

Question **1**

Correct

Marked out of  
1.00

 [Flag question](#)

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

Marks in Maths  $\geq 65$

Marks in Physics  $\geq 55$

Marks in Chemistry  $\geq 50$

Or

Total in all three subjects  $\geq 180$

**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

Answer: (penalty regime: 0 %)

```
1 #include<stdio.h>
2 int main()
3 {
4     int math,physics,chem,total;
5     scanf("%d %d %d",&math,&physics,&chem);
6     total=math+physics+chem;
7     if((math>=65 && physics>=55 && chem>=50)||((total>=180))
8     {
9         printf("The candidate is eligible");
10    }
11    else
12    {
13        printf("The candidate is not eligible");
14    }
15    return 0;
16 }
```

	Input	Expected	Got	
✓	70 60 80	The candidate is eligible	The candidate is eligible	✓
✓	50 80 80	The candidate is eligible	The candidate is eligible	✓


Passed all tests! ✓

Finish review

Question **1**

Correct

Marked out of  
1.00

 [Flag question](#)

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

```

1 #include<stdio.h>
2 #include<limits.h>
3 int main()
4 {
5     int n;
6     int largest=INT_MIN,second_largest=INT_MIN;
7     while(1)
8     {
9         scanf("%d",&n);
10        if(n ==-1)
11        {
12            break;
13        }
14        if(n>largest)
15        {
16            second_largest=largest;
17            largest=n;
18        }
19        else if(n>second_largest&& n!=largest)
20        {
21            second_largest=n;
22        }
23    }
24    if(second_largest !=INT_MIN)
25    {
26        printf("%d",second_largest);
27    }
28
29 }

```

	Input	Expected	Got	
✓	-840 -288 -261 -337 -335 488 -1	-261	-261	✓
✓	-840 -335 -1	-840	-840	✓

Passed all tests! ✓

Question **1**

Correct

Marked out of  
1.00

 [Flag question](#)

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 %)

```
1 #include<stdio.h>
2 int main()
3 {
4     int X,Y,Z;
5     scanf("%d %d %d",&X,&Y,&Z);
6     int smallest=X;
7     if(Y<smallest)
8     {
9         smallest=Y;
10    }
11    if(Z<smallest)
12    {
13        smallest=Z;
14    }
15    printf("%d\n",smallest);
16
17 }
```

	Input	Expected	Got	
✓	40 30 50	30	30	✓
✓	15 15 15	15	15	✓

Passed all tests! ✓

Question 1

Correct

Marked out of

1.00

 Flag question

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 within 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



5	Sum of 5 natural numbers = 15
---	-------------------------------

Answer: (penalty regime: 0 %)

```
1 #include<stdio.h>
2 int sum(int);
3 int main()
4 {
5     int number;
6     scanf ("%d",&number);
7     printf("Sum of %d natural numbers = %d\n",number,sum(number));
8     return 0;
9 }
10 int sum(int value)
11 {
12     int i,total=0;
13     for(i=1;i<=value;i++)
14     {
15         total=total+i;
16     }
17
18     return(total);
19 }
```

	Input	Expected	Got	
✓	5	Sum of 5 natural numbers = 15	Sum of 5 natural numbers = 15	✓

Passed all tests! ✓

Question 1

Incorrect

Marked out of  
1.00 Flag question

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.

**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 %)

```
1 #include<stdio.h>
2 void test();
3 int main()
4 {
5     int a = 9,b = 99;
6     test();
7     printf("Values in main() function a = %d and b = %d\n",a , b);
8     return 0;
9 }
10 void test()
11 {
12     int a = 5, b = 55;
13     printf("Values in test() function a = %d and b = %d\n ",a,b);
14 }
```

	Expected	Got	
✗	Values in test() function a = 5 and b = 55 Values in main() function a = 9 and b = 99	Values in test() function a = 5 and b = 55 Values in main() function a = 9 and b = 99	✗

Your code must pass all tests to earn any marks. Try again.

