

1)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



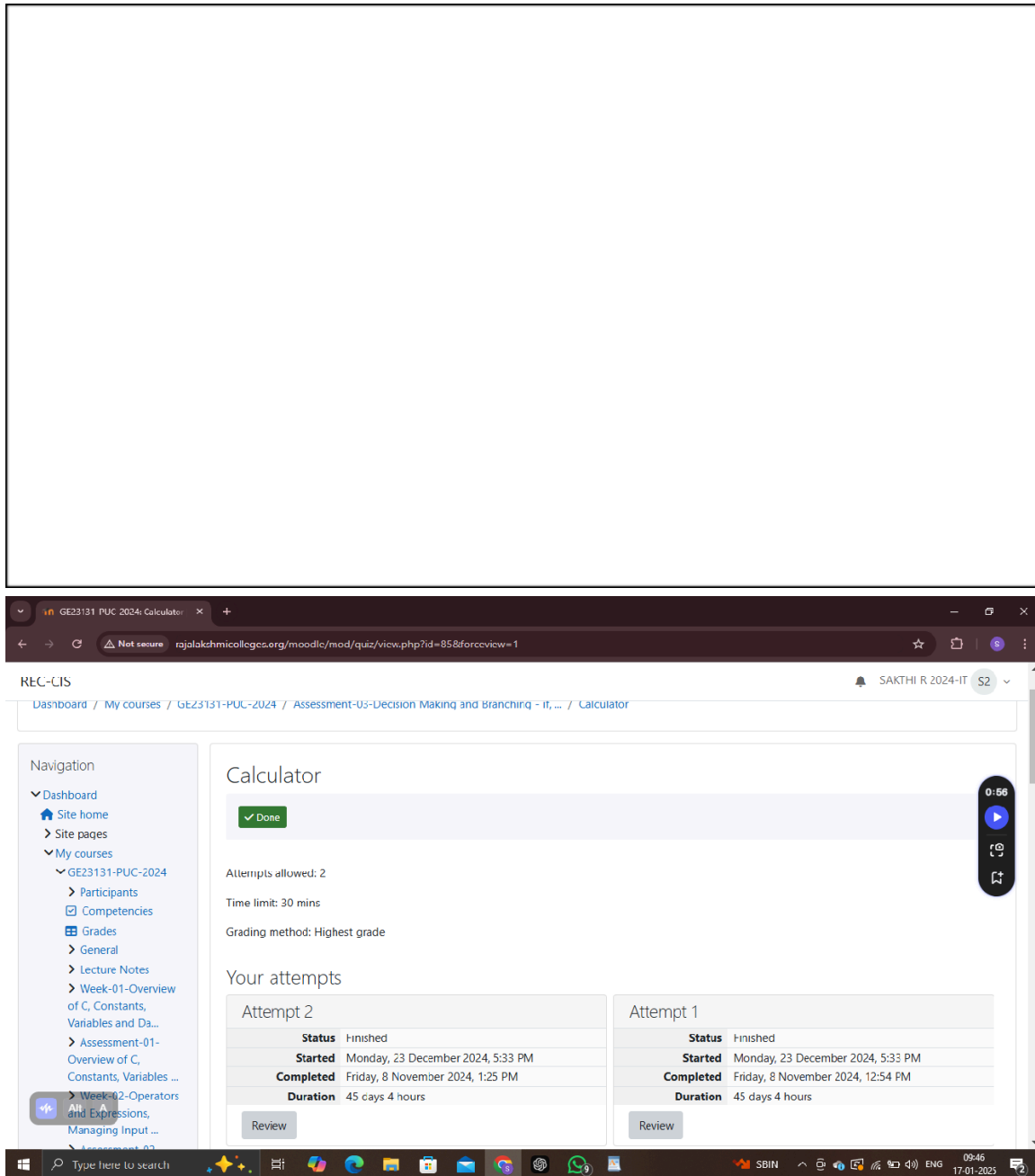
```

1 #include<stdio.h>
2 int main()
3 {
4     int a,b,c,tot;
5     scanf("%d %d %d",&a,&b,&c);
6     tot=a+b+c;
7     if(a>=65&&b>=55 &&c>=50 )
8         printf("The candidate is eligible");
9     else if(tot>=180)
10        printf("The candidate is eligible");
11    else
12        printf("The candidate is not eligible");
13 }

```

	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:  $5 / 2 = 2.500000$



```

1 #include<stdio.h>
2 int main()
3 {
4     int a,b;
5     char c;
6     float r;
7     scanf("%d %d %c",&a,&b,&c);
8     switch(c)
9     {
10         case '+':r=a+b;printf("Result: %d + %d = %f ",a,b,r);break;
11         case '-':r=a-b;printf("Result: %d - %d = %f ",a,b,r);break;
12         case '*':r=a*b;printf("Result: %d * %d = %f ",a,b,r);break;
13         case '/':r=a/b;printf("Result: %d / %d = %f ",a,b,r);break;
14         case '%':r=a%b;printf("Result: %d %% %d = %f ",a,b,r);break;
15         default:r=0.0;printf("Invalid operation.\n");printf("Result: %d %c %d = %f ",a,c,b,r);
16     }
17     return 0;
18 }

```

	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	✓

The screenshot shows a web browser window displaying a Moodle quiz page. The browser's address bar shows the URL: `rajalakshmicolleges.org/moodle/mod/quiz/view.php?id=86&forceview=1`. The page title is "Finding the Second Largest Element". A green "Done" button is visible at the top. Below it is a blue "Re-attempt quiz" button. The page indicates "Attempts allowed: 2", "Time limit: 30 mins", and "Grading method: Highest grade". A section titled "Your attempts" shows a table for "Attempt 1":

	Status	Finished
Started	Monday, 23 December 2024 5:33 PM	
Completed	Tuesday, 12 November 2024 8:29 AM	
Duration	41 days 9 hours	

Below the table is a "Review" button. On the left side of the page, there is a navigation menu with links to "Dashboard", "Site home", "Site pages", and "My courses". Under "My courses", there is a link to "GE23131-PUC-2024" with sub-links for "Participants", "Competencies", "Grades", "General", "Lecture Notes", "Week-01-Overview of C, Constants, Variables and Da...", "Assessment-01-Overview of C, Constants, Variables...", "Week-02-Operators and Expressions, Managing Input...", "Assessment-02-Operators and Expressions, Managing...", "Week-03-Decision Making and Branching - if", and "Making and Branching - if".

