

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

ОТЧЕТ
по лабораторной работе №1
по дисциплине «Программирование»
Тема: Условия, циклы, оператор switch

Студент гр. 0382

Павлов С.Р.

Преподаватель

Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы.

Изучить сборку программ на языке Си, посредством создания 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. 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 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 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: зависимости[tab] командаДля 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.⁴

Использование нескольких целей и их зависимостей особенно полезно в больших проектах, так как при изменении одного файла не потребуется пересобирать весь проект целиком.

Непосредственно программа:

В данной работе были использованы такие конструкции языка Си как:

- Функции стандартной библиотеки ввода-вывода:
 - `printf()` - выводит принимаемые значения на консоль;
 - `scanf()` - считывает входные данные из консоли;
- Операторы:
 - `if(){}` - если выражение в круглых скобках верно, выполняет блок кода в фигурных скобках;
 - `switch(){case x: ; default:}` - в зависимости от значения переменной в круглых скобках, выполняет блок кода, находящийся после «`case x:`», где `x` — значение переменной в круглых скобках. Если `x` не соответствует ни одному `case`, то выполняет блок кода, находящийся после «`default:`».
- Циклы:
 - `while(){}` - на каждой итерации проверяется выражение в круглых скобках, если оно верно — выполняется блок кода в фигурных скобках, иначе — производится выход из цикла;
 - `for(<переменная>, <выражение 1>, <выражение 2>){}` - первым аргументом является переменная цикла, далее, если верно выражение 1 — выполняется блок кода в фигурных скобках и выражение 2, которое зачастую связано с переменной цикла;
- Пользовательские функции:

- *<тип_возвращаемого_значения> имя_функции (список_параметров_функции) {return <возвращаемое_значение>;}* - при вызове в функции *main* выполняет блок кода в фигурных скобках, используя переданные параметры, и возвращает значение после оператора *return* (если тип возвращаемого значения не *void*).

Выполнение работы.

Для решения поставленных задач необходимо считать данные, обработать их и вывести результат на консоль.

Для считывания входных данных используются переменные:

- *command* типа *int* — в этой переменной хранится значение управляющего символа (0, 1, 2 или 3);
- *array* массив типа *int* размера 100 элементов — массив, предназначенный для хранения массива целых чисел, введенных пользователем;
- *size* типа *int* с начальным значением ноль — переменная, хранящая текущее значения индекса нового элемента массива;
- *index* типа *int* — переменная, в которой хранится символ, введенный после числа.

Далее с помощью функции *scanf* в переменную *comand* считывается управляющее значение, после чего с помощью цикла *while*, в каждой итерации которого проверяются условия: *index < 100* и *getchar() != '\n'*, и функцией *scanf* считывается очередной целочисленный элемент массива и следующий за ним символ, также значение переменной *index* и *size* увеличивается на 1 при помощи постфиксного инкремента.

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

- значение функции *index_first_zero(array)* если *command == 0*;

- значение функции *index_last_zero(array,size)* если *command == 1*;
- значение функции *sum_between(array,size)* если *command == 2*;
- значение функции *sum_before_and_after(array,size)* если *command == 3*;
- строка «Данные некорректны» если *command* имеет другое значение.

Описание используемых функций:

1. Функция *int index_first_zero(int array[100])*.

В качестве аргументов принимает целочисленный массив *array*. Объявляется целочисленная переменная *index* в которую будет записан индекс первого нулевого элемента в массиве.

Далее с помощью цикла *while* все элементы массива начиная с индекса 0 и до конца массива, проходят проверку на условие окончание цикла, как только цикл обнаружит нулевой элемент он прекратит итерацию и завершиться, и в переменной *index* будет записано последнее значение присвоенное циклом переменной *index*, которое является индексом первого нулевого элемента.

С помощью оператора *return* функцией *index_first_zero* будет возвращено значение элемента *index*.

2. Функция *int index_last_zero(int array[100], int size)*.

В качестве аргументов принимает целочисленный массив *array* и целочисленную переменную *size*, хранящую кол-во элементов массива. В целочисленную переменную *index* записывается длина массива (*size-1*).

Далее с помощью цикла *while* перебирает все элементы массива начиная с его конца (*size-1*) до первого встретившегося нулевого элемента. Если такой элемент будет найден, то цикл прекратит свою работу, а

присвоенное им значение переменной `index` будет являться последним нулевым элементом.

С помощью оператора `return` функцией `index_last_zero` будет возвращено значение элемента `index`.

3. Функция `int sum_between(int array[100], int size)`.

В качестве аргументов принимает целочисленный массив `array` и целочисленную переменную `size`, хранящую кол-во элементов массива. 1) В целочисленную переменную `last_zero` записывается значение возвращенное функцией `index_last_zero(array,size)`, которое является индексом последнего нулевого элемента. 2) В целочисленную `int i` записывается значение возвращенное функцией `index_first_zero(array)`, которое является индексом первого нулевого элемента. 3) Так же объявляется `sum`, которое принимает значение 0.

