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

ОТЧЕТ

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

Студент гр. 9303 _____ Максимов Е.А.

Преподаватель _____ Чайка К.В.

Санкт-Петербург
2019

Цель работы.

1. Изучить структуру программы на языке программирования С;
2. Научиться использовать функции, массивы элементов и основные управляющие конструкции – условные операторы, счётные операторы, операторы цикла.

Задание.

Вариант лабораторной работы №1.

Реализуйте программу, 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*);

1: индекс последнего отрицательного элемента. (*index_last_negative*);

2: Найти произведение элементов массива, расположенных от первого отрицательного элемента (включая элемент) и до последнего отрицательного (не включая элемент). (*multi_between_negative*);

3: Найти произведение элементов массива, расположенных до первого отрицательного элемента (не включая элемент) и после последнего отрицательного (включая элемент) (*multi_before_and_after_negative*);

иначе необходимо вывести строку "Данные некорректны".

Основные теоретические положения.

Оператор *if–else* используется при необходимости сделать выбор. Синтаксис имеет вид:

```
if (выражение) <оператор-1>
else
     <оператор-2>
```

где часть *else* является необязательной. Сначала вычисляется выражение: если оно "истинно" (отличное от 0), то выполняется блок «оператор-1». Если оно

ложно, и есть часть с *else*, то вместо блока «оператор-1» выполняется блок «оператор-2», в противном случае программа не выполняет ни один из блоков.

Цикл – последовательность операторов, которая может выполняться более одного раза.

Цикл можно объявить тремя способами.

1) оператором *while*:

```
while (выражение) <оператор>
```

Сначала вычисляется значение выражения. Если его значение отлично от нуля, то выполняется оператор и выражение вычисляется снова. Этот цикл продолжается до тех пор, пока значение выражения не станет нулем, после чего выполнение программы продолжается с места после оператора.

2) оператором *for*:

```
for (выражение 1; выражение 2; выражение 3) <оператор>
```

Сначала выполняется выражение 1, затем вычисляется значение выражения 2. Если его значение отлично от нуля, то выполняется оператор и выражение 3, и выражение 2 вычисляется снова. Этот цикл продолжается до тех пор, пока значение выражения 2 не станет нулем, после чего выполнение программы продолжается с места после оператора. Любое из трёх выражений может быть опущено. В случае отсутствия всех выражений цикл становится бесконечным.

3) оператором *do while*:

```
do <оператор> while (выражение)
```

В отличие от *while*, сначала выполняется оператор, а затем вычисляется выражение. Если его значение отлично от нуля, то выполняется оператор и выражение вычисляется снова.

Оператор *switch* вызывает передачу управления к одному из нескольких операторов, в зависимости от значения выражения. Оператор имеет синтаксис:

```
switch (выражение) <оператор>
```

Оператор обычно является составным. Любой оператор внутри этого оператора может быть помечен одним или более вариантным префиксом *case*, имеющим форму:

```
case константстное_выражение:
```

где константное выражение должно иметь тип *int* (integer). Никакие две константы в одном и том же переключателе не могут иметь одинаковое значение.

Кроме того, может присутствовать самое большее один операторный префикс вида *default*. В случае если ни один из вариантов не подходит, то управление передается оператору, помеченному этим префиксом.

Выполнение работы.

В программе использовались следующие переменные:

1. Целого типа (integer):
    a. *array[20]* – массив целых чисел с размерностью 20;
    b. *function* – переменная, которая определяет последующие действия программы в операторе switch;
    c. *number* – определяет количество элементов в массиве, которое заранее не известно;
    d. *i* – локальная переменная-счётчик;
    e. *index_last_negative* – переменная, которая определяет последний отрицательный элемент массива;
    f. *multi_between_negative* – переменная, которая определяет произведение элементов массива, расположенных между крайними отрицательными числами (но не включая первый отрицательный элемент массива);
    g. *multi_before_and_after_negative* – переменная, которая определяет произведение элементов массива, расположенных не между крайними отрицательными числами (но включая последний отрицательный элемент массива).
2. Символьного типа (char):

a. *symbol* – используется для завершения ввода элементов массива после прочтения символа «*n*».