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

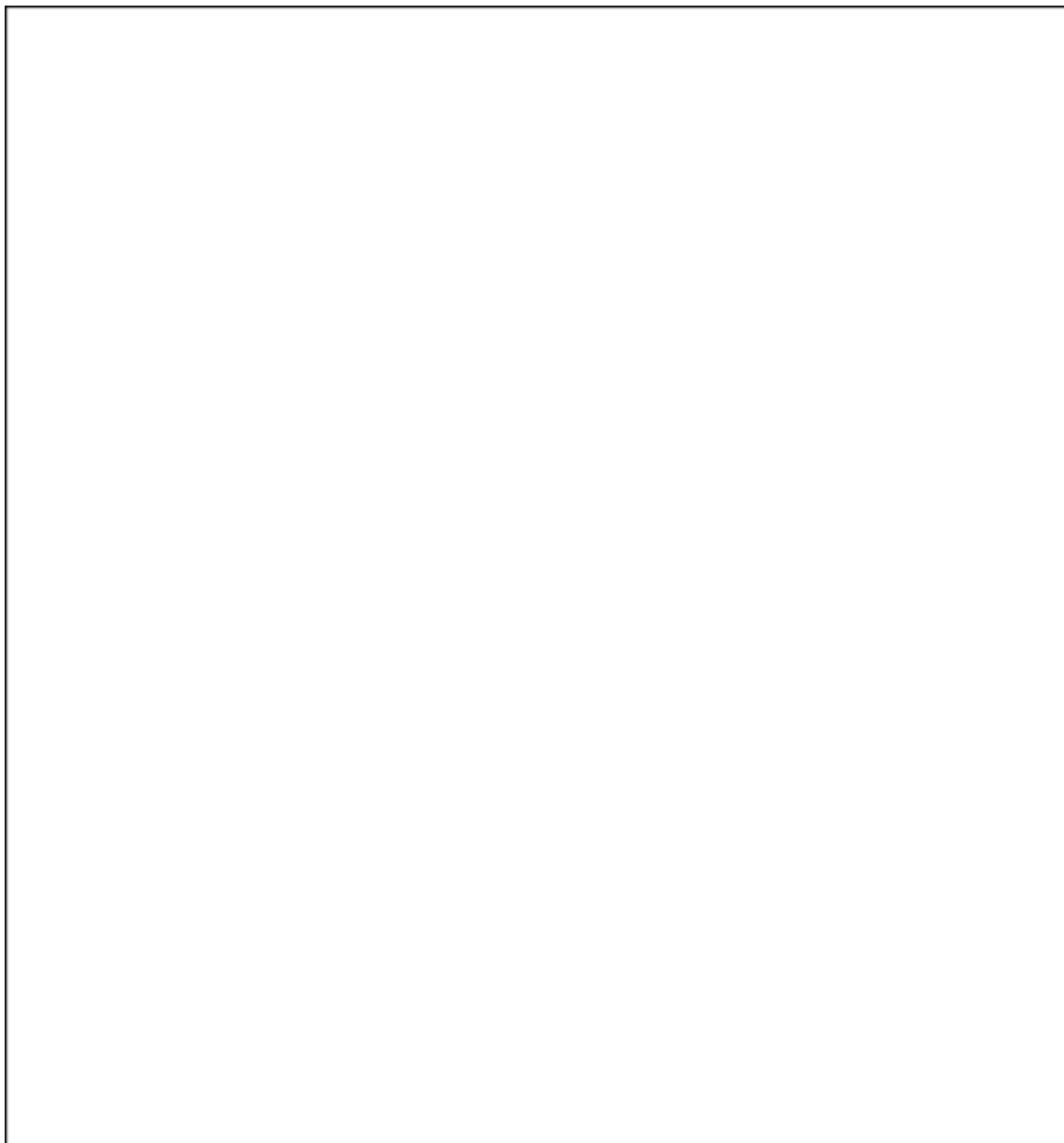
### Test Case 2

Input

-840 -335 -1

Output

-840



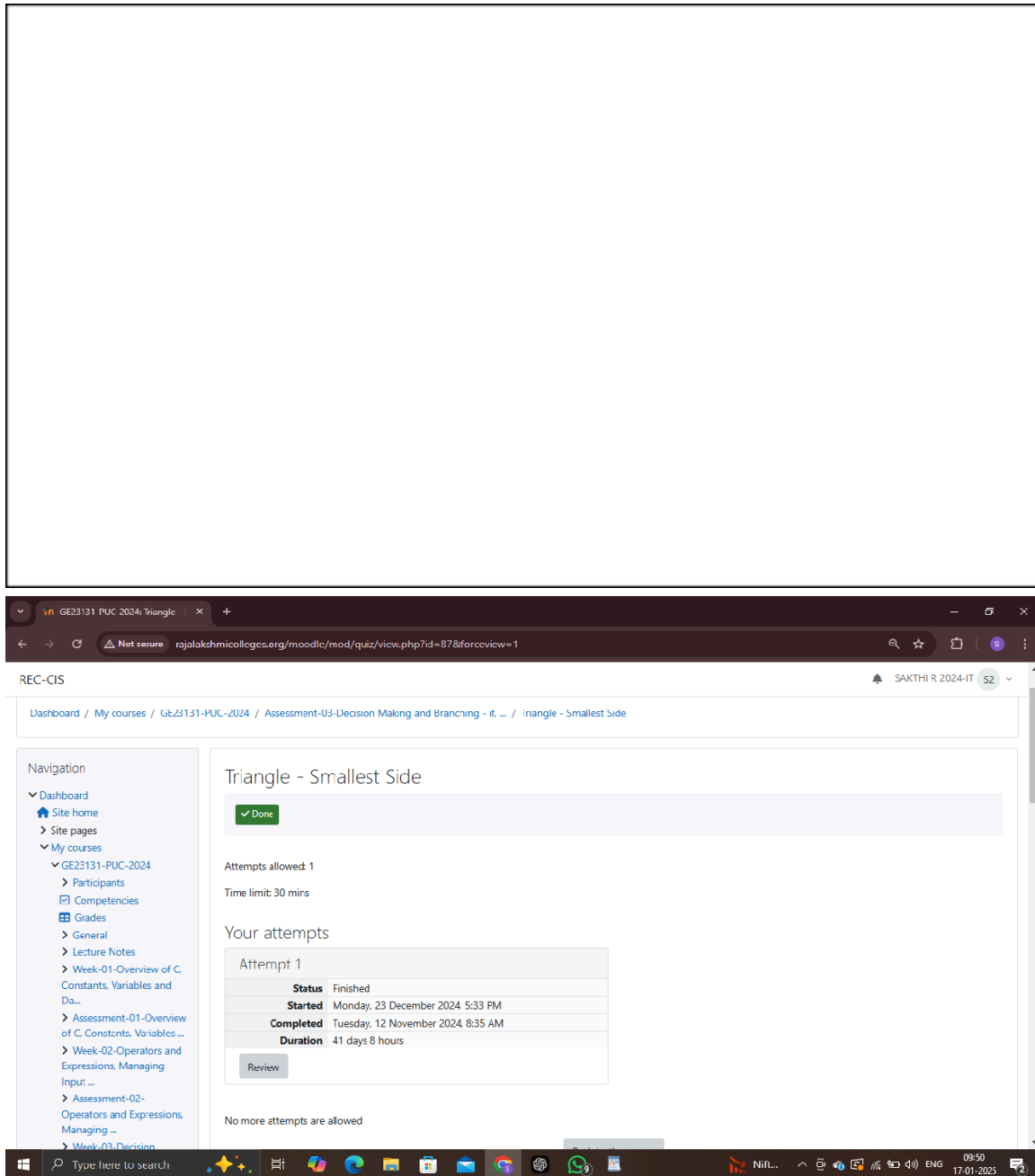
```

1  #include<stdio.h>
2  #include<limits.h>
3  int main()
4  {
5      int num;
6      int lag=INT_MIN;
7      int s1=INT_MIN;
8
9
10     while(1){
11         scanf("%d",&num);
12         if(num==-1)
13             break;
14         if(num>lag)
15         {s1=lag;
16         lag=num;
17         }
18         else if(num<lag&&num>s1)
19             s1=num;
20         }
21         if(s1==INT_MIN)
22             printf("no");
23         else
24             printf("%d",s1);
25         return 0;
26
27     }

```

	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:

$1 \leq X \leq 999999$

$1 \leq Y \leq 999999$

$1 \leq Z \leq 999999$

Example Input/Output 1:

Input:

40

30

50

Output:

30

Example Input/Output 2:

Input:

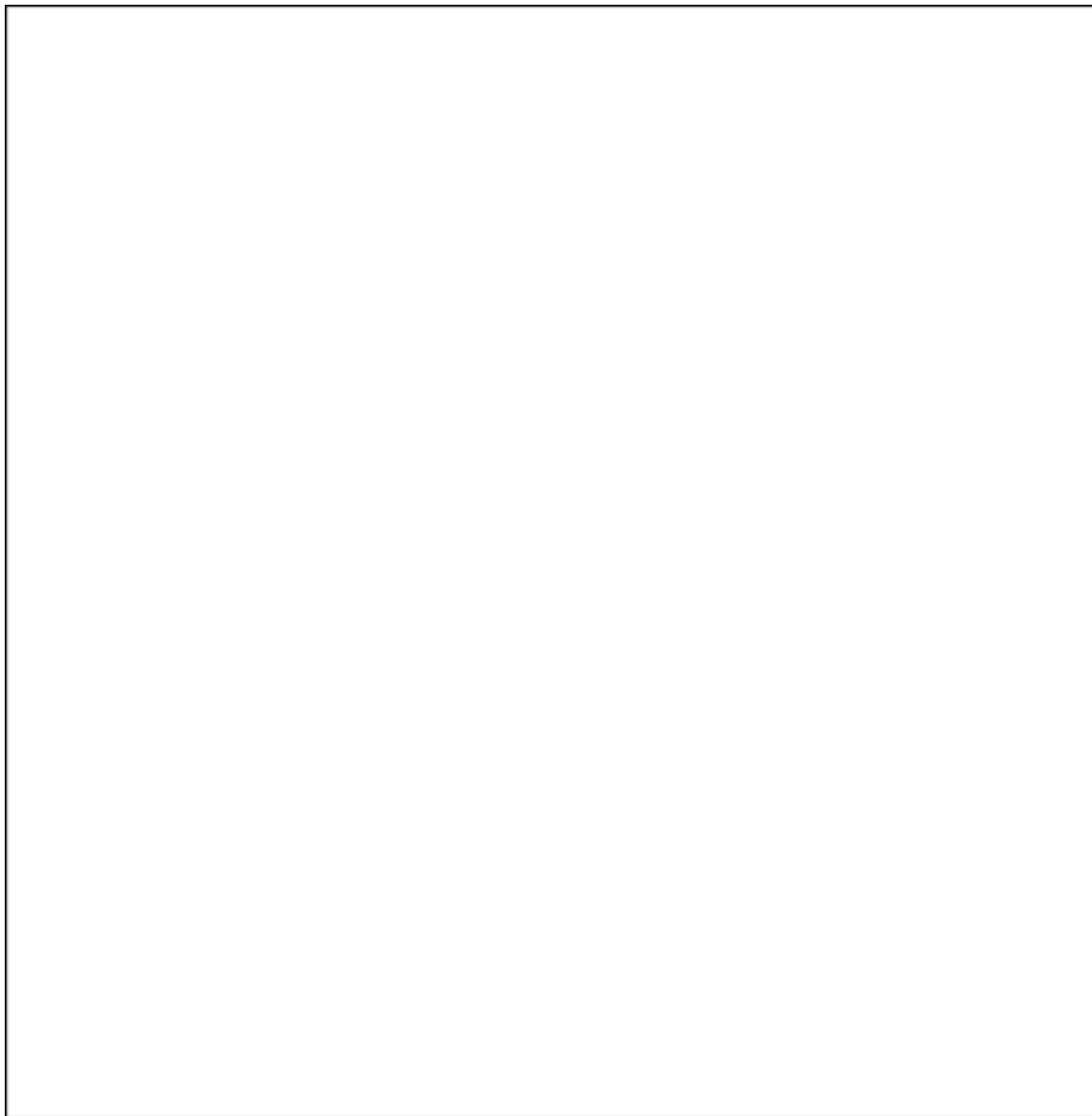
15

15

15

Output:

15



```

1 #include<stdio.h>
2 int main()
3 {
4     int a,b,c;
5     scanf("%d %d %d",&a,&b,&c);
6     int small=a;
7     if(b<small)
8         printf("%d",b);
9     else if(c<small)
10        printf("%d",c);
11     else
12        printf("%d",a);
13 }

```

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

Navigation

- Dashboard
- Site home
- Site pages
- My courses
  - GE23131-PUC-2024
    - Participants
    - Competencies
    - Grades
    - General
    - Lecture Notes
    - Week 01-Overview of C, Constants, Variables and Da...
    - Assessment-01-Overview of C, Constants, Variables ...
    - Week-02-Operators and Expressions, Managing Input ...
    - Assessment 02 Operators and Expressions, Managing ...
    - Week-03-Decision Making and Branching - if, if...e...

## Formal and Actual Arguments

✓ Done

Re-attempt quiz

Time limit: 1 hour

Grading method: Highest grade

### Your attempts

Attempt 1	
<b>Status</b>	Finished
<b>Started</b>	Monday, 23 December 2024 5:33 PM
<b>Completed</b>	Tuesday, 12 November 2024 8:45 AM
<b>Duration</b>	41 days 8 hours

Review

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 within 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;
```

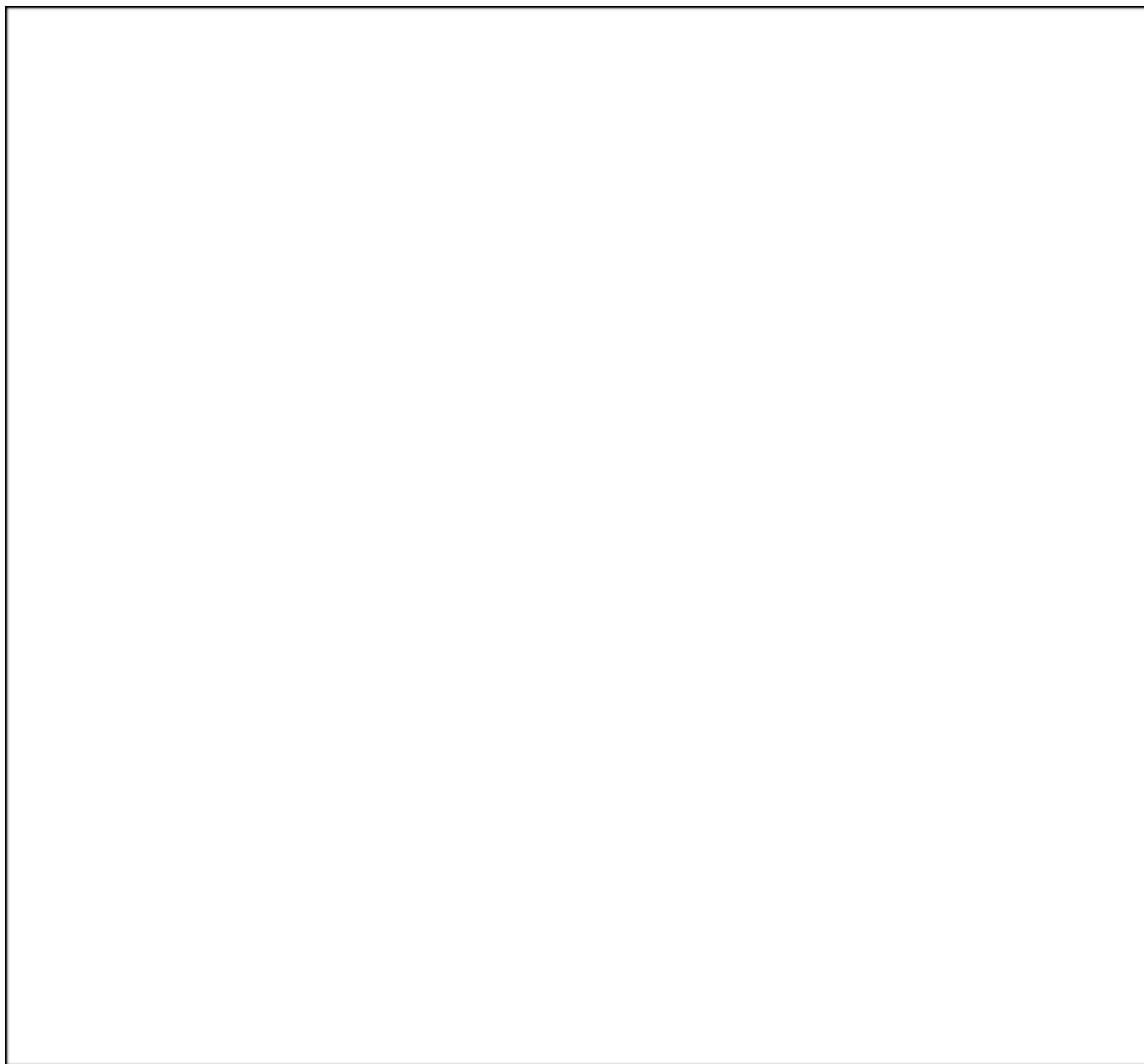
```
    for (i = 1; i <= value; i++)
```

```
    {
```

```
        total = total + i;
```



```
}  
    return(total);  
}
```



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

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

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

Passed all tests! ✓

The screenshot shows a web browser displaying a Moodle quiz page. The browser's address bar shows the URL: `rajalakshmicolleges.org/moodle/mod/quiz/view.php?id=89&forceview=1`. The page title is 'REC-CIS' and the course is 'SAKTHI R 2024-IT S2'. The navigation menu on the left includes 'Dashboard', 'Site home', 'Site pages', 'My courses', and 'GE23131-PUC-2024'. The main content area is titled 'Local and Global Variables' and shows a 'Done' button, a 'Re-attempt quiz' button, a 'Time limit: 1 hour', and a 'Grading method: Highest grade'. Below this, the 'Your attempts' section shows 'Attempt 1' with a table of details:

Attempt 1	
Status	Finished
Started	Monday, 23 December 2024 5:33 PM
Completed	Tuesday, 12 November 2024 9:13 AM
Duration	41 days 8 hours

A 'Review' button is located below the attempt table. The Windows taskbar at the bottom shows the system clock as 09:54 on 17-01-2025.

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;
```

```
}
```

