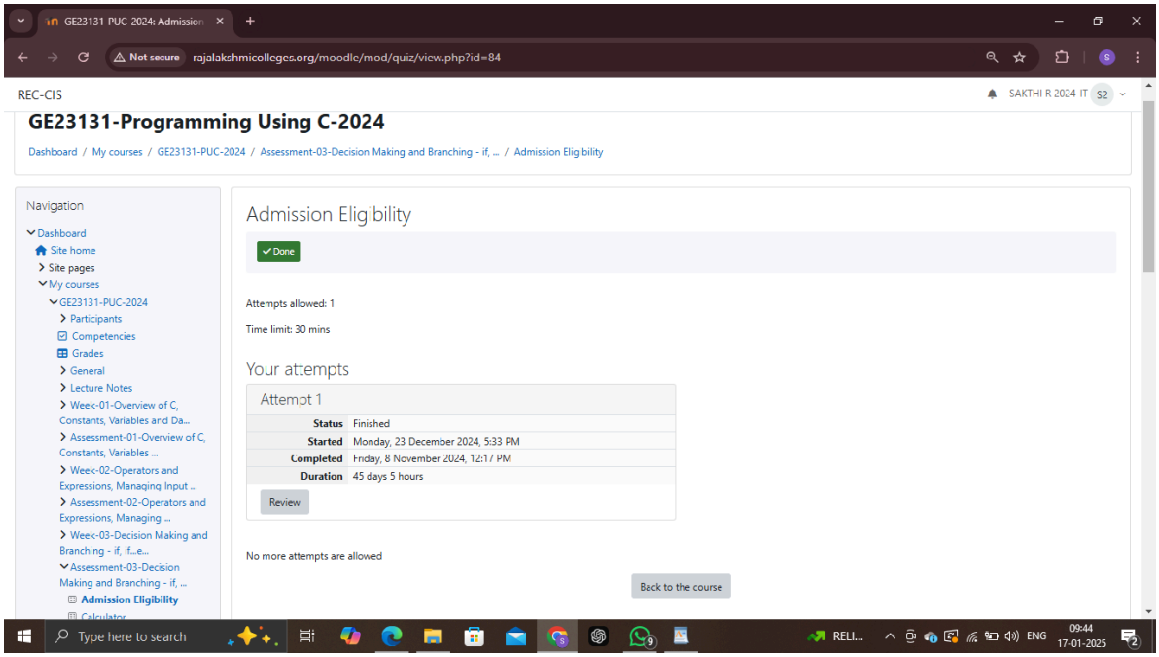


1)Write a C program to find the eligibility of admission for a professional course based on the following criteria:

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

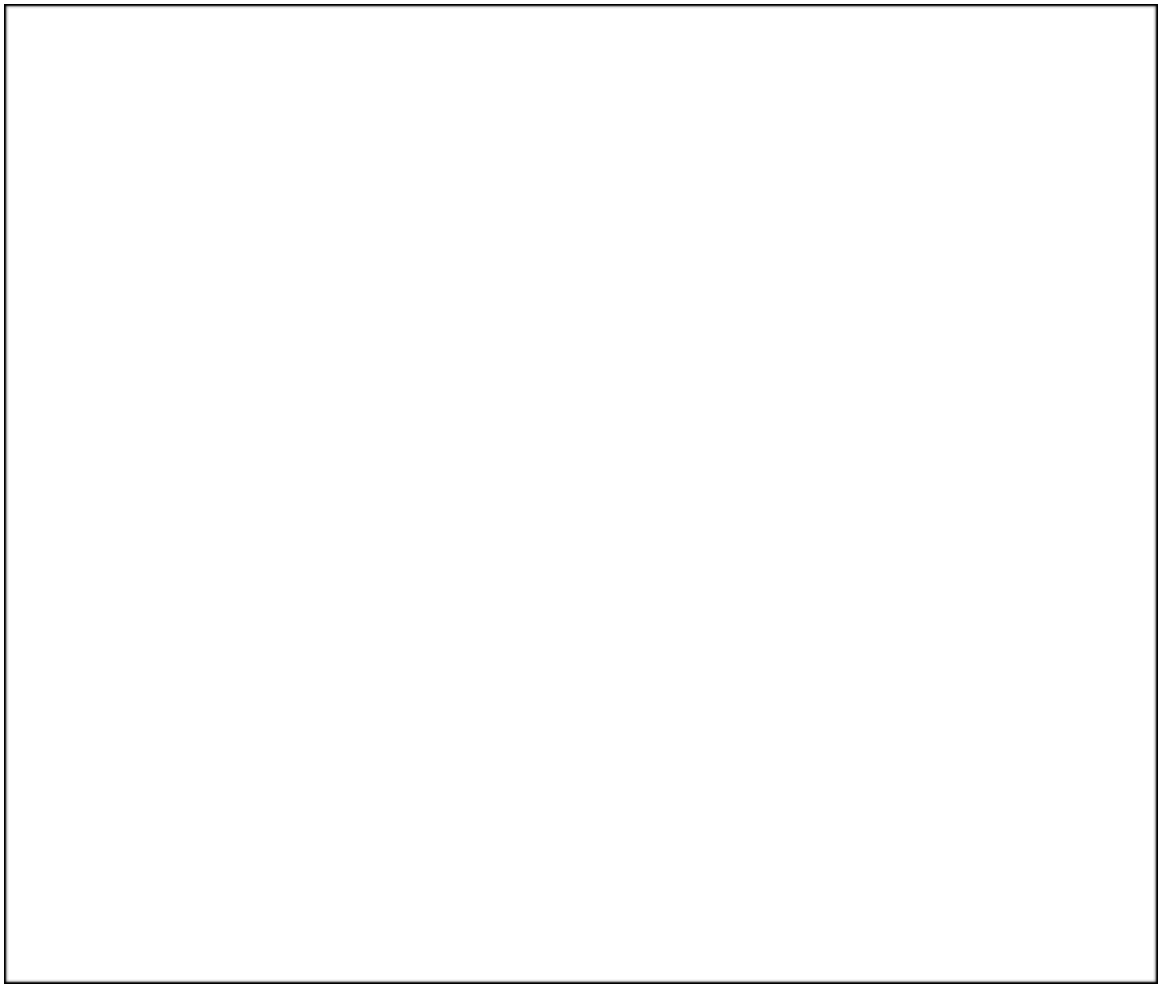
Test Case 3

Input

50 60 40

Output

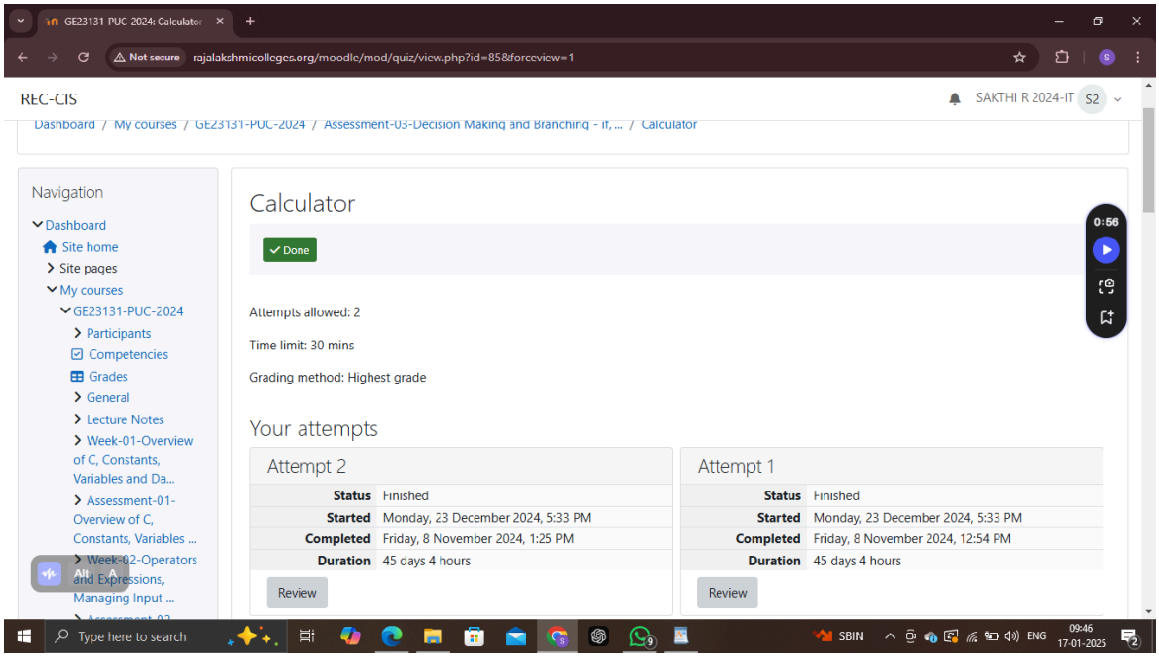
The candidate is not eligible



```c
#include<stdio.h>
int main()
{
    int a,b,c,tot;
    scanf("%d %d %d",&a,&b,&c);
    tot=a+b+c;
    if(a>=65&&b>=55 &&c>=50 )
    printf("The candidate is eligible");
    else if(tot>=180)
    printf("The candidate is eligible");
    else
    printf("The candidate is not eligible");
}
```