Show differences

Question 2

Correct

Marked out of  
1.00[Flag question](#)

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;
```

```
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

```

1 #include<stdio.h>
2 int a = 20;
3 void test();
4 int main()
5 {
6     printf("In main() function a = %d\n", a);
7     test();
8     a = a + 15;
9     printf("In main() function a = %d\n", a);
10    return 0;
11 }
12 void test()
13 {
14     a = a + 20;
15     printf("In test() function a = %d\n",a);
16 }

```

	Expected	Got	
✓	In main() function a = 20 In test() function a = 40 In main() function a = 55	In main() function a = 20 In test() function a = 40 In main() function a = 55	✓

Passed all tests! ✓

Question 3

Incorrect

Marked out of  
1.00

[Flag question](#)

**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 within 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 within 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

```

1 #include<stdio.h>
2 int X = 15;
3 void change1(int X)
4 {
5     printf("In change1() function x = %d\n",X);
6 }
7 void change2()
8 {
9     printf("In change2 () function x = %d\n",X);
10 }
11 int main()
12 {
13     int x =10;
14     printf("In main() function X = %d\n",X);
15     change1(x);
16     change2( );
17     printf("In main() function x = %d\n",X);
18     return 0;
19 }

```

	Expected	Got	
✗	In main() function <b>x</b> = 10 ↵ In change1() function x = 10 ↵ In change2() function x = 15 ↵ In main() function x = 10	In main() function <b>X</b> = 15 ↵ In change1() function x = 10 ↵ In change2( <b>)</b> () function x = 15 ↵ In main() function x = 15	✗

Your code must pass all tests to earn any marks. Try again.

Hide differences

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

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

Reset answer

New Delhi is the capital of India

Answer: (penalty regime: 0 %)

Reset answer

```
1 #include <stdio.h>
2
3 void india_capital(void);
4
5 int main()
6 {
7     india_capital();
8     return 0;
9 }
10
11 void india_capital()
12 {
13     printf("New Delhi is the capital of India\n");
14 }
```

	Expected	Got	
✓	New Delhi is the capital of India	New Delhi is the capital of India	✓

Passed all tests! ✓

Question **2**

Correct

Marked out of  
1.00

🚩 [Flag question](#)

Write a **C** program to demonstrate functions without arguments and without return value.

Write the functions **print()** and **hello()**.

The output is:

```
...***...
Hello! REC
...***...
```

For example:

**Result**

```
...***...
Hello! REC
...***...
```

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

Reset answer

```
1 #include <stdio.h>
2
3 // Write the functions
4 void hello(void)
5 {
6     printf("Hello! REC\n");
7 }
8
9 int main()
10 {
11     printf("...***...\n");
12     hello();
13     printf("...***...");
14     return 0;
15 }
```

	Expected	Got	
✓	...***... Hello! REC ...***...	...***... Hello! REC ...***...	✓

Passed all tests! ✓

Question 3

Correct

Marked out of  
1.00 Flag question

When a function definition has **arguments**, it receives data from the calling function.

The **actual arguments** in the function call must correspond to the **formal parameters** in the function definition, i.e. the number of actual arguments must be the same as the number of formal parameters, and each actual argument must be of the same data type as its corresponding formal parameter.

The **formal parameters** must be valid variable names in the function definition and the **actual arguments** may be variable names, expressions or constants in the function call.

The variables used in actual arguments must be assigned values before the **function call** is made. When a function call is made, copies of the values of actual arguments are passed to the **called function**.

What occurs inside the function will have no effect on the variables used in the **actual argument** list. There may be several different calls to the same function from various places with a program.

Let us consider an example of a function with arguments and without return value:

```
#include <stdio.h>
void largest(int, int);
int main()
{
    int a, b;
    printf("Enter two numbers : ");
    scanf("%d%d" , &a, &b);
    largest(a, b);
    return 0;
}
void largest(int x, int y)
{
    if (x > y)
    {
        printf("Largest element = %d\n", x);
    }
    else
    {
        printf("Largest element = %d\n", y);
    }
}
```

```
    {  
        printf("Largest element = %d\n", y);  
    }  
}
```

In the above sample code the function `void largest(int, int);` specifies that the function receives two integer arguments from the **calling function** and does not return any value to the **called function**.

