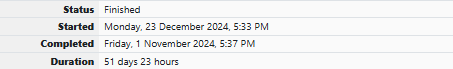
**ASSESSMENT 03**

**ADMISSION ELIGIBILITY**

****

Write a C program to find the eligibility of admission for a professional course based on the following criteria:

Marks in Maths >= 65

Marks in Physics >= 55

Marks in Chemistry >= 50

Or

Total in all three subjects >= 180

**Sample Test Cases**

**Test Case 1**

**Input**

 70   60   80

**Output**

The candidate is eligible

**Test Case 2**

**Input**

50   80   80

**Output**

The candidate is eligible

**Test Case 3**

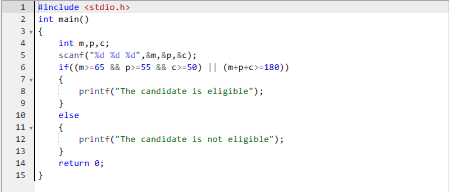
**Input**

50   60   40

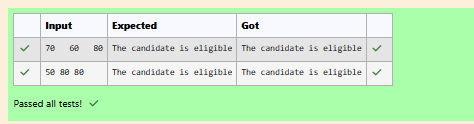
**Output**

The candidate is not eligible

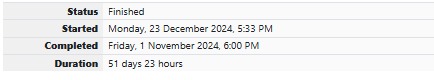
**SOURCE CODE**

****

**OUTPUT**

****

**CALUCULATOR**

****

Complete the calculator program with Basic operations (+, -, \*, /, %) of two numbers using switch statement.

**Sample Test Cases**

**Test Case 1**

**Input**

45

45

+

**Output**

Result: 45 + 45 = 90.000000

**Test Case 2**

**Input**

56

8

%

**Output**

Result: 56 % 8 = 0.000000

**Test Case 3**

**Input**

50

70

$

**Output**

Invalid operation.

Result: 50 $ 70 = 0.000000

**Test Case 4**

**Input**

5

2

/

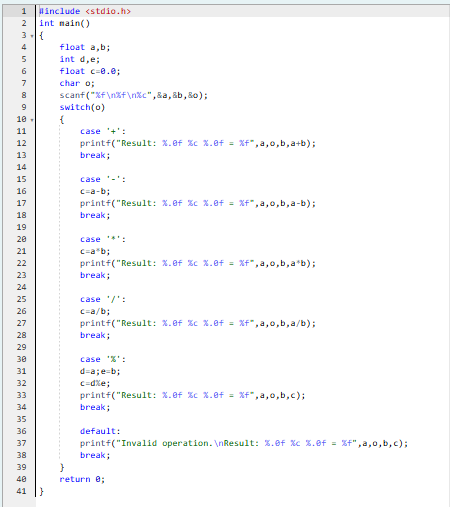
**Output**

Result: 5 / 2 = 2.500000

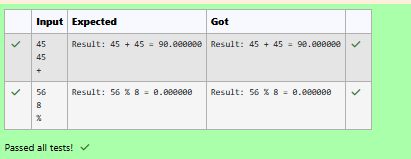
**For example:**

| **Input** | **Result** |
| --- | --- |
| 5  2  / | Result: 5 / 2 = 2.500000 |

**SOURCE CODE**

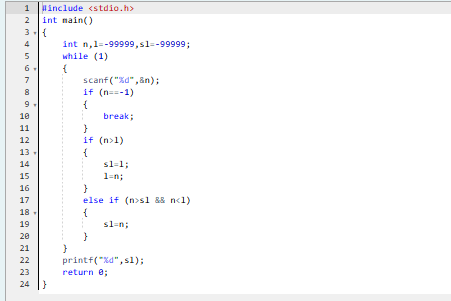
****

**OUTPUT**

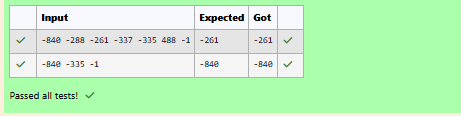
****

**FINDING THE SECOND LARGEST ELEMENT**

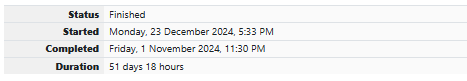
**SOURCE CODE**

****

**OUTPUT**

****

**TRIANGLE – SMALLEST SIDE**

****

The lengths of the sides of a triangle X, Y and Z are passed as the input. The program must print the smallest side as the output.