	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! ✓



2)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

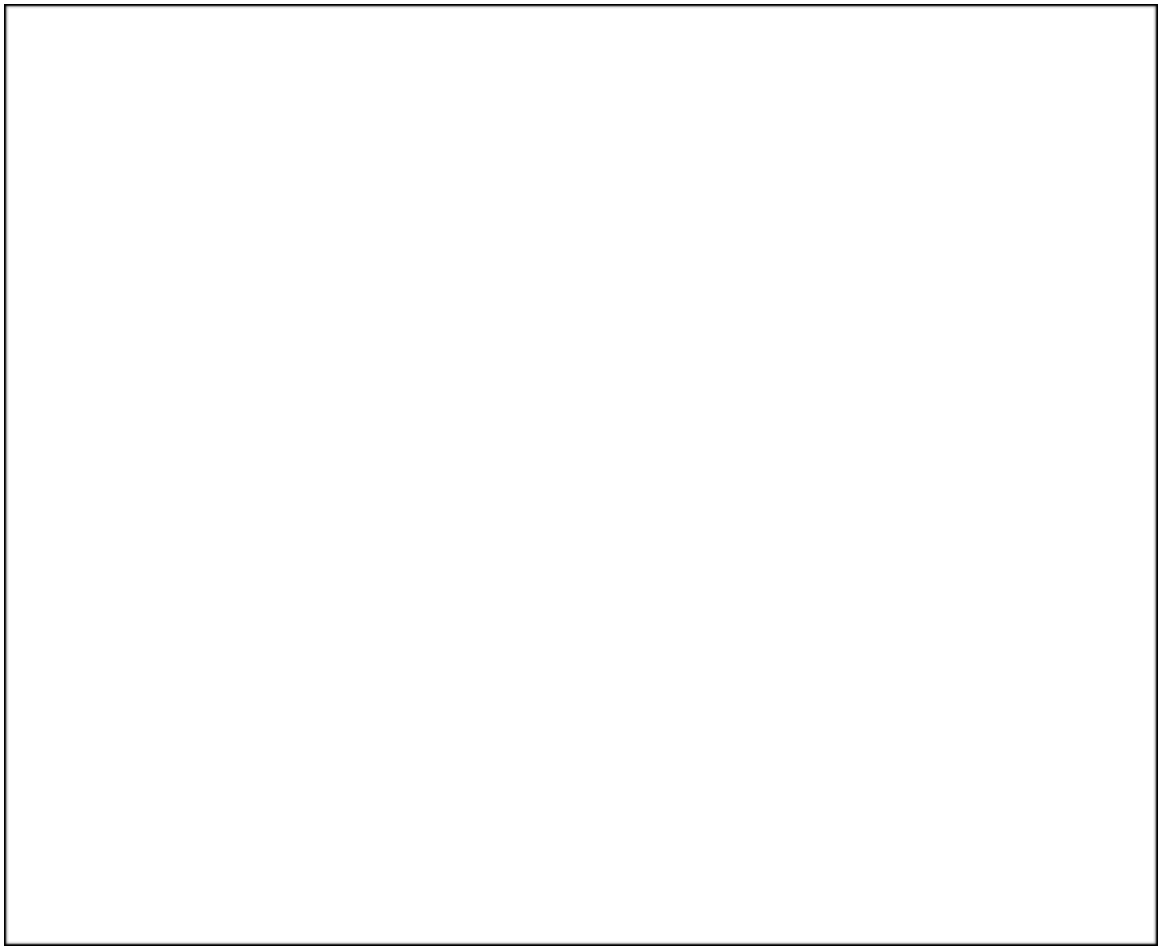
5

2

/

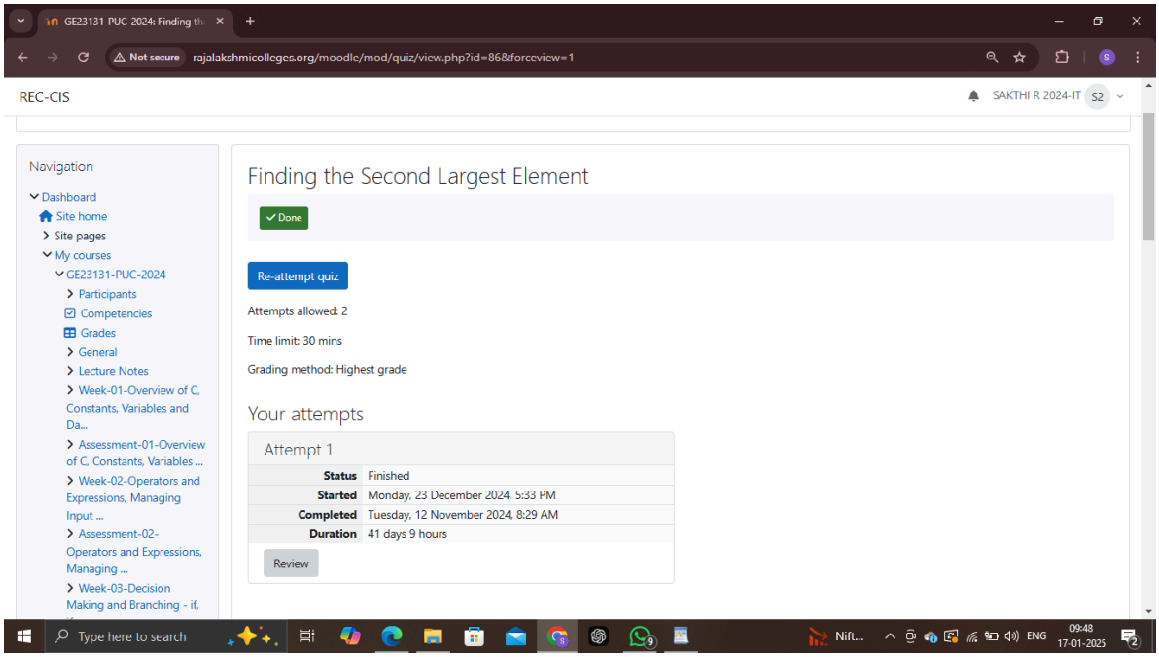
Output

Result: 5 / 2 = 2.500000



```c
#include<stdio.h>
int main()
{
    int a,b;
    char c;
    float r;
    scanf("%d %d %c",&a,&b,&c);
    switch(c)
    {
        case '+':r=a+b;printf("Result: %d + %d = %f ",a,b,r);break;
        case '-':r=a-b;printf("Result: %d - %d = %f ",a,b,r);break;
        case '*':r=a*b;printf("Result: %d * %d = %f ",a,b,r);break;
        case '/':r=a/b;printf("Result: %d / %d = %f ",a,b,r);break;
        case '%':r=a%b;printf("Result: %d %% %d = %f ",a,b,r);break;
        default:r=0.0;printf("Invalid operation.\n");printf("Result: %d %c %d = %f ",a,c,b,r);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	45 45 +	Result: 45 + 45 = 90.000000	Result: 45 + 45 = 90.000000	✓
✓	56 8 %	Result: 56 % 8 = 0.000000	Result: 56 % 8 = 0.000000	✓



3)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

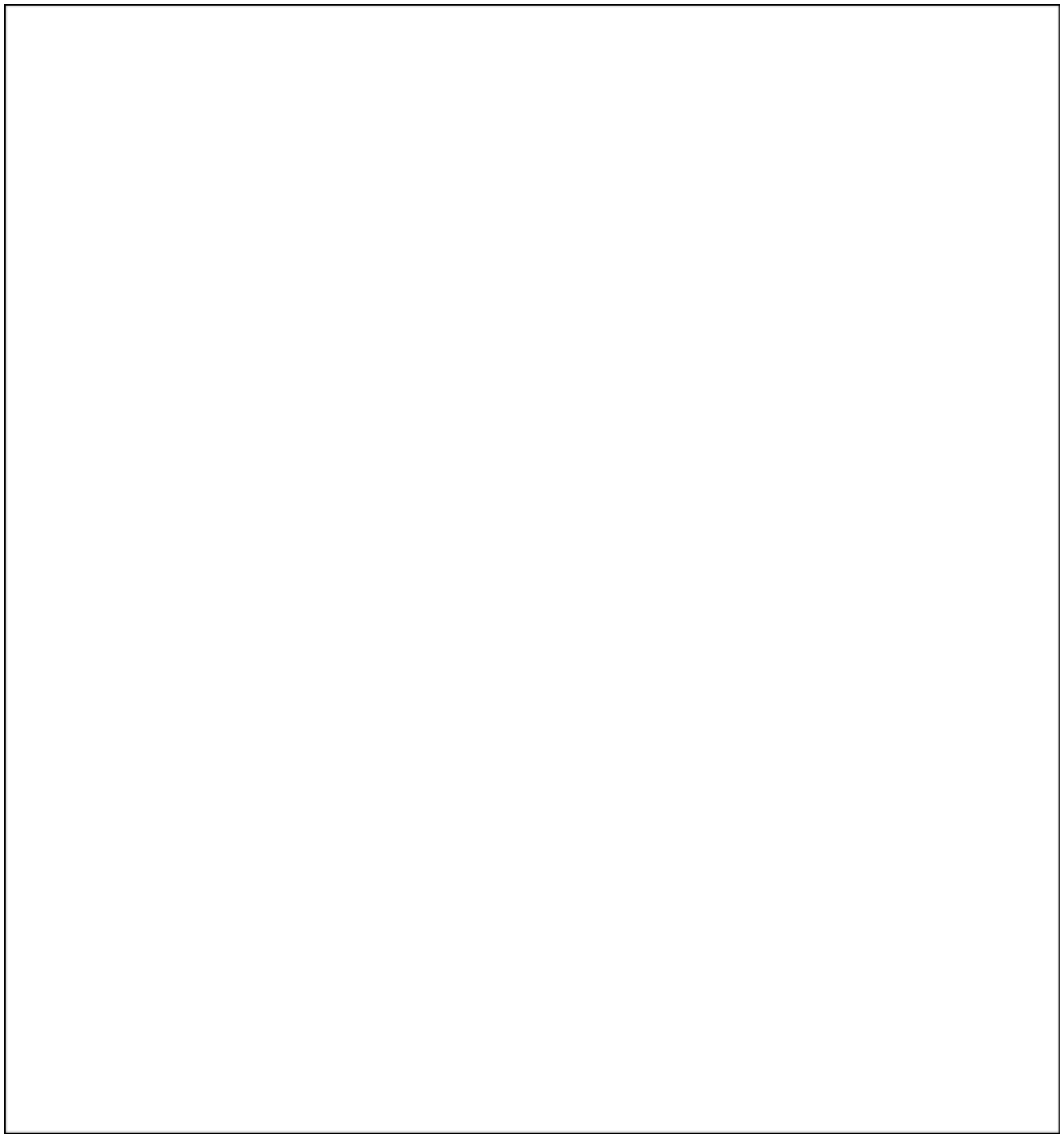
Test Case 2

Input

-840 -335 -1

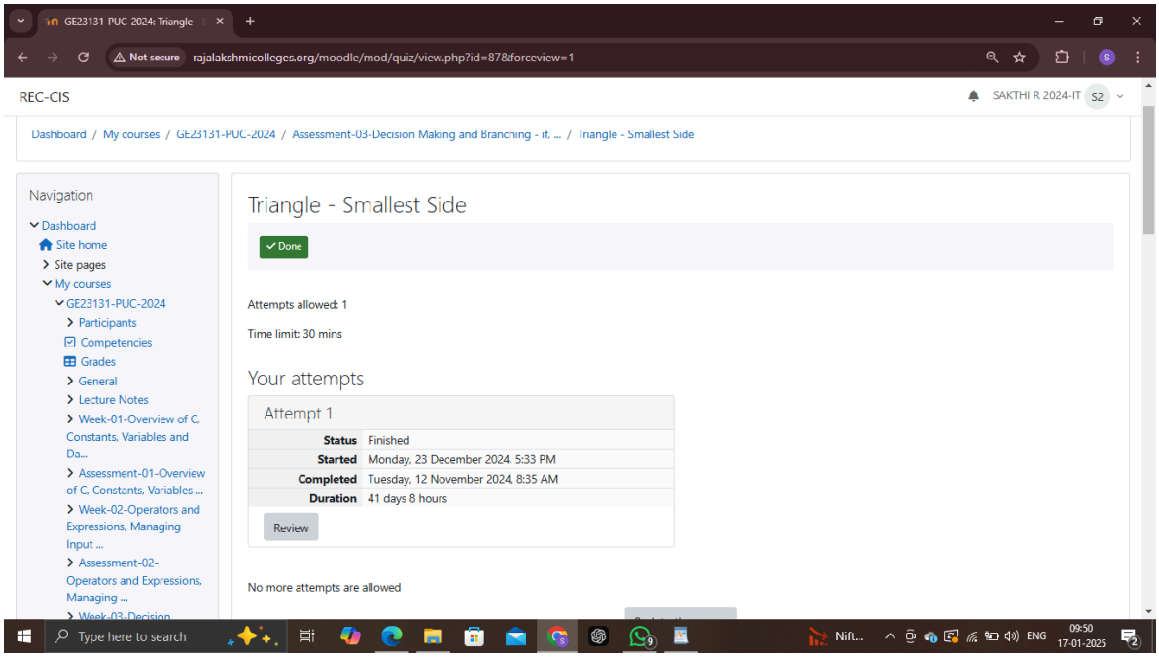
Output

-840



