
}
}
++length;

}

switch(a){
case '0':
f = index_first_negative(mass);
printf("%d", f);
break;
case '1':
f = index_last_negative(mass);
printf("%d", f);
break;
case '2':
f = multi_between_negative(mass);
printf("%d", f);
break;
case '3':
f = multi_before_and_after_negative(mass, length);
printf("%d", f);
break;
```

```
default:
printf ("Данные некорректны");
break;
}
return 0;

}
```

Название файла: index_first_negative.c

```
#include "index_first_negative.h"

int index_first_negative(int mass[20]) {
int i, first;
first = -1;
for (i = 0; i < 20; i++) {
if ((first == -1) && (mass[i] < 0) ){
first = i;
}
}
return first;
}
```

Название файла: index_first_negative.h

```
#include <stdio.h>

int index_first_negative(int mass[20]);
```

Название файла: index_last_negative.c

```
#include "index_last_negative.h"

int index_last_negative(int mass[20]) {
int i, last;
last = -1;
for (i = 19; i >= 0; i--) {
if ((last == -1) && (mass[i] < 0) ){
last = i;
}
}
return last;
}
```

Название файла: index_last_negative.h

```
#include <stdio.h>

int index_last_negative(int mass[20]);
```

Название файла: multi_between_negative.c

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

```
#include "multi_between_negative.h"

int multi_between_negative(int mass[20]){
int i, between, first, last;
between = 1;
first = index_first_negative(mass);
last = index_last_negative(mass);
for (i = first; i < last; i++) {
between = between * (mass[i]);
}
return between;
}
```

Название файла: multi_between_negative.h

```
#include <stdio.h>

int multi_between_negative(int mass[20]);
```

Название файла: multi_before_and_after_negative.c

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

int multi_before_and_after_negative(int mass[20], int length) {
```

```
int i, before_and_after, first, last;


before_and_after = 1;

first = index_first_negative(mass);

last = index_last_negative(mass);


for (i = 0; i < first; i++) {

before_and_after = before_and_after * (mass[i]);

}


for (i = last; i < length; i++) {

before_and_after = before_and_after * (mass[i]);

}


return before_and_after;

}
```

Название файла: multi_before_and_after_negative.h

```
#include <stdio.h>


int multi_before_and_after_negative(int mass[20], int length);
```

Название файла: Makefile

```
menu: menu.o index_first_negative.o index_last_negative.o
multi_between_negative.o multi_before_and_after_negative.o
```

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


menu.o: menu.c index_first_negative.h

	gcc -c menu.c


index_first_negative.o: index_first_negative.c

	gcc -c index_first_negative.c


index_last_negative.o: index_last_negative.c

	gcc -c index_last_negative.c


multi_between_negative.o: multi_between_negative.c

	gcc -c multi_between_negative.c


multi_before_and_after_negative.o: multi_before_and_after_negative.c

	gcc -c multi_before_and_after_negative.c


clean:

	rm -rf *.o menu
```