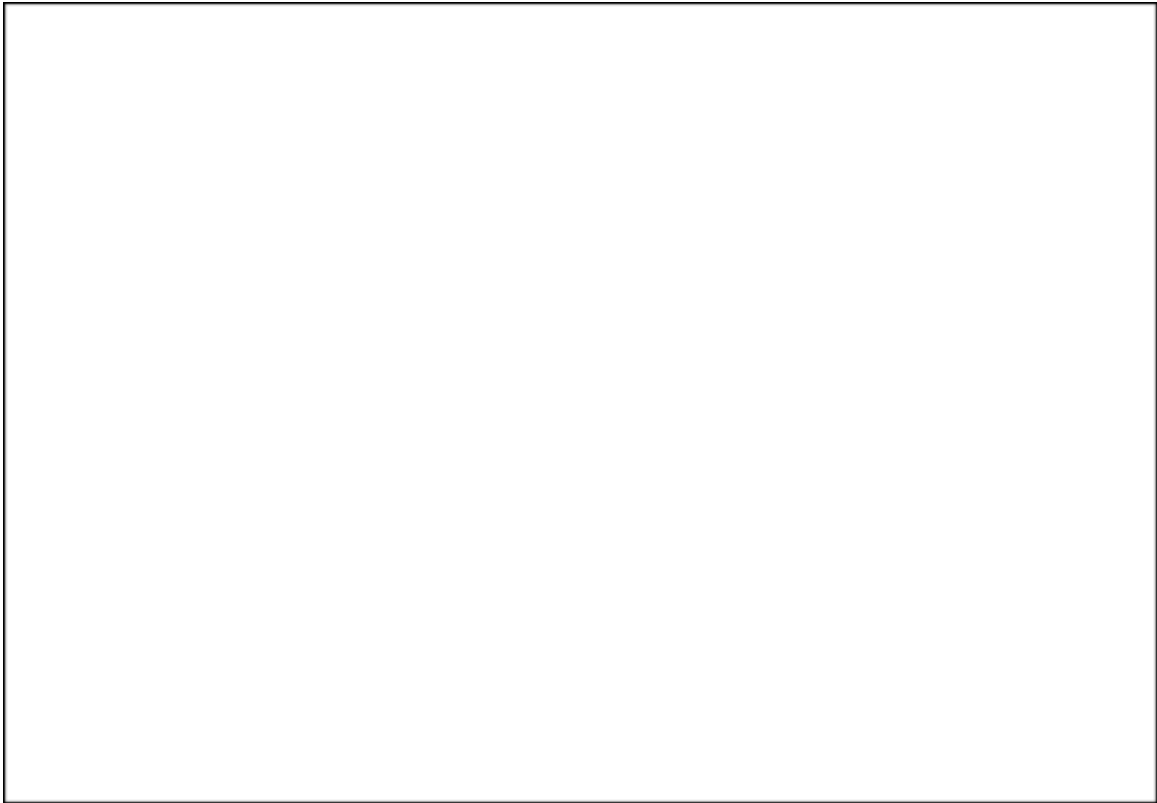
```
void test()
```

```
{
```

```
    int a = 5, b = 55;
```

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

```
}
```



Answer: (penalty regime: 0 %)

```
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	✓

Passed all tests! ✓

Navigation

- Dashboard
- Site home
- Site pages
- My courses
  - GE23131-PUC-2024
    - Participants
    - Competencies
    - Grades
      - General
      - Lecture Notes
      - Week-01-Overview of C, Constants, Variables and Da...
      - Assessment-01-Overview of C, Constants, Variables...
      - Week-02-Operators and Expressions, Managing Input ...
      - Assessment-02-Operators and Expressions, Managing ...
      - Week-03-Decision Making and Branching - if, if e

### Different categories of Functions

✓ Done

Re-attempt quiz?

Time limit: 3 hours

Grading method: Highest grade

#### Your attempts

Attempt 1	
<b>Status</b>	Finished
<b>Started</b>	Monday, 23 December 2024 5:33 PM
<b>Completed</b>	Sunday, 17 November 2024, 1:06 PM
<b>Duration</b>	36 days 4 hours

Review

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;
```

```
}
```

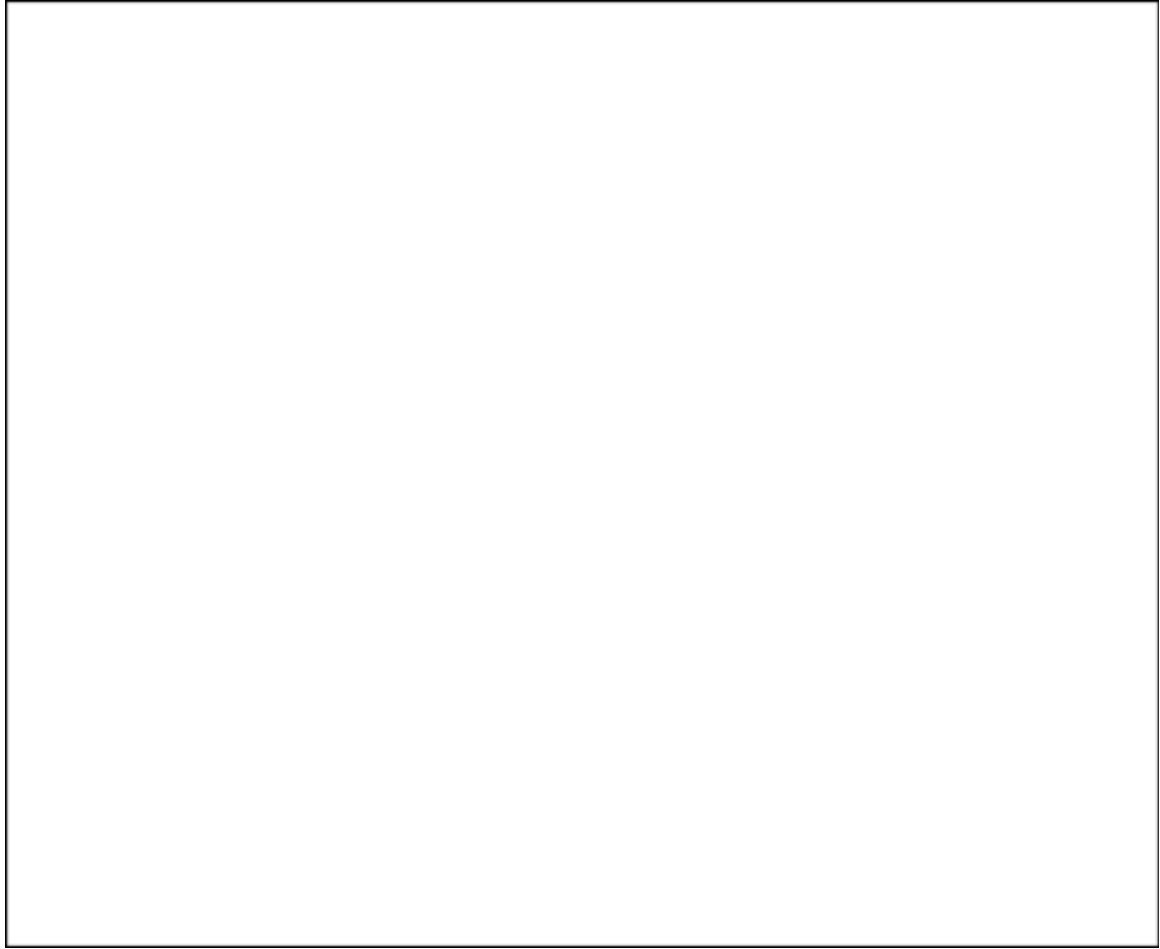
```
void india_capital()
```

```
{
```

```
    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.