```c
#include<stdio.h>
#include<limits.h>
int main()
{
    int num;
    int lag=INT_MIN;
    int s1=INT_MIN;


    while(1){
    scanf("%d",&num);
    if(num==-1)
    break;
    if(num>lag)
    {s1=lag;
    lag=num;
    }
    else if(num<lag&&num>s1)
    s1=num;
    }
    if(s1==INT_MIN)
    printf("no");
    else
    printf("%d",s1);
    return 0;

}
```

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



4)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:

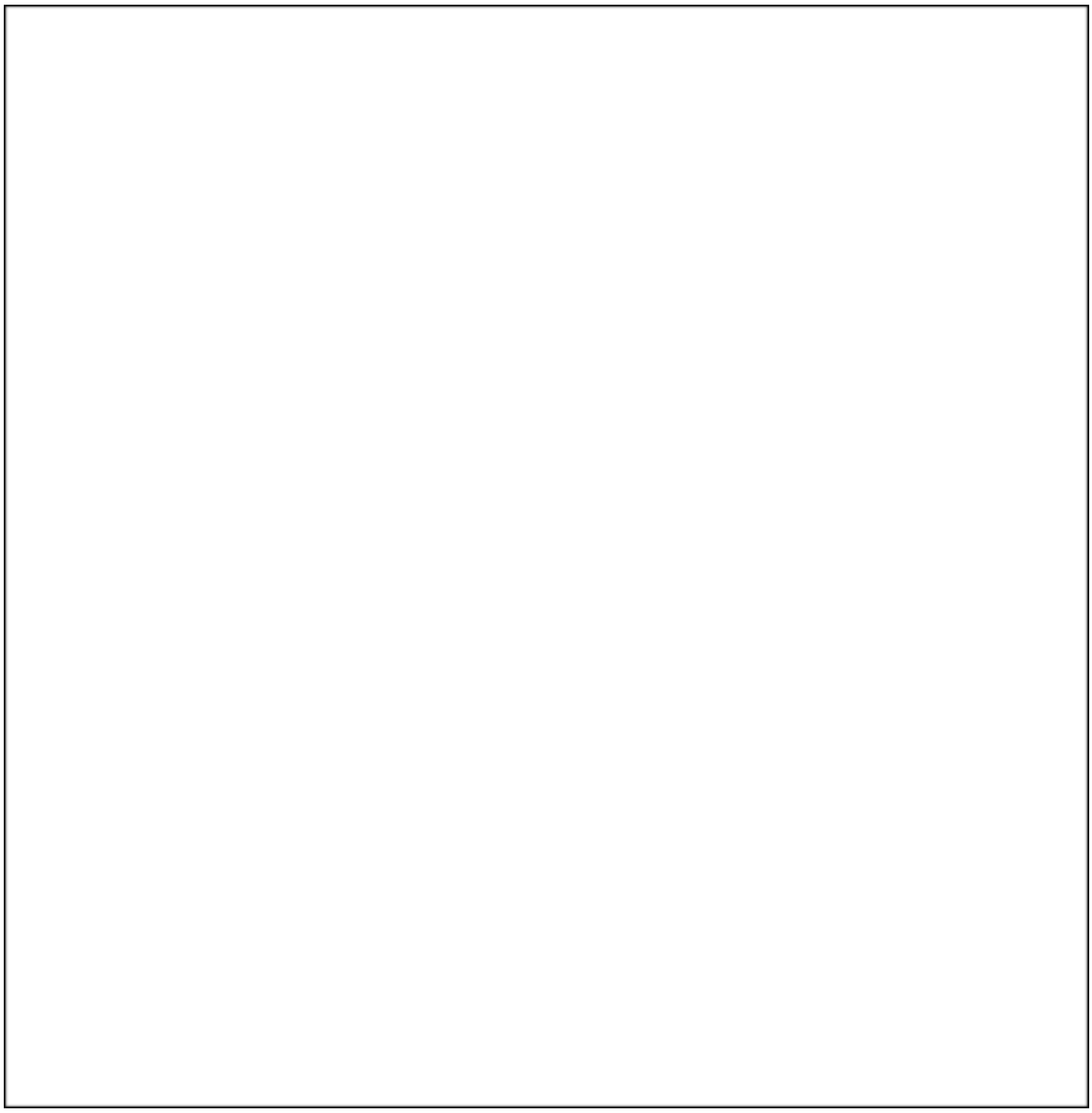
15

15

15

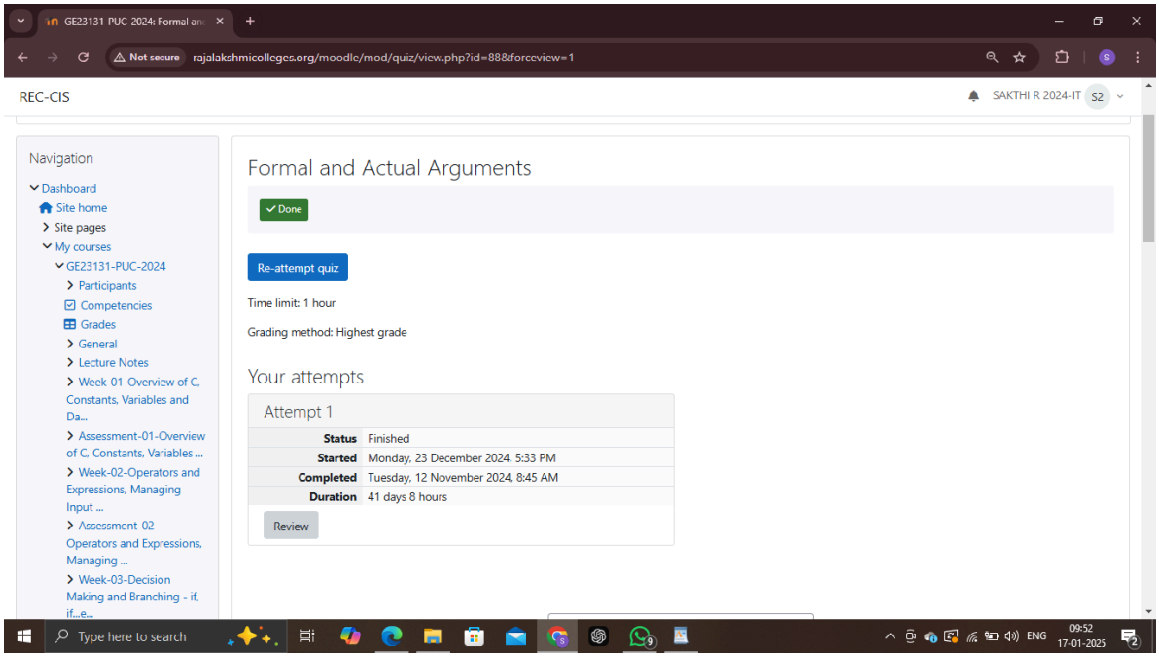
Output:

15



