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

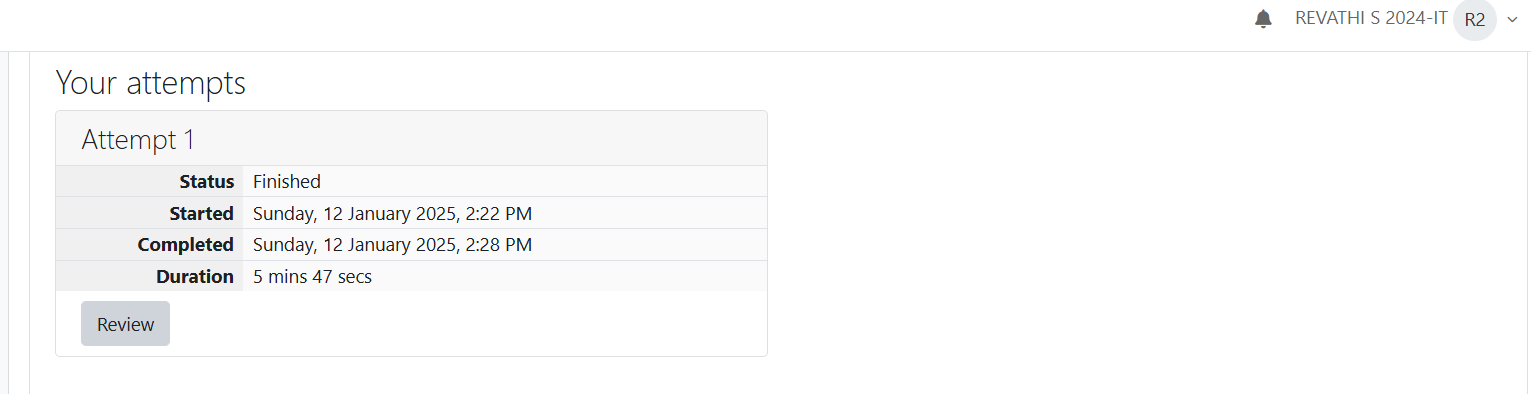
**Test Case 3**

**Input**

50   60   40

**Output**

The candidate is not eligible



A screenshot of a computer program

Description automatically generated

You are given a sequence of integers as input, terminated by a -1. (That is, the input integers may be positive, negative or 0. A -1 in the input signals the end of the input.)

-1 is not considered as part of the input.

Find the second largest number in the input. You may not use arrays.

**Sample Test Cases**

**Test Case 1**

**Input**

-840 -288 -261 -337 -335 488 -1

**Output**

-261

**Test Case 2**

**Input**

-840 -335 -1

**Output**

-840

A close-up of a computer screen

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

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

Answer:(penalty regime: 0 %)

A close-up of a computer screen

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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 |

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

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 |

Answer:(penalty regime: 0 %)

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**Local variables are declared and used inside a function (or) in a block of statements.  
  
Local variables are created at the time of function call and destroyed when the function execution is completed.  
  
Local variables are accessible only with in the particular function where those variables are declared.  
  
Global variables are declared outside of all the function blocks and these variables can be used in all functions.  
  
Global variables are created at the time of program beginning and reside until the end of the entire program.  
  
Global variables are accessible in the entire program.  
  
If a local and global variable have the same name, then local variable has the highest precedence to access with in the function.  
  
Let us consider an example:**

**#include <stdio.h>**

**void change();  
int x = 20; // Global Variable x**

**int main()   
{**

**int x = 10; // Local Variable x**

**change();**

**printf("%d", x); // The value 10 is printed  
 return 0;**

**}**

**void change()   
{**

**printf("%d", x); // The value 20 is printed**

**}**

**In the above code the global and local variables have the same variable name x, but they are different.  
  
In main() function the local variable x is only accessed, so it prints the value 10.  
  