Далее с помощью цикла `for` все модули элементов массива находящиеся между первым нулевым элементом (`i`) и последним нулевым элементом (`last_zero`), суммируются и их сумма записываются в переменную `sum`.

С помощью оператора `return` функцией `sum_between` будет возвращено значение элемента `sum`.

4. Функция `int sum_before_and_after(int array[100], int size)`.

В качестве аргументов принимает целочисленный массив `array` и целочисленную переменную `size`, хранящую кол-во массива. 1) В целочисленную переменную `i_last` записывается значение возвращенное функцией `index_last_zero(array,size)`, которое является индексом последнего нулевого элемента. 2) В целочисленную `int i_first` записывается значение возвращенное функцией `index_first_zero(array)`, которое является индексом первого нулевого элемента. 3) Так же объявляется `sum`, которое принимает значение 0.

Далее с помощью цикла *for* все модули элементов массива находящиеся перед первым нулевым элементом (*i_first*) и после последнего нулевого элемента (*last_zero*), суммируются и их сумма записываются в переменную *sum*.

С помощью оператора *return* функцией *sum_between* будет возвращено значение элемента *sum*.

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. | 101 0 1 2 3 4 0 5\n | Данные некорректны | Программа работает правильно |
| 2. | 0 1 0 3 4\n | 1 | Программа работает правильно |
| 3. | 1 1 2 3 4 0\n | 4 | Программа работает правильно |
| 4. | 2 9 0 1 1 1 0 9\n | 3 | Программа работает правильно |
| 5. | 3 9 0 1 1 1 0 9\n | 18 | Программа работает правильно |
| 6. | 2 1 2 3 4 5 6 7 0 9 0 1 2 3 4 5 6 7\n | 9 | Программа работает правильно |
| 7. | 2 0 -10 -10 10 -10 0 2\n | 40 | Программа работает правильно |
| 8. | 3 0 1 0 0 9 0 228\n | 228 | Программа работает правильно |

Выводы.

В ходе работы были изучены основные управляющие конструкции языка Си.

Разработана программа, выполняющая считывание исходных с помощью функции `scanf()` и цикла `while(){}` в переменную *option* и массив *array[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 результата определённой функции в зависимости от входного управляющего значения *option*:

- если *command* = 0 — выводится результат функции *int index_first_zero(array)*;
- если *command* = 1 — выводится результат функции *int index_last_zero(array,size)*;
- если *command* = 2 — выводится результат функции *int sum_between(array,size)()*;
- если *command* = 3 — выводится результат функции *int sum_before_and_after(array,size)*;

Если значение *command* 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_zero.h"
#include "index_last_zero.h"
#include "sum_between.h"
#include "sum_before_and_after.h"

int main(){

    int command;
    int array[100], size=0, index=0;

    scanf("%d", &command);

    while ((index < 100)&&(getchar() != '\n')){
        scanf("%d,", &array[size]);
        size += 1;
        index += 1;
    }

    switch (command) {
        case 0: printf("%d\n", index_first_zero(array));
                break;
        case 1: printf("%d\n", index_last_zero(array,size));
                break;
        case 2: printf("%d\n", sum_between(array,size));
                break;
        case 3: printf("%d\n", sum_before_and_after(array,size));
                break;
        default: printf("Данные некорректны\n");
                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 array[100]){
    int index=0;
    while ((array[index]) != 0) index += 1;
    return index;
}
```

Название файла: index_last_zero.c

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

int index_last_zero(int array[100], int size){
    int index=size-1;
    while ((array[index]) != 0) index -= 1;
    return index;
}
```

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 array[100], int size){
    int last_zero = index_last_zero(array,size);
    int i = index_first_zero(array);
    int sum=0;
    for (;i<=last_zero; i++){
        sum += abs(array[i]);
    }
    return sum;
}
```

Название файла: sum_after_and_before.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 array[100], int size){
    int sum=0;
```

```

    int i_last = index_last_zero(array, size);
    int i_first = index_first_zero(array);
    for (int k=0; k<i_first; k++){
        sum += abs(array[k]);
    }
    for (int k=i_last; k<size; k++){
        sum += abs(array[k]);
    }
    return sum;
}

```

Название файлов:

index_first_zero/index_last_zero/sum_between/sum_before_and_after(.c)

```

int index_first_zero(int array[100]);
int index_last_zero(int array[100], int size);
int sum_between(int array[100], int size);
int sum_before_and_after(int array[100], int size);

```

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 -std=c99 -c menu.c
index_first_zero.o: index_first_zero.c
        gcc -std=c99 -c index_first_zero.c
index_last_zero.o: index_last_zero.c
        gcc -std=c99 -c index_last_zero.c
sum_between.o: sum_between.c
        gcc -std=c99 -c sum_between.c
sum_before_and_after.o: sum_before_and_after.c
        gcc -std=c99 -c sum_before_and_after.c
clean:

```

```
rm *.o menu
```