```c
#include<stdio.h>
int main()
{
    int a,b,c;
    scanf("%d %d %d",&a,&b,&c);
    int small=a;
    if(b<small)
    printf("%d",b);
    else if(c<small)
    printf("%d",c);
    else
    printf("%d",a);
}
```

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



5)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().

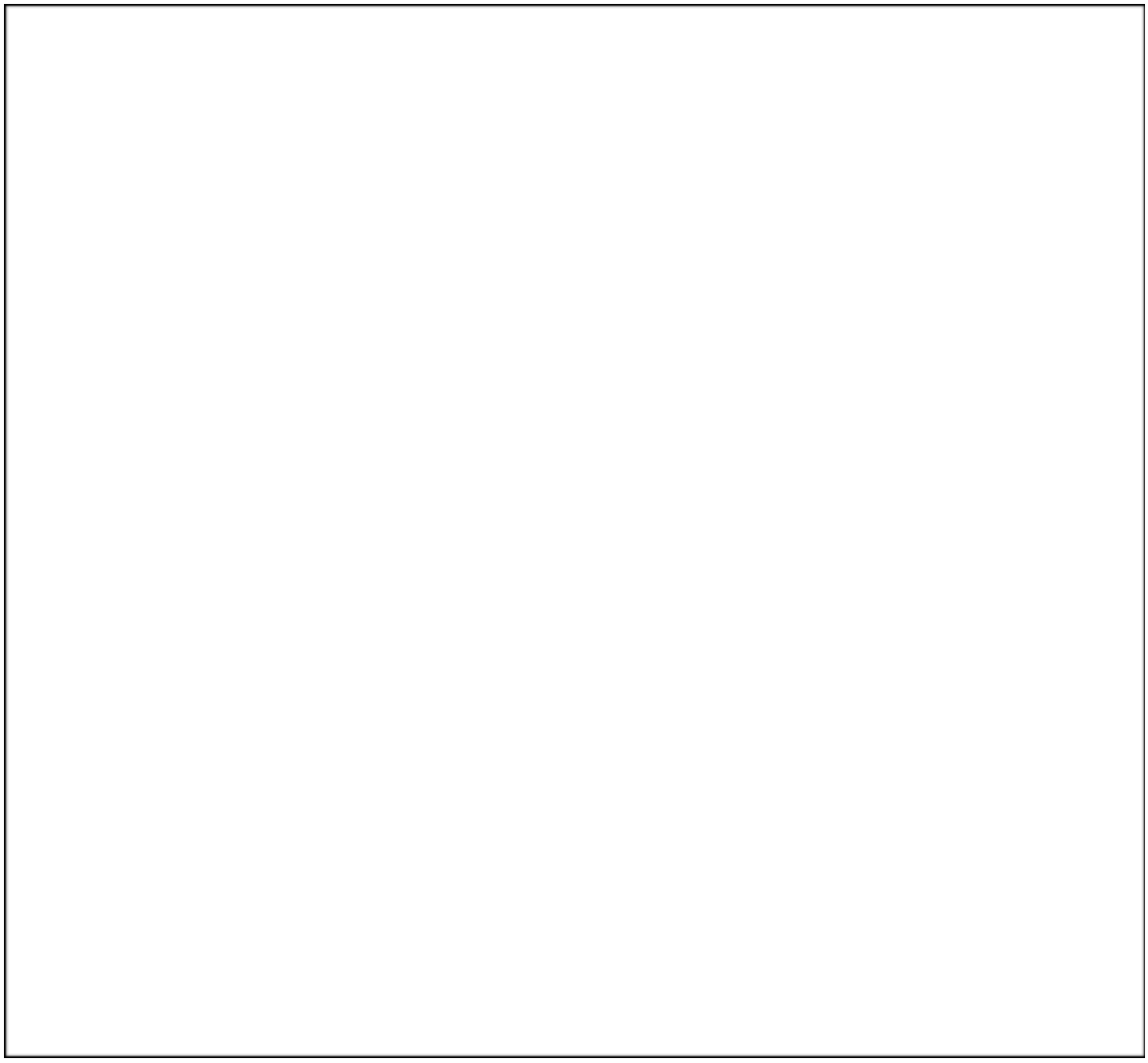
The values of actual arguments a and b are copied in to the formal arguments x and y respectively.

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);
}
```

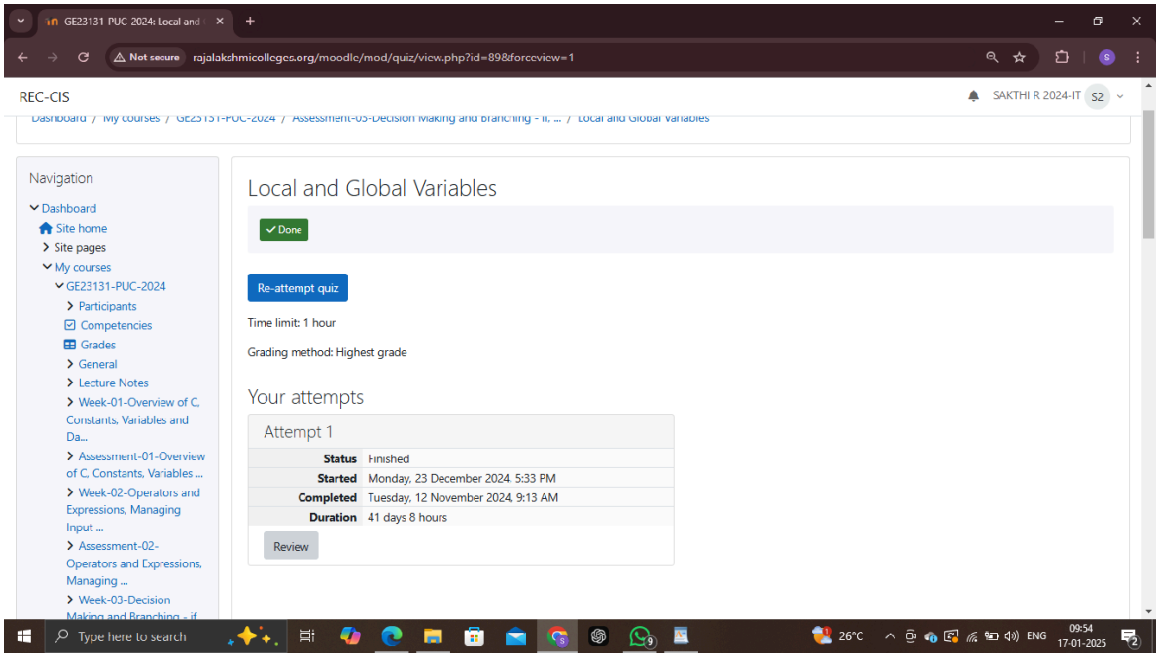


Answer: (penalty regime: 0 %)

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

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

Passed all tests! ✓



6)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.

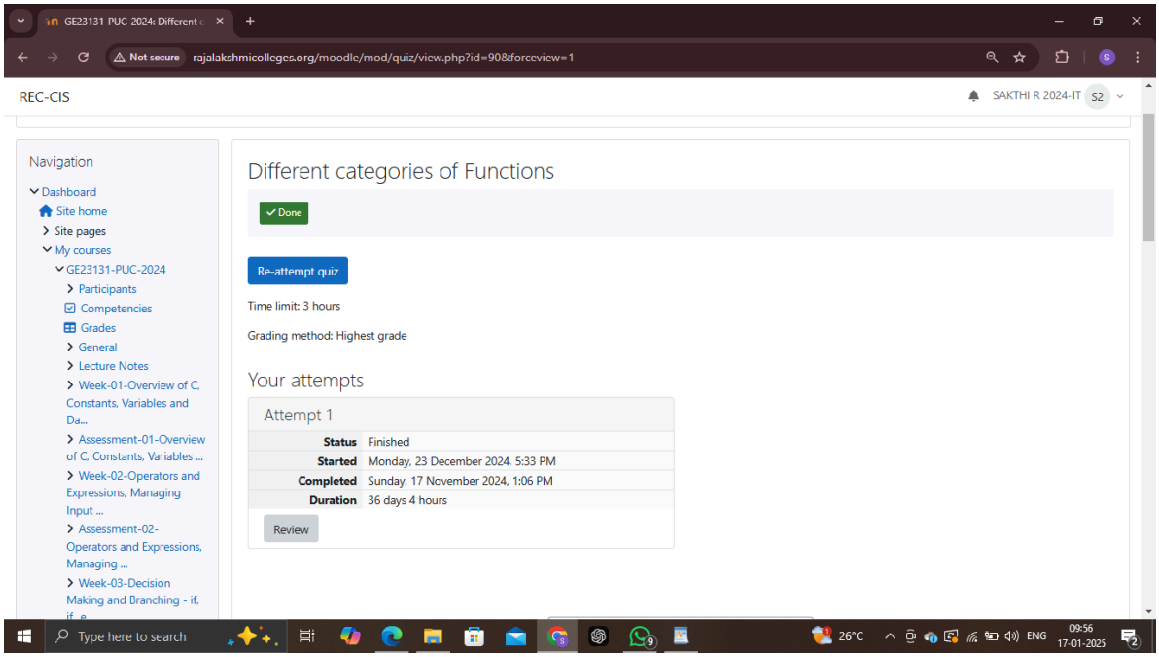
```
#include <stdio.h>

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);
}
```

Answer: (penalty regime: 0 %)

```c
#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);
}
```

	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	✓

Passed all tests! ✓



7)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.