```
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! ✓

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

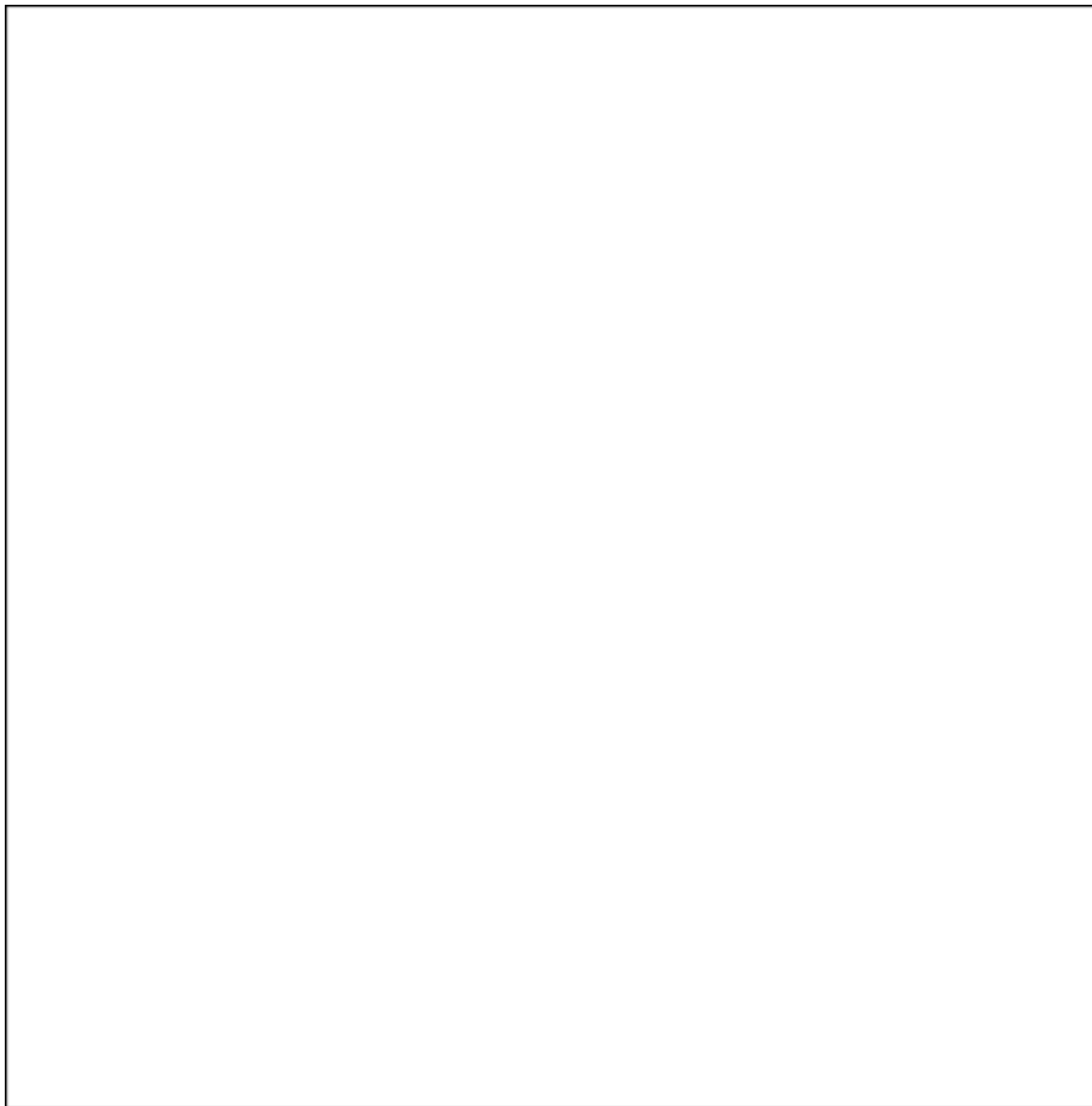
Write the functions print() and hello().

The output is:

... \*\*\* ...

Hello! REC

... \*\*\* ...



```

1  #include<stdio.h>
2
3  void print();
4  void hello();
5  int main()
6  {
7  print();
8  hello();
9  print();
10 }
11
12 void print()
13
14 {
15 printf("...***...\n");
16
17 }
18
19
20
21 void hello()
22
23 {
24 printf("Hello! REC\n");
25 }

```

	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);
    }
}
```

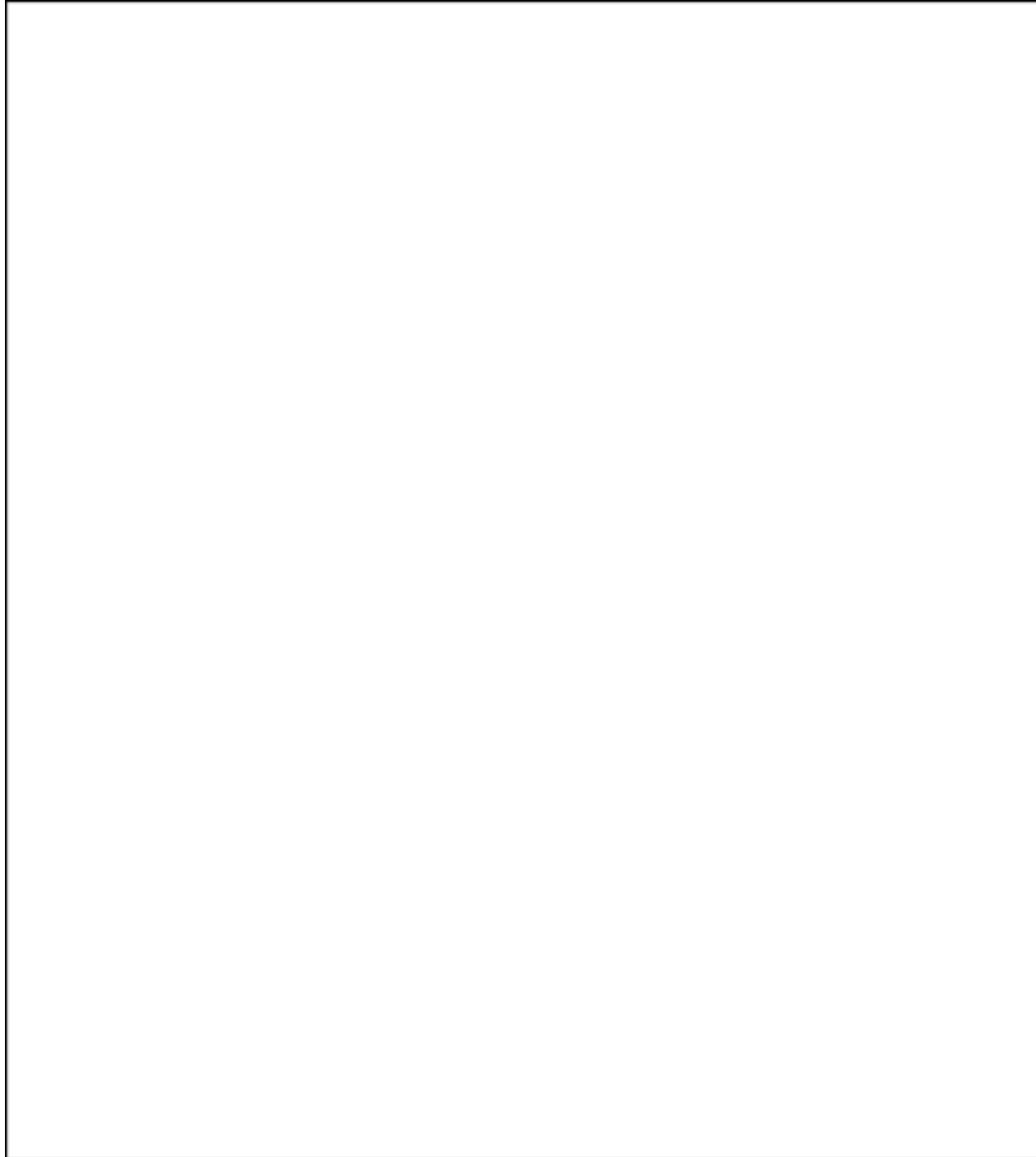
```
}
```

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.