When the function call `largest(a, b)` is made in the `main()` function, the values of actual arguments `a` and `b` are copied in to the formal parameters `x` and `y`.

After completion of execution of `largest(int x, int y)` function, it does not return any value to the `main()` function. Simply the control is transferred to the `main()` function.

Fill in the missing code in the below program to find the largest of two numbers using **largest()** function.

**For example:**

Input	Result
27 18	Largest element = 27
13 17	Largest element = 17



Answer: (penalty regime: 0 %)

Reset answer

```
1 #include <stdio.h>
2
3 void largest(int, int);
4
5 int main()
6 {
7     int a, b;
8     scanf("%d%d", &a, &b);
9     largest(a,b); // Correct the code
10    return 0;
11 }
12
13 void largest(int x, int y)
14 {
15     // Correct the code
16     if (x>y)
17     {
18         // Correct the code
19         printf("Largest element = %d\n", x);
20     }
21     else
22     {
23         printf("Largest element = %d\n", y);
24     }
25 }
```

	Input	Expected	Got	
✓	27 18	Largest element = 27	Largest element = 27	✓
✓	13 17	Largest element = 17	Largest element = 17	✓

Passed all tests! ✓

Question 4

Correct

Marked out of  
1.00[Flag question](#)

Fill the missing code to understand the concept of a function with arguments and without return value.

**Note:** Take pi value as 3.14

The below code is to find the area of circle using functions.

**For example:**

Input	Result
11.23	Area of circle = 395.994476

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

Reset answer

```
1 #include <stdio.h>
2 #define pi 3.14
3
4 void area_circle(float);
5
6 int main()
7 {
8     float radius;
9     scanf("%f", &radius);
10    area_circle(radius);
11    return 0;
12 }
13
14 void area_circle(float radius)
15 {
16     //Correct the code
17     // Write the code to calculate the area of circle
18     float area=pi*radius*radius;
19     printf("Area of circle = %f\n", area);
20 }
```

	Input	Expected	Got	
✓	11.23	Area of circle = 395.994476	Area of circle = 395.994476	✓

Passed all tests! ✓

Question **5**

Correct

Marked out of  
1.00

🚩 Flag question

When a function has **no arguments**, it does not receive any data from the calling function.

When a function **return a value**, the calling function receives data from the called function.

Let us consider an example of a function without arguments and with return value:

```
#include <stdio.h>
int sum(void);
int main()
{
    printf("\nSum of two given values = %d\n", sum());
    return 0;
}
int sum() {
    int a, b, total;
    printf("Enter two numbers : ");
    scanf("%d%d", &a, &b);
    total = a + b;
    return total;
}
```

In the above sample code the function `int sum(void);` specifies that the function does not receive any arguments but return a value to the **calling function**.

Fill in the missing code in the below program to find sum of two integers.

**For example:**

Reset answer

```

1  #include <stdio.h>
2
3  int sum(void);
4
5  int main()
6  {
7      printf("Sum of two given values = %d\n", sum());
8      return 0;
9  }
10
11 int sum()
12 {
13     int a,b,total;
14     scanf("%d %d ",&a,&b);
15     total=a+b;
16     return total;
17     // Fill in the missing code
18     // Read two integers
19     // Find sum
20     // Return sum
21 }

```


	Input	Expected	Got	
✓	9 5	Sum of two given values = 14	Sum of two given values = 14	✓
✓	45 78	Sum of two given values = 123	Sum of two given values = 123	✓

Passed all tests! ✓

Question 

Correct

Marked out of  
1.00

 [Flag question](#)

When a **function definition** has arguments, it receives data from the calling function.

After taking some desired action, only one value will be returned from **called function** to **calling function** through the return statement.

If a function returns a value, the **function call** may appear in any expression and the returned value used as an operand in the evaluation of the expression.