**Input Format:**

The first line denotes the value of X.  
The second line denotes the value of Y.  
The third line denotes the value of Z.

**Output Format:**

The first line contains the length of the smallest side.

**Boundary Conditions:**

1 <= X <= 999999  
1 <= Y <= 999999  
1 <= Z <= 999999

**Example Input/Output 1:**

Input:  
40  
30  
50

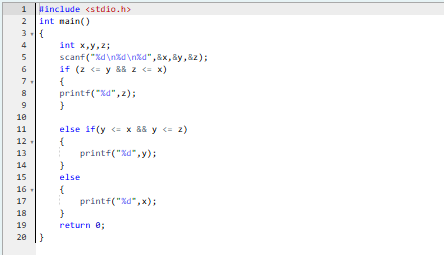
Output:  
30

**Example Input/Output 2:**

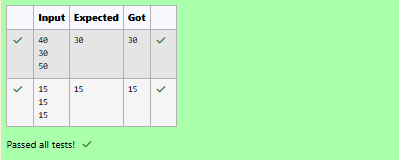
Input:  
15  
15  
15

Output:  
15

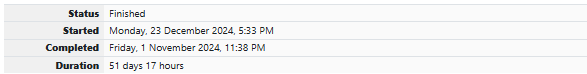
**SOURCE CODE**

****

**OUTPUT**

****

**FORMAL AND ACTUAL ARGUMENTS**

****

An argument is an expression which is passed to a function by its caller in order for the function to perform its task. It is an expression in the comma-separated list bound by the parentheses in a function call expression.  
  
A function may be called by the portion of the program with some arguments and these arguments are known as actual arguments (or) original arguments.  
  
Actual arguments are local to the particular function. These variables are placed in the **function declaration** and **function call**. These arguments are defined in the **calling function**.  
  
The parameters are variables defined in the function to receive the arguments.  
  
Formal parameters are those parameters which are present in the **function definition**.  
  
**Formal parameters** are available only with in the specified function. Formal parameters belong to the **called function**.  
  
**Formal parameters** are also the local variables to the function. So, the formal parameters are occupied memory when the function execution starts and they are destroyed when the function execution completed.  
  
Let us consider the below example:

#include <**stdio.h**>

**int** add(int, int);

int main()   
{

**int** a = 10, b = 20;

printf("Sum of two numbers = %d\n", add(a, b)); // variables a, b are called actual arguments

return 0;  
}

**int** add(**int** x, **int** y)   
{

// variables x, y are called formal parameters  
 return(x + y);  
}

In the above code whenever the function call add(a, b) is made, the execution control is transferred to the function definition of add().  
  
The values of actual arguments a and b are copied in to the formal arguments x and y respectively.

The formal parameters x and y are available only with in the function definition of add(). After completion of execution of add(), the control is transferred back to the main().

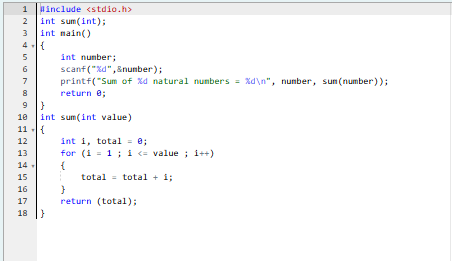
See & retype the below code which will demonstrate about formal and actual arguments.

#include <stdio.h>  
  
int sum(int);  
  
int main()  
{  
    int number;  
    scanf("%d", &number);  
    printf("Sum of %d natural numbers = %d\n", number, sum(number));  
    return 0;  
}  
  
int sum(int value)  
{  
    int i, total = 0;  
    for (i = 1; i <= value; i++)  
    {  
        total = total + i;  
    }  
    return(total);  
}

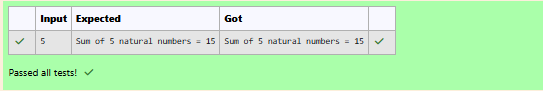
**For example:**

| **Input** | **Result** |
| --- | --- |
| 5 | Sum of 5 natural numbers = 15 |

**SOURCE CODE**

****

**OUTPUT**

****

**LOCAL AND GLOBAL VARIABLES**

A local variable is declared inside a function.  
  