Function without arguments and without return value

Function with arguments and without return value

Function without arguments and with return value

Function with arguments and with return value

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

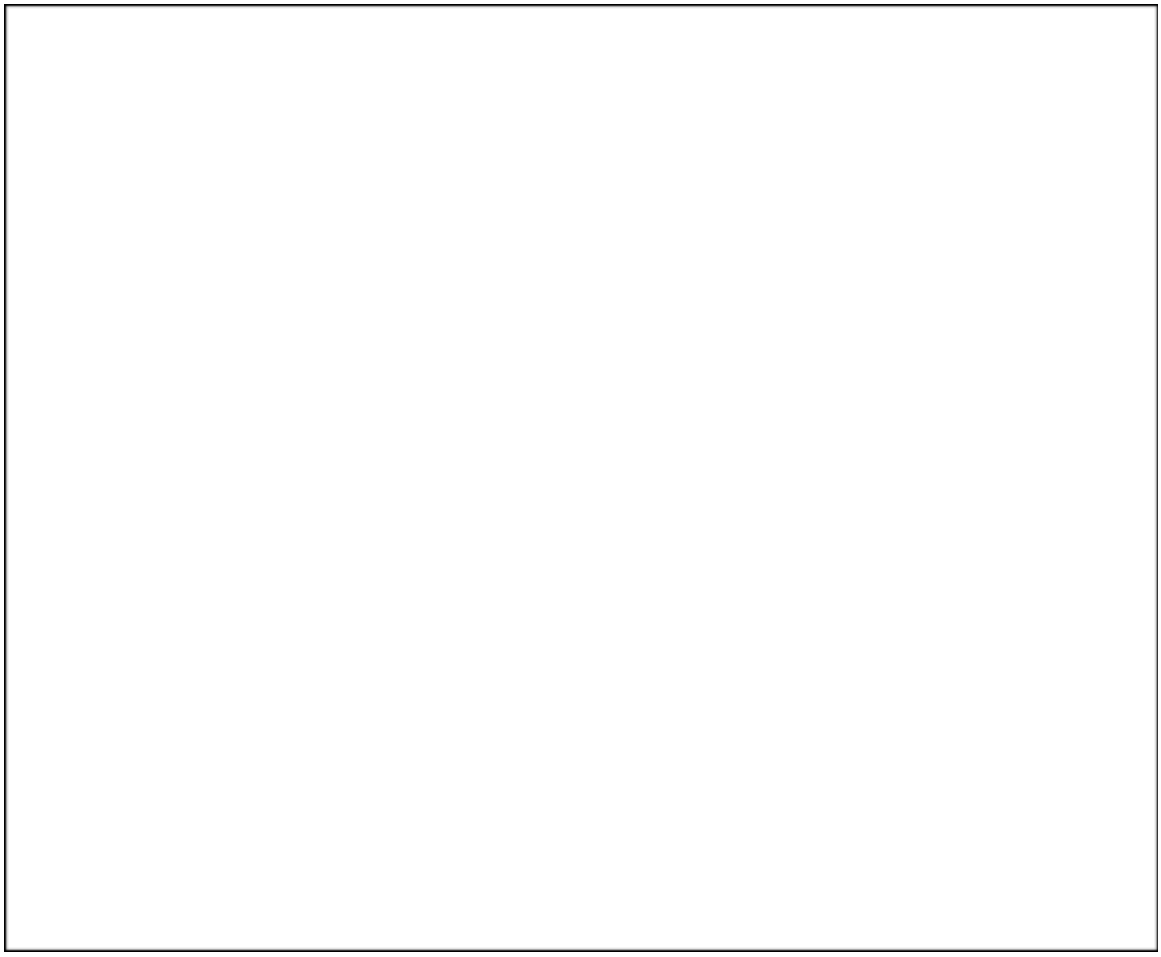
Similarly, when a function does not return a value, the calling function does not receive any data from the called function.

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.



```c
#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");
}
```

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

Passed all tests! ✓

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

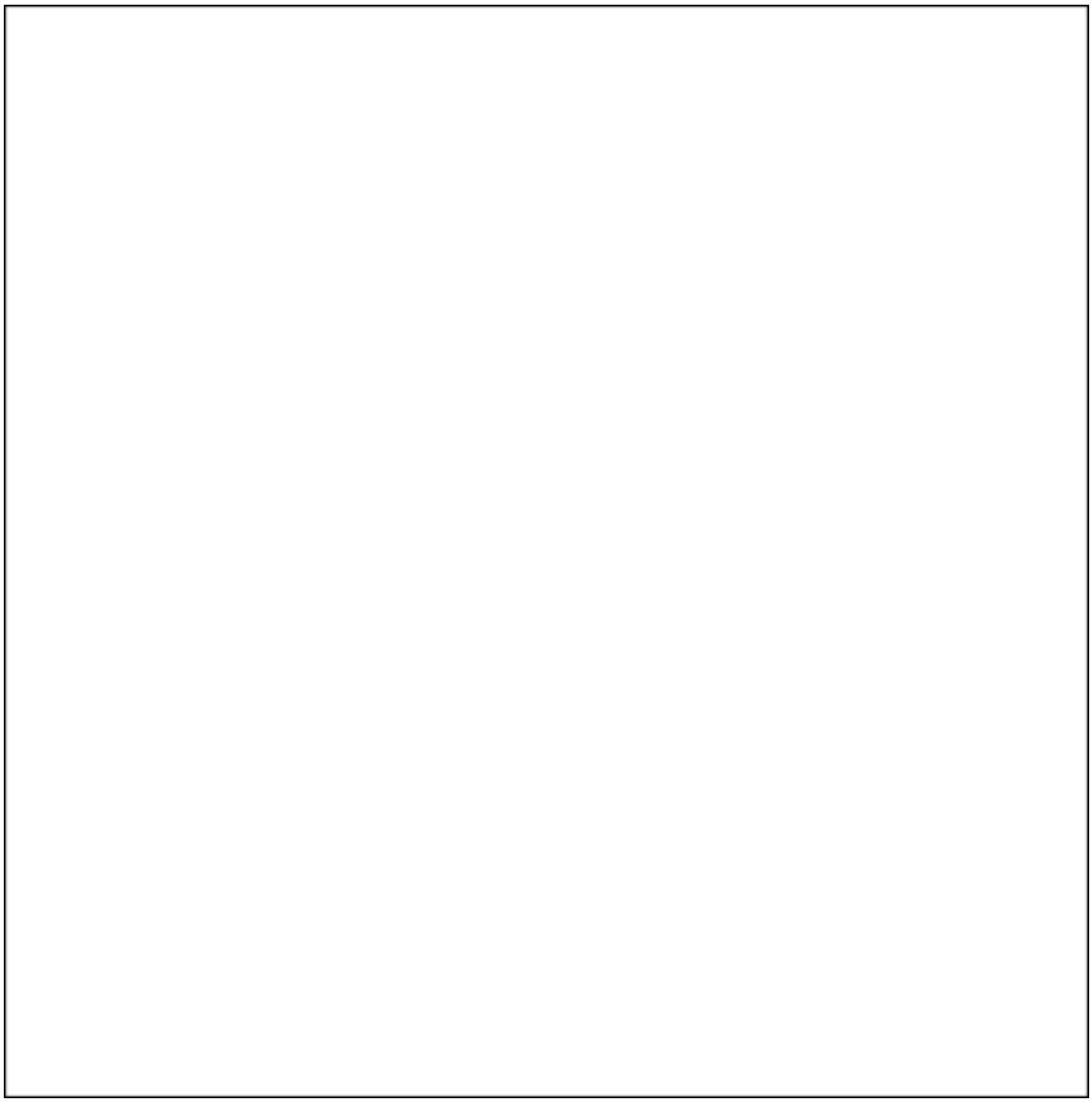
Write the functions print() and hello().

The output is:

...***...