```
1  #include <stdio.h>
2
3  void largest (int, int);
4
5  int main()
6  {
7
8
9
10
11
12
13
14
15
16
17  int a, b;
18  scanf("%d%d", &a, &b);
19
20  largest (a, b);
21  return 0;
22  }
23
24
25
26
27
28
29  void largest(int x, int y)
30  {
31
32
33
34  if (x > y)
35  {
36  printf("Largest element = %d\n",x);
37  }
38  else
39  { printf("Largest element = %d\n",y);
40
41  }
42
43  }
```

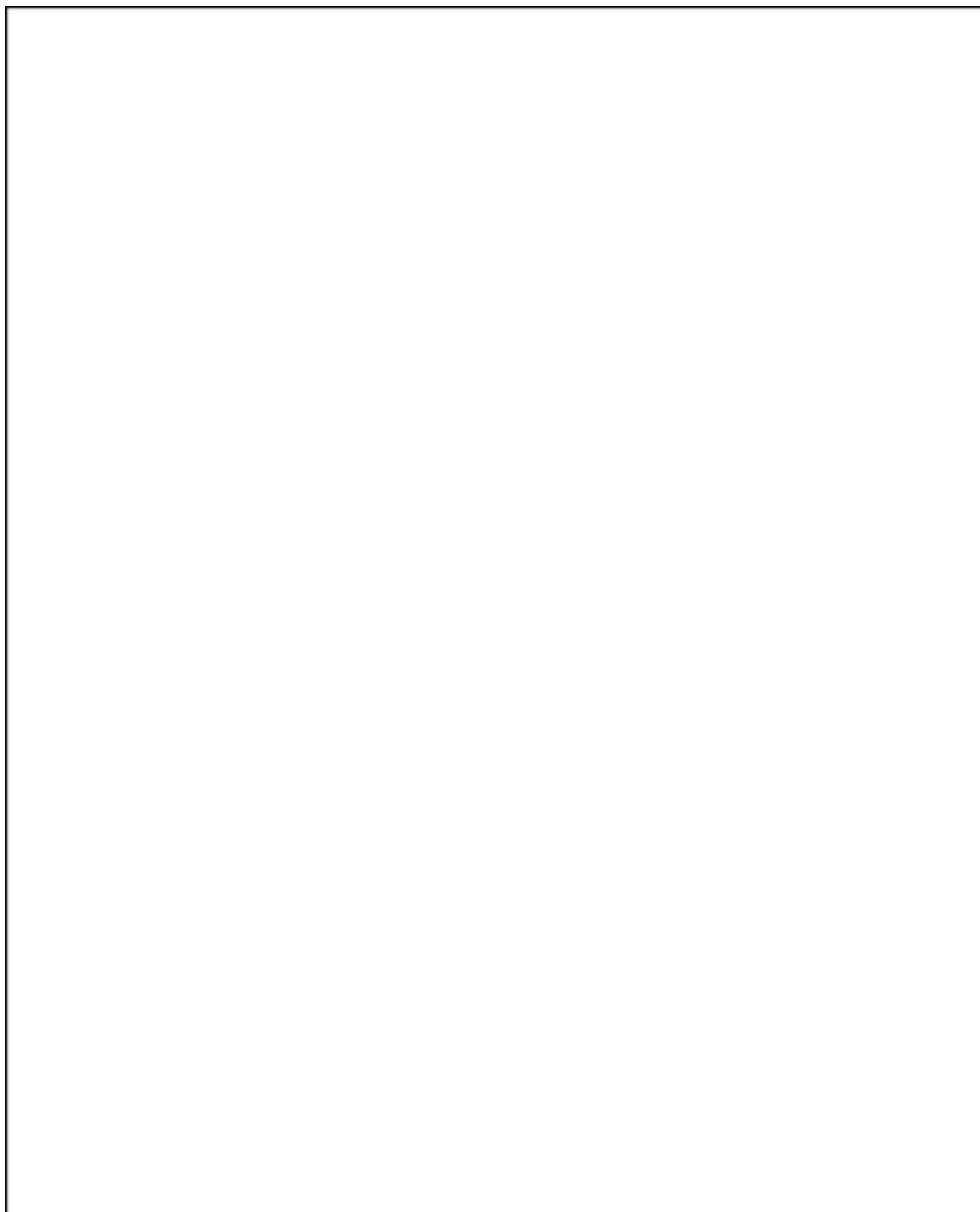
	Input	Expected	Got	
✓	27 18	Largest element = 27	Largest element = 27	✓
✓	13 17	Largest element = 17	Largest element = 17	✓

Passed all tests! ✓

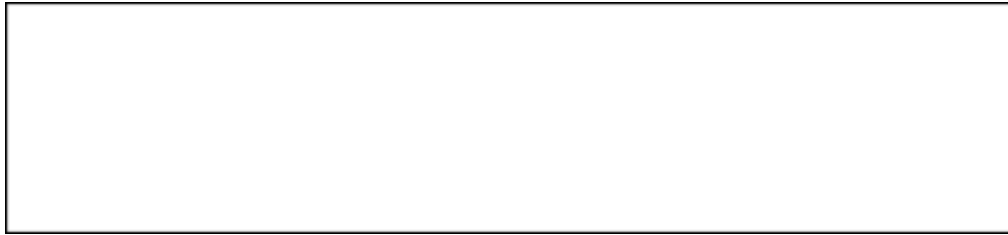
10) Fill the missing code to understand the concept of a function with arguments and without return value.

Note: Take pi value as 3.14

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



```
3  #include <stdio.h>
4
5  void area_circle(float);
6
7
8
9  int main()
10
11 {
12
13
14     float radius;
15
16     scanf("%f", &radius);
17
18     area_circle(radius);
19
20     return 0;
21
22 }
23
24
25
26
27
28 void area_circle(float radius)
29
30 {
31
32
33
34
35
36
37
38
39
40     float area= 3.14*radius*radius;
41
42
43     printf("Area of circle = %f\n", area);
44 }
```



	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 returns 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);

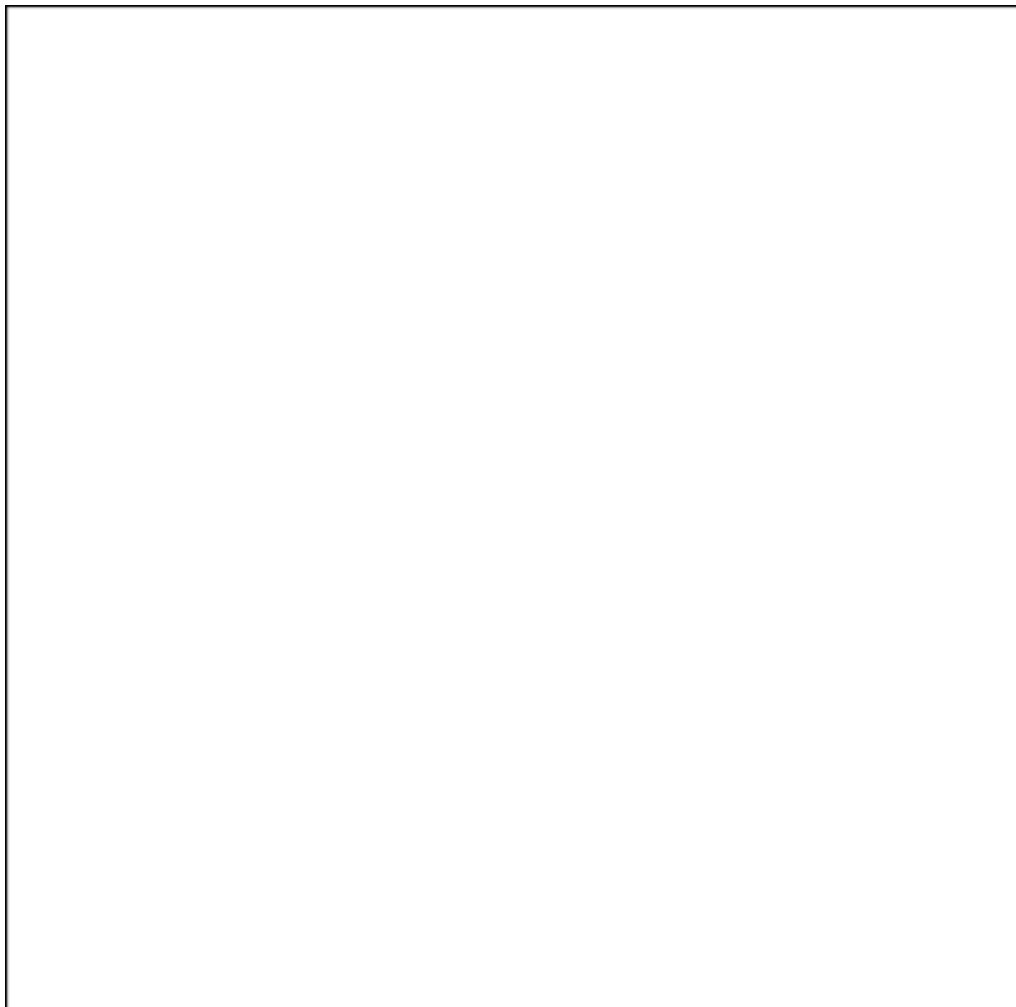
int main()
{
    printf("\nSum of two given values = %d\n", sum());
    return 0;
}

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.



