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: Условия, циклы, оператор switch

Студент гр. 7381 _____ Лукашев Р.С.

Преподаватель _____ Берленко Т.А.

Санкт-Петербург

2017

Цель работы.

Ознакомиться с базовыми типами данных языка С, операциями над ними и выражениями.

Изучить функции стандартной библиотеки для ввода/вывода.

Научиться использовать условия, циклы, оператор switch.

Ознакомиться с простейшей отладкой на С.

Создать на основе полученных знаний проект.

Основные теоретические положения.

В языке С имеется несколько основных типов:

- char - один байт. Обычно используется для хранения символов
- int - целое число
- float - вещественное число
- double - вещественное число двойной точности

Также, есть ряд квалификаторов:

- short - короткое
- long - длинное
- unsigned - беззнаковое

Влияние спецификаторов размера типа (short, long) зависит от конкретной архитектуры.

Специального типа данных для хранения символьных строк в языке С нет. Для хранения строк используются массивы типа char, в которых после последнего символа строки находится нулевой символ '\0'.

Массив - структура данных, в виде набора элементов одного типа, расположенные в памяти друг за другом.

В языке С присутствуют привычные арифметические операции: +, -, *, / и операция %.

Дело в том, что операция деления (/) над целыми числами выполняется с отбрасыванием дробной части. Операция % позволяет получить остаток от деления (и допустима только над целыми).

Операции отношения

>=, >, ==, !=, <=, <

Логические операции

- ! - логическое НЕ

- && - логическое И
- || - логическое ИЛИ

Логические выражения вычисляются "ленивым" образом слева направо. Таким образом, как только становится ясным результат выражения, вычисление его аргументов прекращается.

В языке С предусмотрены специальные операции увеличения (++) и уменьшения (--). Есть префиксная и постфиксная их версии: отличие их в том, что префиксная операция выполняется до использования переменной в выражении, а постфиксная — после.

Если в выражениях встречаются операнды различных типов, то они преобразуются к общему типу в соответствии с небольшим набором правил. Автоматически производятся только преобразования, имеющие смысл, как, например, преобразование целого в плавающее в выражениях типа f+i. Выражения же, лишённые смысла, такие как использование переменной типа float в качестве индекса, запрещены.

В языке С существует возможность оперировать отдельными битами (применимо только к целым типам данных).

Для этого есть следующие логические операции:

- & - побитовое И
- | - побитовое ИЛИ
- ^ - побитовое исключающее ИЛИ
- << - побитовый сдвиг влево
- >> - побитовый сдвиг вправо
- ~ - дополнение (унарная операция)

Наиболее часто используемые функции для ввода и вывода: printf и scanf. Они позволяют использовать генерировать и интерпретировать форматные строки.
Printf:

*int printf (const char * format, arg1, arg2, ...argN);*

преобразует , определяет формат и печатает свои аргументы в стандартный поток вывода под управлением строки *format.* Управляющая строка содержит два типа объектов: обычные символы, которые просто копируются в выходной поток, и спецификации преобразований, каждая из которых вызывает преобразование и печать очередного аргумента printf. Каждая спецификация преобразования начинается с символа % и заканчивается символом преобразования.
Функция возвращает количество успешно выведенных символов.
Scanf:

*int scanf (const char * format, arg1, arg2, ...argN);*
читает символы из стандартного ввода, интерпретирует их в соответствии с форматом, указанном в аргументе control, и помещает результаты в остальные аргументы. Управляющий аргумент описывается ниже; другие аргументы, каждый из которых должен быть указателем, определяют, куда следует поместить соответствующим образом преобразованный ввод. Управляющая строка обычно содержит спецификации преобразования, которые используются для непосредственной интерпретации входных последовательностей.
Функция возвращает количество успешно считанных аргументов.

Операторный блок - несколько операторов, сгруппированные в единый блок с помощью фигурных скобок
{ [<оператор 1>...<оператор N>] }

Условный оператор:

if (<выражение>) <оператор 1> [else <оператор 2>]

Если *выражение* интерпретируется как истина, то *оператор1* выполняется. Может иметь необязательную ветку else, путь выполнения программы пойдет в случае если выражение ложно. В языке С любое ненулевое выражение расценивается как истина.

Оператор множественного выбора

switch(<выражение>){
case <константное выражение 1>: <операторы 1>
…
case <константное выражение N>: <операторы N>
[default: <операторы>]
}

Выполняет поочередное сравнение выражения со списком константных выражений. При совпадении, выполнение программы начинается с соответствующего оператора. В случае, если совпадений не было, выполняется необязательная ветка default. Важно помнить, что операторы после первого совпадения будут выполняться далее один за другим. Чтобы этого избежать, следует использовать оператор break.

Цикл с предусловием

while (<выражение>) <оператор>

На каждой итерации цикла происходит вычисление выражения и если оно истинно, то выполняется тело цикла.

Цикл с постусловием.

do <оператор> while <выражение>;

На каждой итерации цикла сначала выполняется тело цикла, а после вычисляется выражение. Если оно истинно — выполняется следующая итерация.

Цикл со счетчиком.

for ([<начальное выражение>]; [<условное выражение>]; [<выражение приращения>]) <оператор>

Условием продолжения цикла как и в цикле с предусловием является некоторое выражение, однако в цикле со счетчиком есть еще 2 блока — начальное выражение, выполняемое один раз перед первым началом цикла и выражение приращения, выполняемое после каждой итерации цикла. Любая из трех частей оператора for может быть опущена.
Оператор break — досрочно прерывает выполнение цикла.
Оператор continue — досрочный переход к следующей итерации цикла.