Hello! REC

...***...



```c
#include<stdio.h>

void print();
void hello();
int main()
{
print();
hello();
print();
}

void print()

{
printf("...***...\n");

}



void hello()

 {
printf("Hello! REC\n");
}
```

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

Passed all tests! ✓

9)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);
        }
```

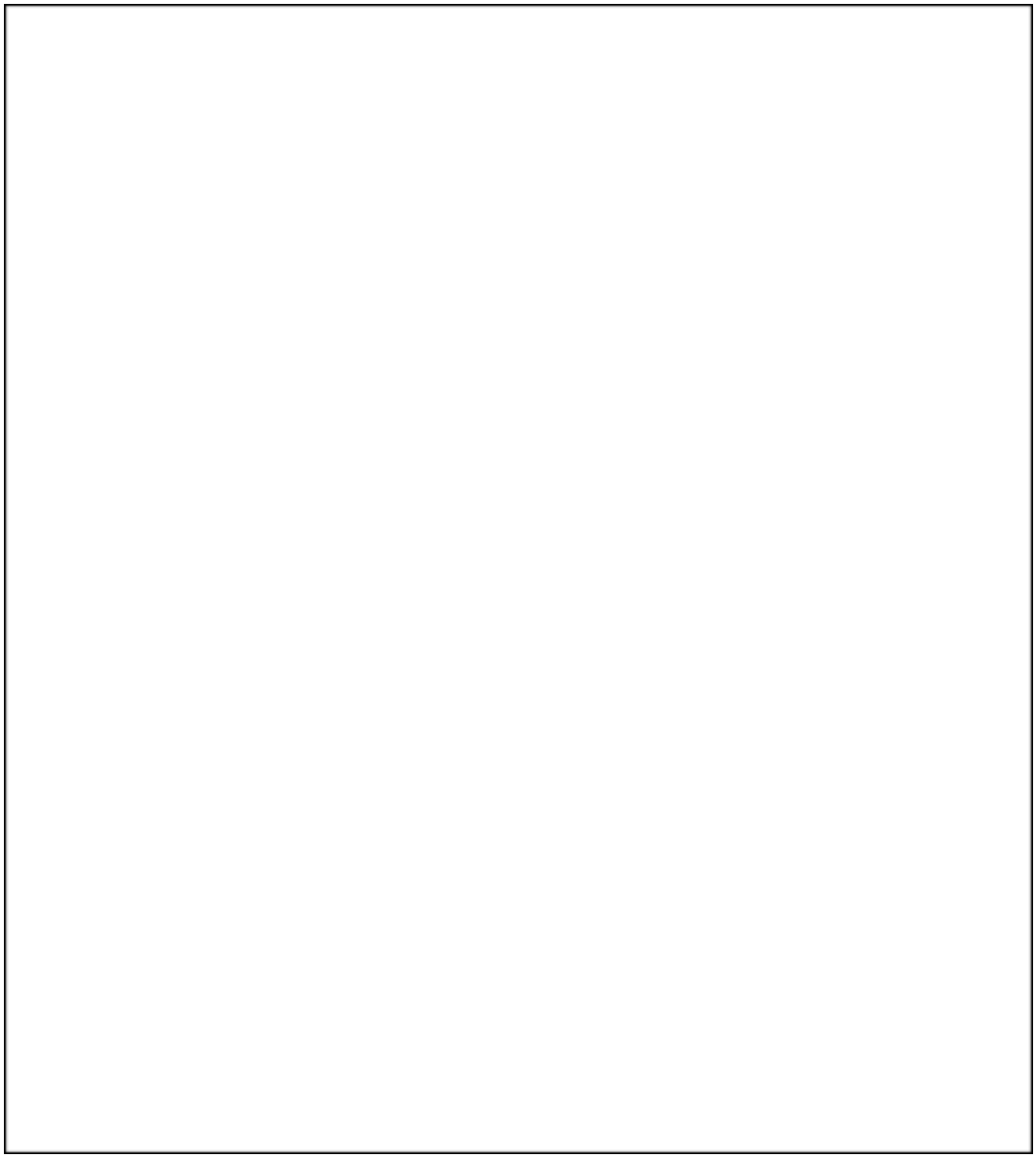
}

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.

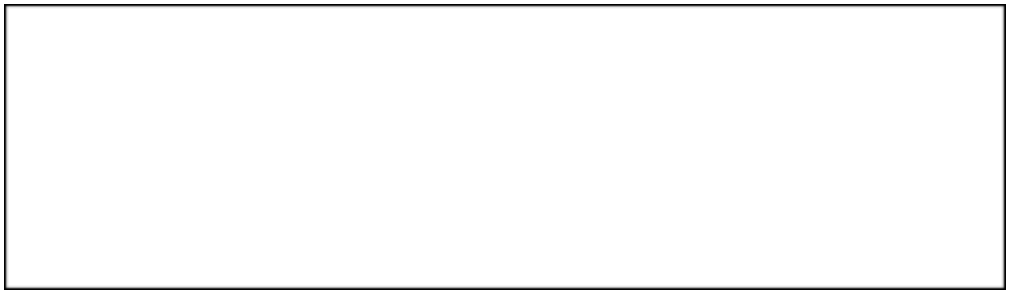
When the function call largest(a, b) is made in the main() function, the values of actual arguments a and b are copied in to the formal parameters x and y.

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.



```c
#include <stdio.h>

 void largest (int, int);

 int main()

 {









int a, b;

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);

}

}
```



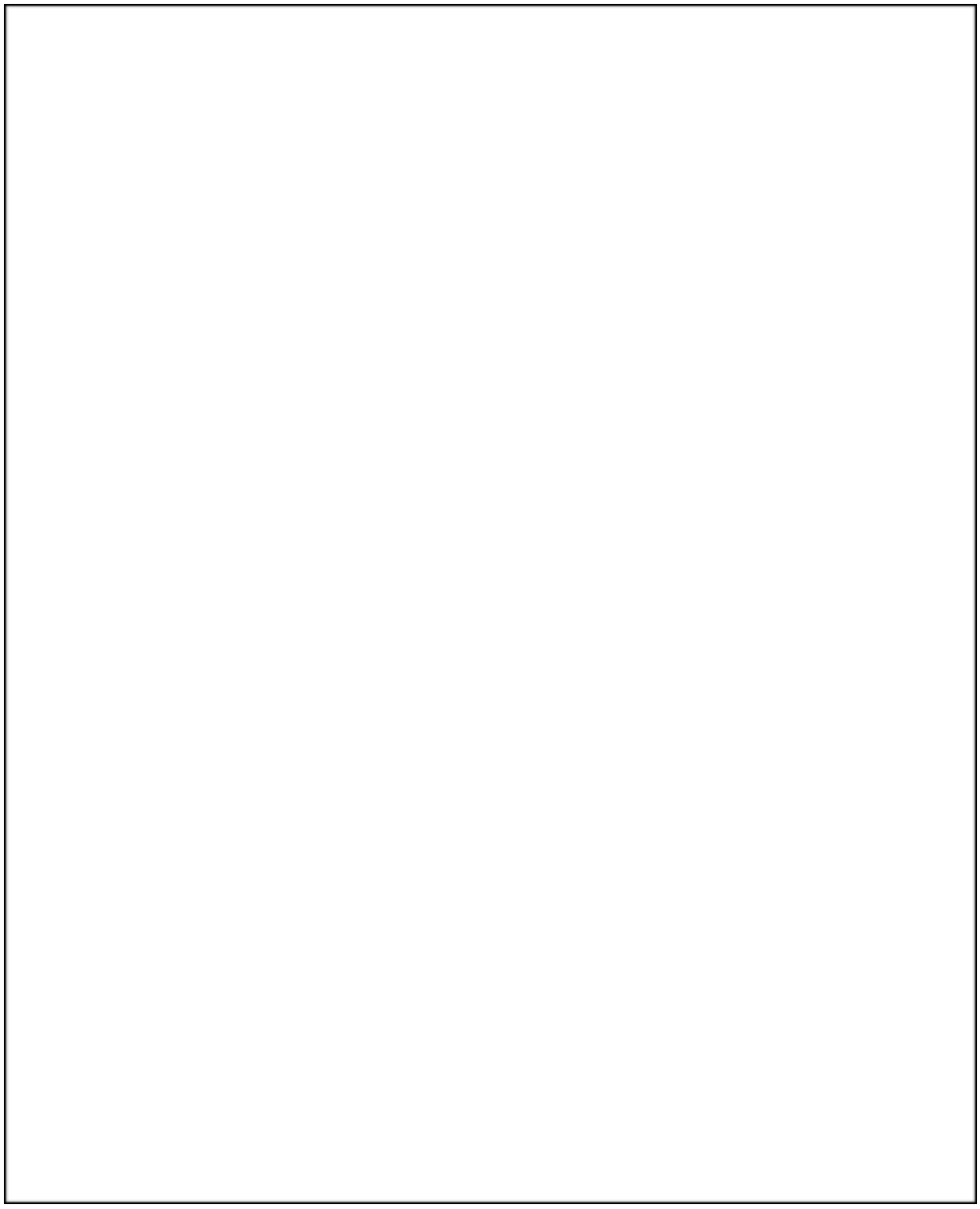
	Input	Expected	Got	
✓	27 18	Largest element = 27	Largest element = 27	✓
✓	13 17	Largest element = 17	Largest element = 17	✓

Passed all tests! ✓

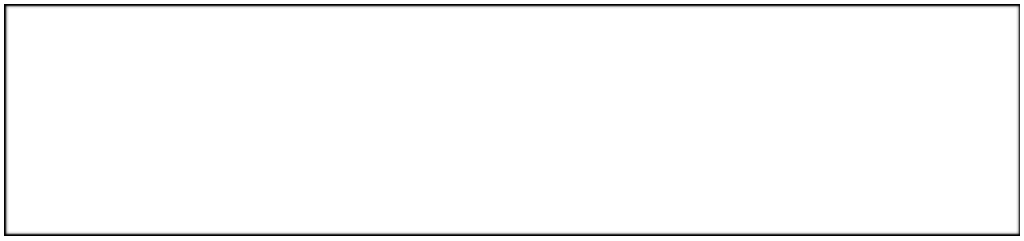
10)Fill the missing code to understand the concept of a function with arguments and without return value.

Note: Take pi value as 3.14

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



```c
#include <stdio.h>

void area_circle(float);


int main()

{


float radius;

scanf("%f", &radius);

area_circle(radius);

return 0;

}




void area_circle(float radius)