```

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
16     total =a + b;
17
18
19
20     return total;
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! ✓

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));

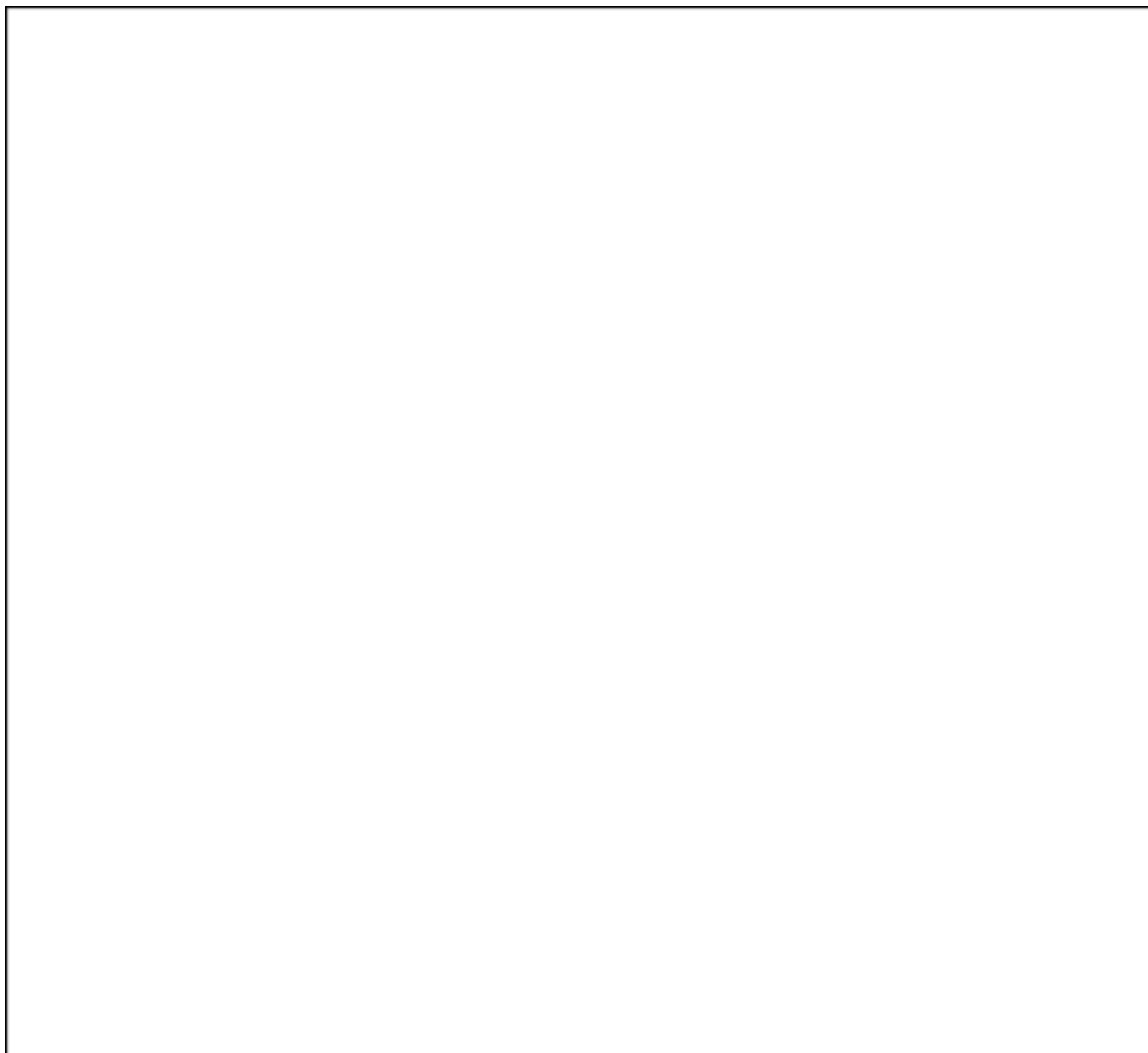
    return 0;
}

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



```

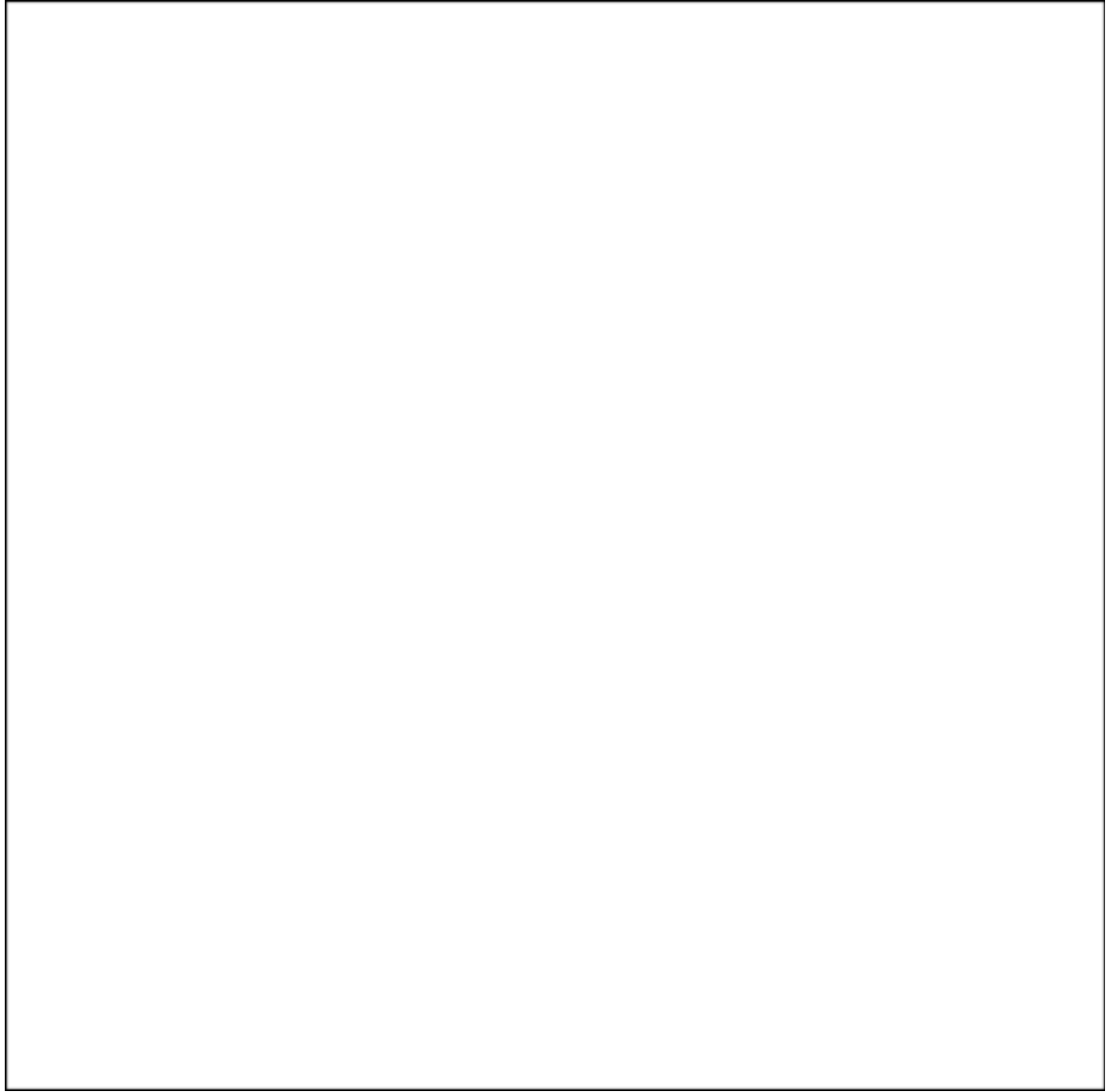
1  #include <stdio.h>
2
3  int largest(int, int, int); int main()
4
5  {
6
7      int a, b, c;
8
9      scanf("%d%d%d", &a, &b, &c);
10
11     printf("Largest of the given three numbers = %d ",largest(a,b,c));
12
13     return 0;
14
15 }
16
17 int largest(int x, int y, int z)
18
19 {
20
21     if ((x>y) && (x > z))
22
23     {
24
25         return x;
26
27     } else if (y > z) {
28
29         return y;
30     }
31     else
32     {
33
34         return z;
35
36     }
37
38 }
39
40
41 }