Let us consider an example of a function with arguments and with return value:

```
#include <stdio.h>
int largest(int, int, int);
int main()
{
    int a, b, c;
    printf("Enter three numbers : ");
    scanf("%d%d%d" , &a, &b, &c);
    printf(" Largest of the given three numbers = %d\n", largest(a, b, c));
    return 0;
}
int largest(int x, int y, int z)
{
    if ((x > y) && (x > z))
    {
        return x;
    }
    else if (y > z)
    {
        return y;
    }
    else
    {
        return z;
    }
}
```

In the above sample code the function `int largest(int, int, int);` specifies that the function receives three values and returns a value to the **calling function**.

Fill in the missing code in the below program to find the largest of three numbers using **largest()** function.

For example:

For example:

Input	Result
99 49 29	Largest of the given three numbers = 99
45 67 35	Largest of the given three numbers = 67

Answer: (penalty regime: 0 %)

Reset answer

```
1 #include <stdio.h>
2
3 int largest(int, int, int);
4
5 int main()
6 {
7     int a, b, c;
8     scanf("%d%d%d", &a, &b, &c);
9     printf("Largest of the given three numbers = %d\n", largest(a,b,c)); // Correct the code
10    return 0;
11 }
12
13 int largest(int x, int y,int z)
14 {
15     // Correct the code
16     if ((x>y)&&(x>z))
17     {
18         // Correct the code
19         return x ; // Correct the code
20     }
21     else if (y>z)
22     {
23         // Correct the code
24         return y ; // Correct the code
25     }
26     else
27     {
28         return z; // Correct the code
29     }
30 }
```

	Input	Expected	Got	
✓	99 49 29	Largest of the given three numbers = 99	Largest of the given three numbers = 99	✓
✓	45 67 35	Largest of the given three numbers = 67	Largest of the given three numbers = 67	✓

Passed all tests! ✓

Question **7**

Correct

Marked out of  
1.00

🚩 [Flag question](#)

Fill in the missing code in the below code to understand about function with arguments and with return value.

The below code is to find the factorial of a given number using functions.

**For example:**

Input	Result
3	Factorial of a given number 3 = 6

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

Reset answer

```
1 #include <stdio.h>
2
3 int factorial(int number);
4
5 int main()
6 {
7     int number;
8     scanf("%d", &number);
9     printf("Factorial of a given number %d = %d\n", number, factorial(number));
10    return 0;
11 }
12
13 int factorial(int number)
14 {
15     int i, factorial = 1;
16     for (i=1;i<=number;i++)
17     {
18         // Write code to calculate the factorial of a given number
19         factorial=factorial*i;
20     }
21     // Write the return statement
22     return factorial;
23 }
```

	Input	Expected	Got	
✓	3	Factorial of a given number 3 = 6	Factorial of a given number 3 = 6	✓

Passed all tests! ✓



Question **8**

Correct

Marked out of  
1.00

 [Flag question](#)

Write a **C** program to demonstrate functions without arguments and with return value.

The below code is used to check whether the given number is a prime number or not.

Write the function **prime()**.

Sample Input and Output:

5

The given number is a prime number

**For example:**

Input	Result
5	The given number is a prime number
27	The given number is not a prime number
121	The given number is not a prime number
1	The given number is not a prime number

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

```
1  #include <stdio.h>
2
3  int prime(int num);
4
5  int main()
6  {
7      int num;
8      scanf("%d",&num);
9      if (prime(num) )
10     {
11         printf("The given number is a prime number\n");
12     }
13     else
14     {
15         printf("The given number is not a prime number\n");
16     }
17     return 0;
18 }
19
20 // Write the function prime()
21 int prime(int num)
22 {
23     if(num<=1)
24         return 0;
25     for(int i=2;i<=num/2;i++)
26     {
27         if(num%i==0)
28         {
29             return 0;
30         }
31     }
32     return 1;
33 }
34 }
35
```

	Input	Expected	Got	
✓	5	The given number is a prime number	The given number is a prime number	✓
✓	27	The given number is not a prime number	The given number is not a prime number	✓
✓	121	The given number is not a prime number	The given number is not a prime number	✓
✓	1	The given number is not a prime number	The given number is not a prime number	✓

Passed all tests! ✓

Finish review