In change() function the variable x is not declared locally so it access global variable x, so it prints 20.**

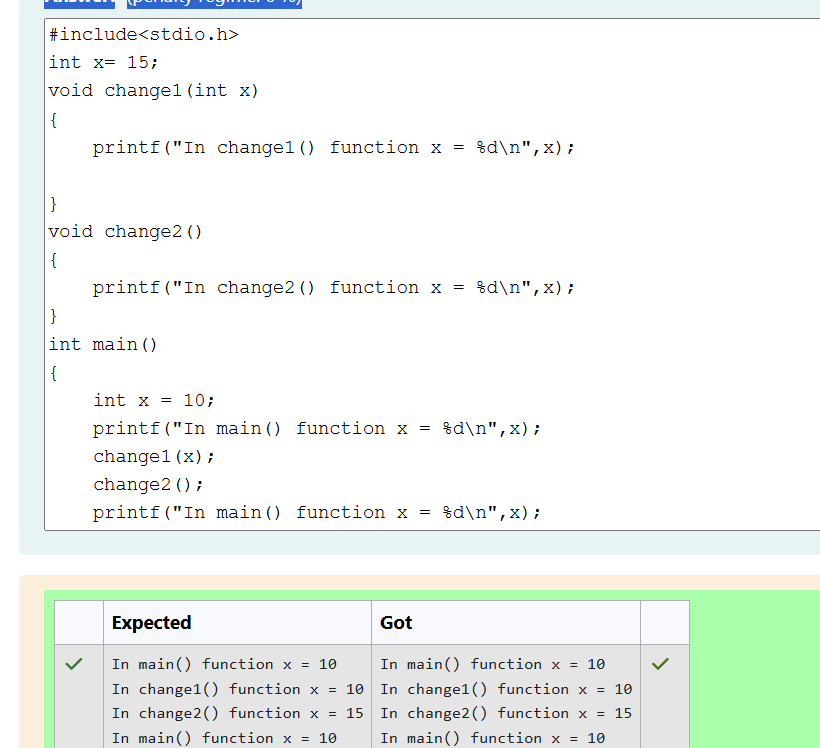
**See & retype the below code which will demonstrate about local and global variables.**

**#include <stdio.h>  
  
int x = 15;  
  
void change1(int x)  
{  
    printf("In change1() function x = %d\n", x);  
}  
  
void change2()  
{  
    printf("In change2() function x = %d\n", x);  
}  
  
int main()  
{  
    int x = 10;  
    printf("In main() function x = %d\n", x);  
    change1(x);  
    change2();  
    printf("In main() function x = %d\n", x);  
    return 0;  
}**

**For example:**

| **Result** |
| --- |
| **In main() function x = 10**  **In change1() function x = 10**  **In change2() function x = 15**  **In main() function x = 10** |

**Answer:(penalty regime: 0 %)**

****

**All the C functions can be called either with arguments or without arguments in a C program. These functions may or may not return values to the calling function.  
  
Depending on the arguments and return values functions are classified into 4 categories.**

1. **Function without arguments and without return value**
2. **Function with arguments and without return value**
3. **Function without arguments and with return value**
4. **Function with arguments and with return value**

**When a function has no arguments, it does not receive any data from the calling function.  
  
Similarly, when a function does not return a value, the calling function does not receive any data from the called function.  
  
In effect, there is no data transfer between the calling function and the called function in the category function without arguments and without return value.  
  
Let us consider an example of a function without arguments and without return value:**

**#include <stdio.h>**

**void india\_capital(void);**

**int main()   
{**

**india\_capital();  
 return 0;**

**}**

**void india\_capital()   
{**

**printf("New Delhi is the capital of India\n");**

**}**

**In the above sample code the function void india\_capital(void); specifies that the function does not receive any arguments and does not return any value to the main() function.  
  
Identify the below errors and correct them.**

**For example:**

| **Result** |
| --- |
| **New Delhi is the capital of India** |

**A screenshot of a computer

Description automatically generated**