```

	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



```

1  #include <stdio.h>
2
3  int factorial(int);
4
5  int main()
6
7  {
8
9  int number;
10
11 scanf("%d", &number);
12
13 printf("Factorial of a given number %d = %d", number, factorial(number));
14 return 0;
15
16 }
17
18 int factorial(int n)
19
20 {
21
22 int i, fact = 1; for (i=1; i<=n; i++)
23
24 {
25
26 fact*=i;
27
28 }
29
30
31 return fact;
32
33 // write the return statement
34
35 }

```

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

Passed all tests! ✓

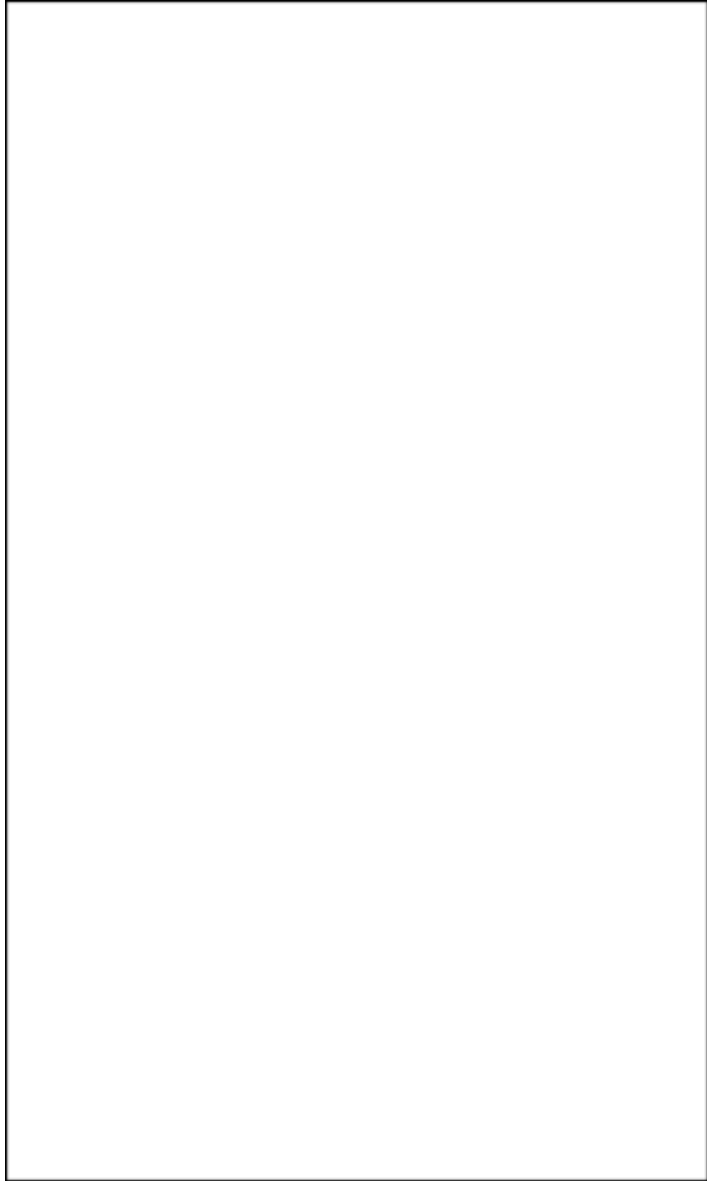
14) Write a C program to demonstrate functions without arguments and with return value.

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

Write the function prime().

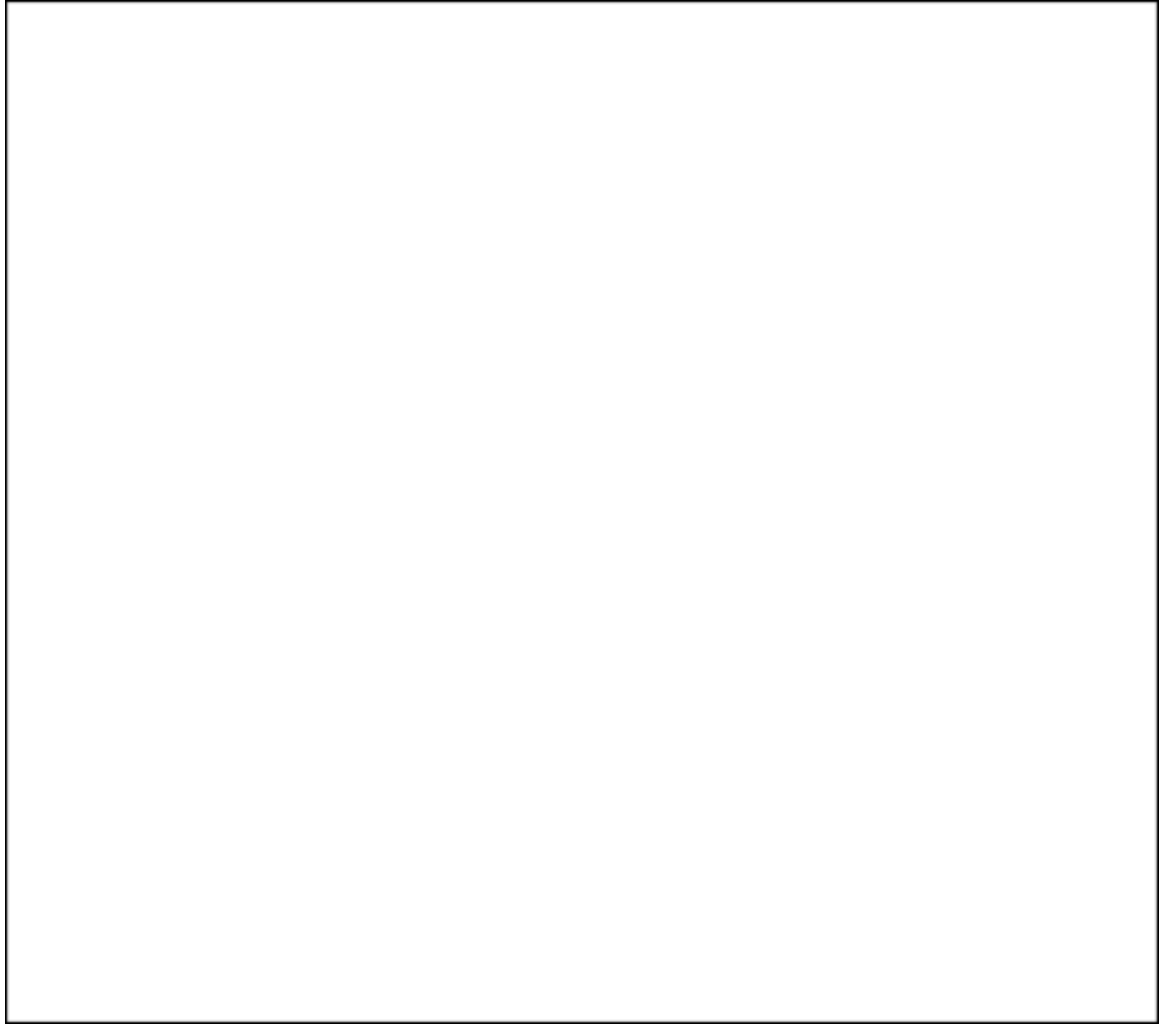
Sample Input and Output:

The given number is a prime number



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





```

35 scanf("%d",&a);
36
37 if(a<=1)
38
39 {
40
41 return 1;
42
43 }
44
45 for(i=2;i<a;i++)
46
47 {
48
49 if(a%i==0)
50
51 {
52
53 return 1;
54
55 }
56
57 }
58
59 return 0;
60
61 }

```

	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	✓