В программе реализованы следующие функции:

1. Функция int *array_read(int *array)* принимает на вход незаполненный массив. Функция считывает элементы массива со строки до тех пор, пока не считает символ переноса строки (не более 20 по условию), и считает количество записанных элементов массива. Функция заполняет массив и возвращает количество элементов.
2. Функция *int index_first_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит первый отрицательный элемент массива. Функция возвращает индекс первого отрицательного элемента массива.
3. Функция *int index_last_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит последний отрицательный элемент массива. Функция возвращает индекс последнего отрицательного элемента массива.
4. Функция *int multi_between_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит произведение элементов массива между первым отрицательным элементом массива (не включая его) и последним отрицательным элементом массива (включая его). Функция возвращает данное произведение элементов массива.
5. Функция *int multi_before_and_after_negative_function(int *array, int *number)* принимает на вход массив целых чисел и их количество. Функция находит произведение элементов массива от первого до первого отрицательного элемента массива (не включая его) и от последнего отрицательного элемента массива (включая его) до последнего. Функция возвращает данное произведение элементов массива.

Разработанный программный код см. в приложении А.

Результаты тестирования.

№ п/п	Входные данные	Выходные данные	Комментарии
1.	0 1 2 3 -4 5 -6 7 -8 9 10	3	Тест пройден
2.	0 1 -2 3 -4 5	1	Тест пройден
3.	0 1 2 3 4 5 6 7 8 9 -10 11 12 13 14 15 16 17 18 19 20	9	Тест пройден
4.	1 1 -2 3 -4 5 6 7 -8 9 10	7	Тест пройден
5.	1 1 -2 3 -4 5	3	Тест пройден
6.	2 0 2 3 -4 -1 6 7 -8 9 10	168	Тест пройден
7.	2 1 2 3 -4 5 6 7 -8 9 10	-840	Тест пройден
8.	2 1 -2 3 -4 5	-6	Тест пройден
9.	2 -5 5 -5 5 -5 5 -5	-15625	Тест пройден
10.	3 0 2 3 -4 5 6 7 -8 9 10	0	Тест пройден
11.	3 1 -2 3 -4 5	-20	Тест пройден
12.	3 -5 5 -5 5 -5 5 -5	-5	Тест пройден
13.	5 1 2 3 -4 5 6 7 -8 9 10	Данные некорректны	Тест пройден
14.	8 -1 -2 -3 -4 -5	Данные некорректны	Тест пройден
15.	00009 -9 -9 -9 -9	Данные некорректны	Тест пройден

Выводы.

В ходе лабораторной работы были изучены: структура программы языка программирования С, массивы элементов, функции и основные управляющие конструкции языка *if-else, switch, while, for, do while*.

Была разработана программа, которая считывает с клавиатуры исходные данные, формирует массив из исходных данных и выполняет определённые операции над этими данными. Для обработки команд пользователя использовались условные операторы *if-else*, операторы циклов *for*.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: main.c

```
#include<stdio.h>

int array_read(int *array){
      char symbol;
      for(int i=0; i<20; i++){
          scanf("%i%c", &array[i], &symbol);
          if(symbol == '\n'){
              i++;
              return i;
          }
      }
}

int index_first_negative_function(int *array, int *number){
      for (int i=0; i<*number; i++){
          if (array[i]<0){
              return i;
          }
      }
}

int index_last_negative_function(int *array, int *number){
      int index_last_negative=-1;
      for (int i=0; i<*number; i++){
          if (array[i]<0){
              index_last_negative=i;
          }
      }
      return index_last_negative;
}

int multi_between_negative_function(int *array, int *number){
      int multi_between_negative=1;
      for(int i=index_first_negative_function(array, number);
i<index_last_negative_function(array, number); i++){
          multi_between_negative=multi_between_negative*array[i];
      }
      return multi_between_negative;
}

int multi_before_and_after_negative_function(int *array, int *number){
      int multi_before_and_after_negative=1;
      for(int i=0; i<index_first_negative_function(array, number); i++){
           multi_before_and_after_negative=multi_before_and_after_negativ
      e*array[i];
      }
      for(int i=index_last_negative_function(array, number); i<*number;
i++){
           multi_before_and_after_negative=multi_before_and_after_negativ
      e*array[i];
      }
      return multi_before_and_after_negative;
```

```
}

int main(){
      int function;
      scanf("%i", &function);
      int array[20];
      int number=array_read(array);
      switch (function){
          case 0:
              printf("%i\n", index_first_negative_function(array,
&number));
              break;
          case 1:
              printf("%i\n", index_last_negative_function(array,
&number));
              break;
          case 2:
              printf("%i\n", multi_between_negative_function(array,
&number));
              break;
          case 3:
              printf("%i\n",
multi_before_and_after_negative_function(array, &number));
              break;
          default:
              printf("Данные некорректны\n");
      }
}
```