A **local variable** is visible only inside their function, only statements inside function can access that local variable.  
  
**Local variables** are declared when the function execution started and local variables gets destroyed when control exits from function.  
  
Let us consider an example:

#include <**stdio.h**>

**void** test();

**int** main()   
{

**int** a = 22, b = 44;

test();

printf("Values in main() function a = %d and b = %d\n", a, b);

return 0;  
}

**void** test()   
{

**int** a = 50, b = 80;

printf("Values in test() function a = %d and b = %d\n", a, b);

}

In the above code we have 2 functions main() and test(), in these functions local variables are declared with same variable names a and b but they are different.  
  
**Operating System** calls the main() function at the time of execution. the **local variables** with in the main() are created when the main() starts execution.  
  
when a call is made to test() function, first the control is transferred from main() to test(), next the local variables with in the test() are created and they are available only with in the test() function.  
  
After completion of execution of test() function, the local variables are destroyed and the control is transferred back to the main() function.

See & retype the below code which will demonstrate about local variables.

#include <stdio.h>

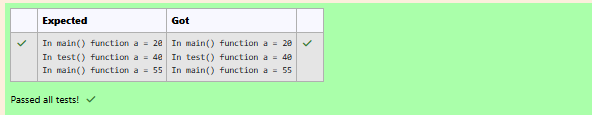
void test();  
  
int main()  
{  
    int a = 9, b = 99;  
    test();  
    printf("Values in main() function a = %d and b = %d\n", a, b);  
    return 0;  
}  
  
void test()  
{  
    int a = 5, b = 55;  
    printf("Values in test() function a = %d and b = %d\n", a, b);  
}

**For example:**

| **Result** |
| --- |
| Values in test() function a = 5 and b = 55  Values in main() function a = 9 and b = 99 |

**SOURCE CODE**

**OUTPUT**

****

**QUESTION 2**

Global variables are declared outside of any function.  
  
A **global variable** is visible to any every function and can be used by any piece of code.  
  
Unlike **local variable**, **global variables** retain their values between function calls and throughout the program execution.  
  
Let us consider an example:

#include <**stdio.h**>

**int** a = 20; // Global declaration

**void** test();

**int** main()   
{

printf("In main() function a = %d\n", a); // Prints 20

test();

a = a + 15; // Uses global variable

printf("In main() function a = %d\n", a); // Prints 55  
 return 0;

}

**void** test()   
{

a = a + 20; // Uses global variable

printf("In test() function a = %d\n", a); // Prints 40

}

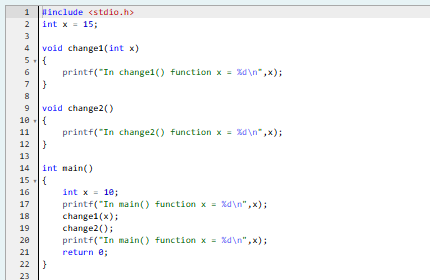
In the above code the **global variable** a is declared outside of all the functions. So, the variable a can be accessed in every function.  
  
**Operating System** calls the main() function at the time of execution. the variable a has no local declaration, so it access the global variable a.  
  
In test() function also there is no local declaration of variable a, the variable a gets access from the global.  
  
The global variables are destroyed only after completion of execution of entire program.  
  
See & retype the below code which will demonstrate about global variables.

#include <stdio.h>  
  
int a = 20;  
  
void test();  
  
int main()  
{  
    printf("In main() function a = %d\n", a);  
    test();  
    a = a + 15;  
    printf("In main() function a = %d\n", a);  
    return 0;  
}  
  
void test()  
{  
    a = a + 20;  
    printf("In test() function a = %d\n", a);  
}

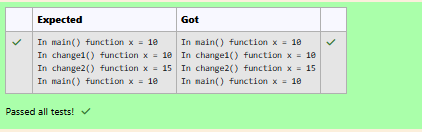
**For example:**

| **Result** |
| --- |
| In main() function a = 20  In test() function a = 40  In main() function a = 55 |

**SOURCE CODE**

****

**OUTPUT**

****