 {









float area= 3.14*radius*radius;


printf("Area of circle = %f\n", area);
}
```



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

Passed all tests! ✓

11)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.

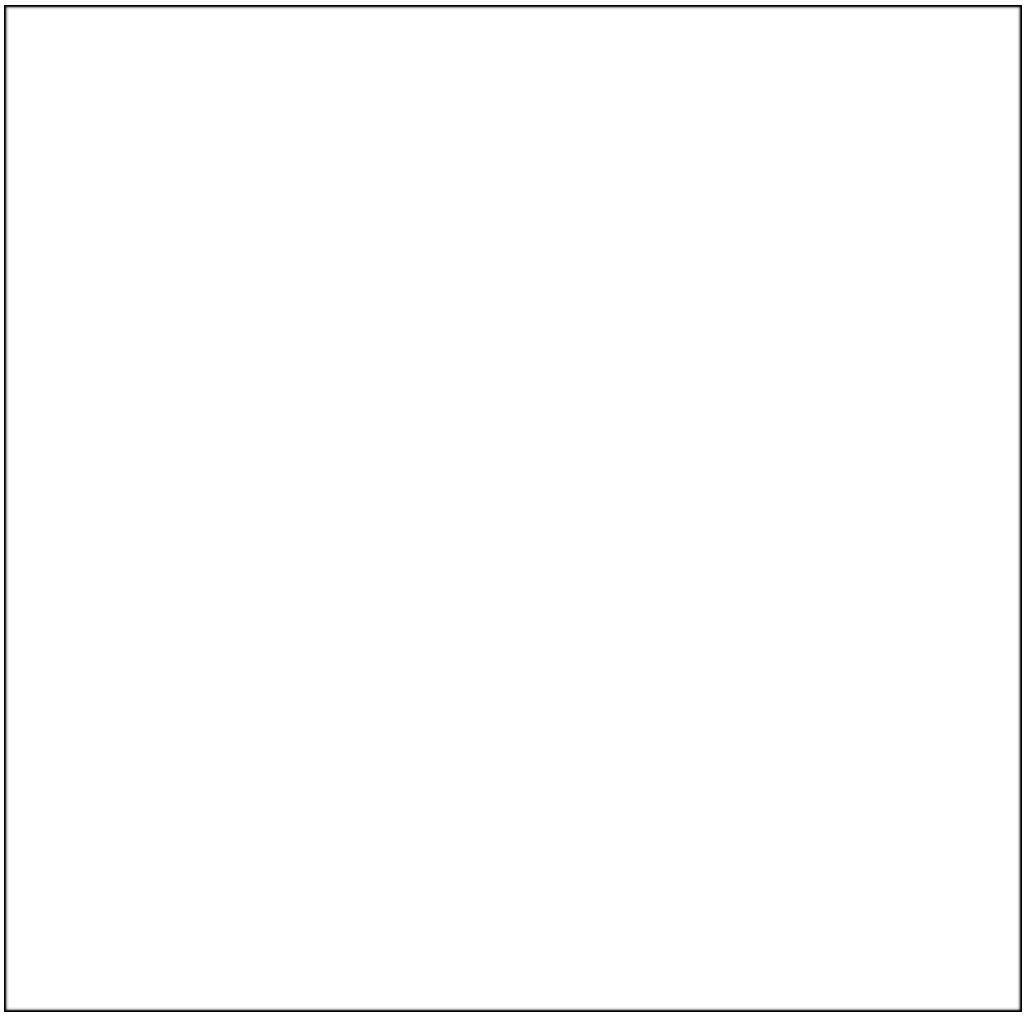
Let us consider an example of a function without arguments and with return value:

```
#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.



```c
#include <stdio.h>

int sum(void);

int main()
{
    printf("Sum of two given values = %d\n", sum());
    return 0;
}

int sum()
{
    int a,b,total;
scanf("%d%d", &a, &b);

total =a + b;



return total;
}
```

	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! ✓

12)

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.

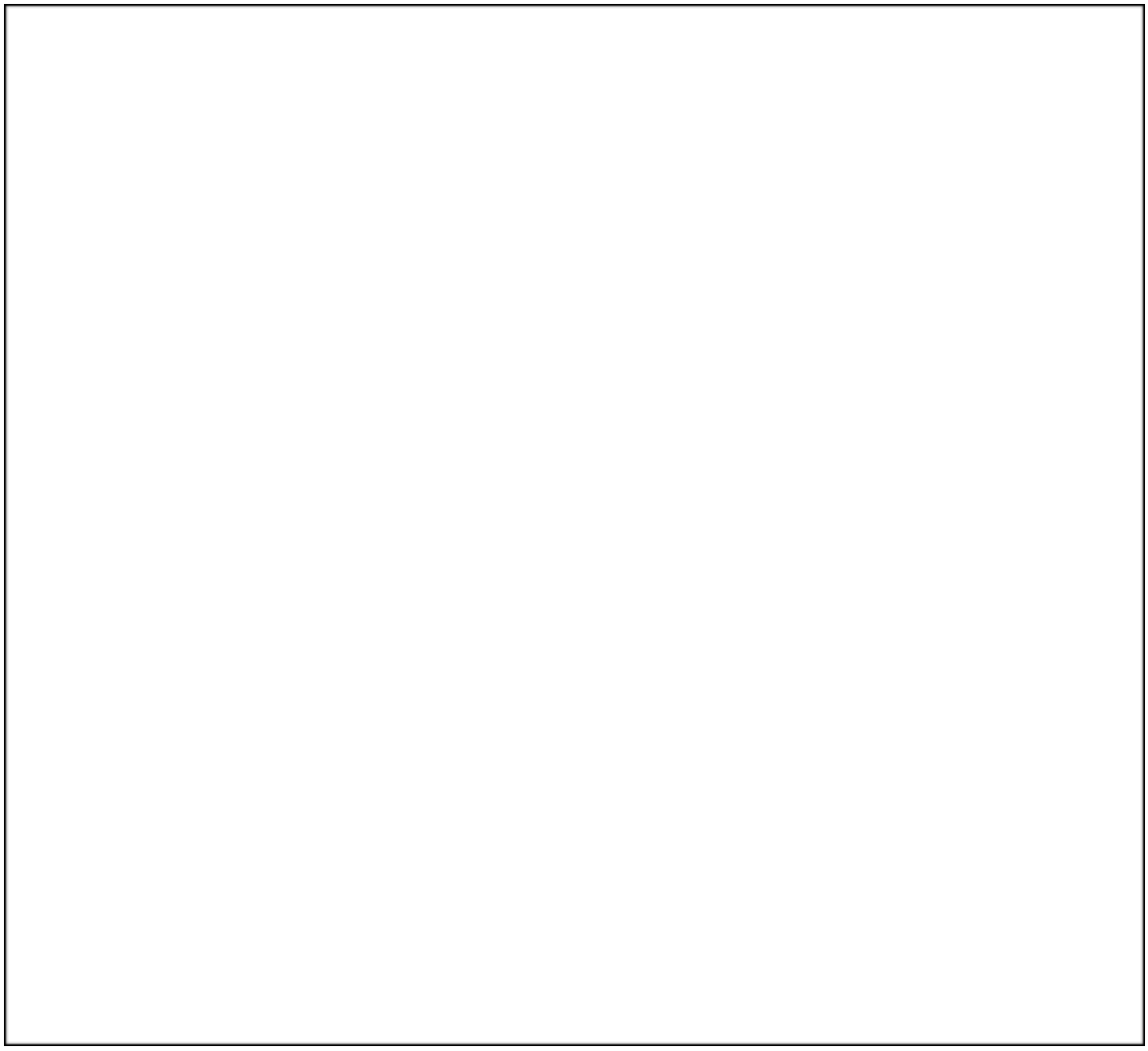
Let us consider an example of a function with arguments and with return value:

#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

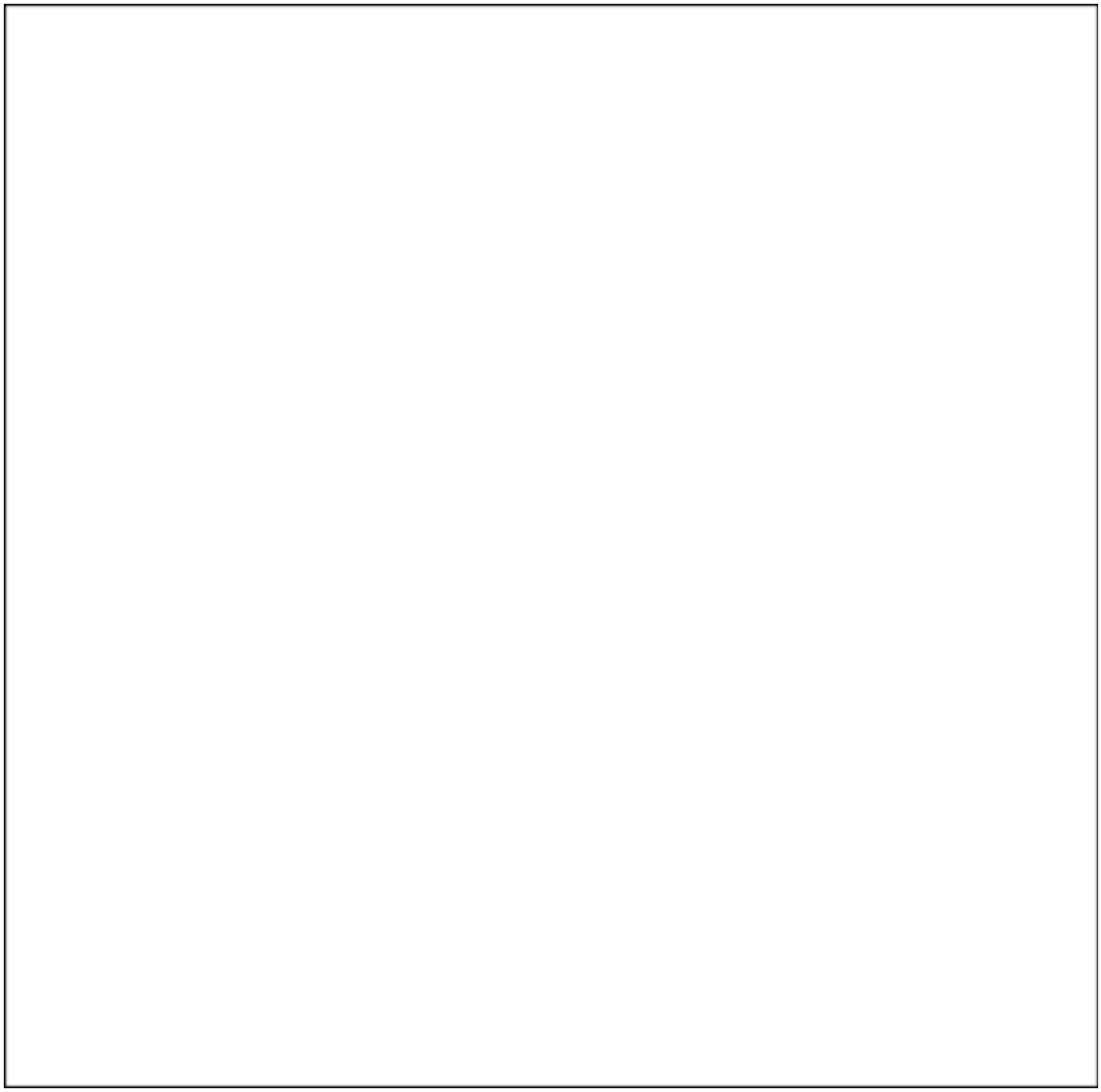


```c
#include <stdio.h>

int largest(int, int, int); int main()

{

int a, b, c;

scanf("%d%d%d", &a, &b, &c);

printf("Largest of the given three numbers = %d ",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;

}



}
```

	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	✓

13)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



```c
#include <stdio.h>

int factorial(int);

int main()

{

int number;

scanf("%d", &number);

printf("Factorial of a given number %d = %d",number,factorial(number));
return 0;

}

int factorial(int n)

{

int i, fact = 1; for (i=1;i<=n;i++)

{

fact*=i;


}

return fact;

// Write the return statement

}
```

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

Passed all tests! ✓

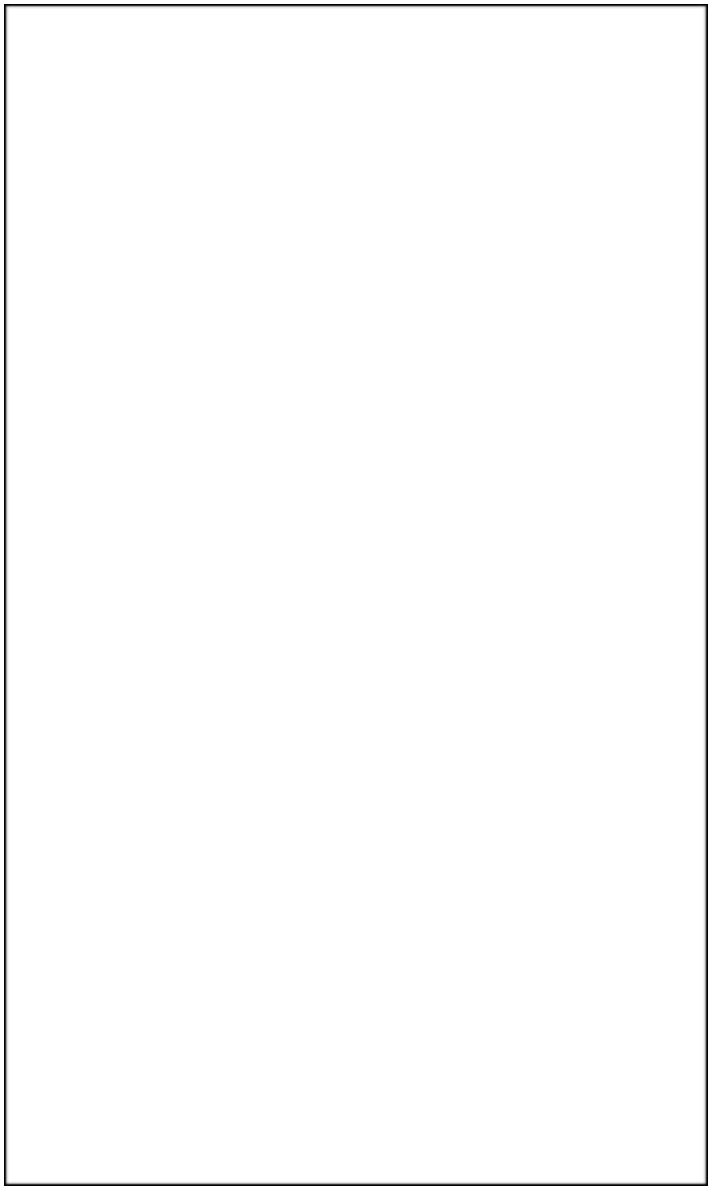
14)Write a C program to demonstrate functions without arguments and with return value.

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

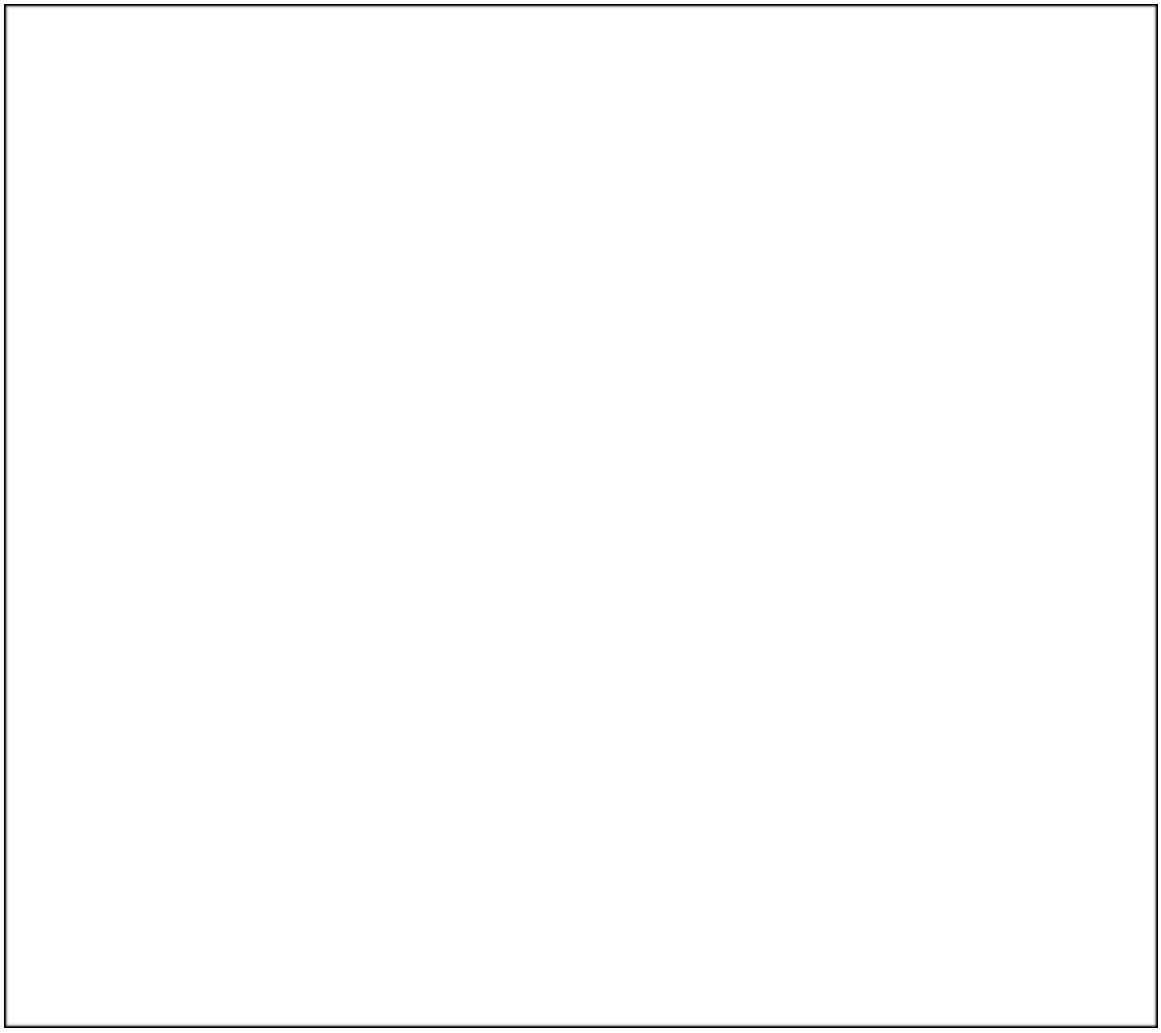
Write the function prime().

Sample Input and Output:

The given number is a prime number



```c
#include <stdio.h>

int prime();

int main()

{

if (prime() == 0)

{

printf("The given number is a prime num

}

else

{

printf("The given number is not a prime

}

return 0;

}

int prime()

{

int a,i;
```



```c
scanf("%d",&a);

if(a<=1)

{

return 1;

}

for(i=2;i<a;i++)

{

if(a%i==0)

{

return 1;

}

}

return 0;

}
```

	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	✓