Зачастую программы могут успешно компилироваться, но при этом работать не так как программист того ожидает, либо аварийно завершаться. Чтобы устранить проблему необходимо прежде всего ответить на вопрос - в какой строке исходного кода происходит неправильное поведение. Наиболее простым способом является логгирование - добавление в программу специальных вызовов printf, которые протоколируют ход ее работы.

printf("%s, %s, %d: ваше сообщение\n", __FILE__, __func__, __LINE__);

Здесь __FILE__ , __func__ , __LINE__ - специальные макросы, которые заменяются препроцессором на имя полный путь к файлу, название функции и номер строки.

При активном использовании логов неизбежно возникает следующая проблема - при достаточно интенсивном выводе на консоль становится сложно выделить важную информацию. Для решения этой проблемы существует стандартный механизм - вывод цветных строк в командную строку.
Указание на то, какой цвет использовать, делается с помощью добавления специального кода в выводимую на консоль строку. Код цвета представляет собой команду для терминала - "переключи текущий цвет на цвет N". Существует также специальный код для возврата к цвету по умолчанию - "\033[0m" . Данную команду необходимо использовать после любых манипуляций с цветом для того, чтобы не повлиять на корректность отображения вывода других приложений в данной сессии терминала.
Кодирование цветов осуществляется следующим образом:

\033[STYLE;BACKGROUND_COLOR;FOREGROUND_COLORm

где STYLE это стиль отображения (0 - обычный, 1 - полужирный, 4 - подчеркнутый, 9 - зачеркнутый), BACKGROUND_COLOR и - FOREGROUND_COLOR коды цвета из таблицы Color table для фона и текста соответственно.
При этом, можно по желанию пропускать один или несколько этих атрибутов, например .
\033[STYLE;BACKGROUND_COLORm
\033[STYLE;FOREGROUND_COLORm

Ход работы.

В текущей директории был создан проект с make-файлом. Главная цель приводит к сборке проекта. Файл, который реализующий главную функцию, называется menu.c; исполняемый файл - menu. Определение каждой функции расположено в отдельном файле, название файлов указано в скобках около описания каждой функции.

Была реализована функция-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.

В зависимости от значения, функция выводит следующее:

0 : индекс первого чётного элемента. (index_first_even.c)
1 : индекс последнего нечётного элемента. (index_last_odd.c)
2 :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 :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 выводится строка "Данные некорректны".

Выводы.

Были изучены базовые типы данных языка С, операции с ними, функции стандартной библиотеки языка С для ввода/вывода, условия, циклы, оператор switch. Произошло ознакомление с простейшей отладкой. На основе полученных знаний создан проект.

Исходный код проекта.

- Файл «Makefile»:

```
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
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
	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
	gcc -c sum_before_even_and_after_odd.c
menu.o: menu.c
	gcc -c menu.c
```

- Файл «menu.c»:

```
#include <stdio.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;
 char c;
 scanf("%d", &N);
 int a[100];
 int i = 0;
        while (c!='\n'){
         scanf("%d%c", &a[i++], &c);
        }
 long command = 0;
        switch(N){
         case 0:
                command = (long)index_first_even(a,i); break;
         case 1:
                command = (long)index_last_odd(a,i); break;
         case 2:
                command = sum_between_even_odd(a,i); break;
         case 3:
                command = sum_before_even_and_after_odd(a,i); break;
```

```
        default:
                command = -1;
        }
 if(command == -1){
 printf("Данные некорректны\n");
 return 0;
 }
 printf("%ld\n", command);
 return 0;
}
```

- Файл «index_first_even.h»:

```
int index_first_even(int* a, int i);
```

- Файл «index_first_even.c»:

```
#include "index_first_even.h"

int index_first_even(int* a, int i){
 int j = 0;
  while(j<i)
   if(a[j++]%2==0)
    return --j;
 return -1;
}
```

- Файл «index_last_odd.h»:

```
int index_last_odd(int* a, int i);
```

- Файл «index_last_odd.c»:

```
#include "index_last_odd.h"

int index_last_odd(int* a, int i){
 while(i>0)
  if(a[--i]%2)
   return i;
 return -1;
}
```

- Файл «sum_between_even_odd.h»:

```
long sum_between_even_odd(int* a, int i);
```

- Файл «sum_between_even_odd.c»:

```
#include <stdlib.h>
#include "index_first_even.h"
#include "index_last_odd.h"

long sum_between_even_odd(int* a, int i){
 int first = index_first_even(a,i);
 int last = index_last_odd(a,i);
 long sum = 0;
```

```
  if((first!=-1) && (last!=-1)){
   for(int j = first; j < last; j++){
    sum+=abs(a[j]);
   }
   return sum;
  }
return -1;
}
```

- Файл «sum_before_even_and_after_odd.h»:

```
int sum_before_even_and_after_odd(int* a, int i);
```

- Файл «sum_before_even_and_after_odd.c»:

```
#include <stdlib.h>
#include "index_first_even.h"
#include "index_last_odd.h"

long sum_before_even_and_after_odd(int* a, int i){
 int first = index_first_even(a,i);
 int last = index_last_odd(a,i);
 long sum = 0;
  if((first != -1) && (last != -1)){
   for(int j = 0; j < first; j++) sum+=abs(a[j]);
   for(int k = i - 1; k >= last; k--) sum+=abs(a[k]);
   return sum;
  }
 return -1;
}
```