GE23131-Programming Using C-2024-2025

Week 3 Assessment

Decision Making – if , if…else , if…else if

ABENANTHAN P

240701005

Admission Elgibilities

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

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int m,p,c;
    scanf("%d %d %d",&m,&p,&c);
    if((m>=65&&p>=55&&c>=50)||((m+p+c)>=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! ✓

Finding Second Largest Number

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

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int num,largest,seclar;

    largest = seclar = -1000000;

    while (1){
        scanf("%d",&num);
        if(num==-1){
            break;
        }
        if(num>largest){
            seclar=largest;
            largest=num;
        }else if (num > seclar && num!=largest){
            seclar=num;
        }
    }
    printf("%d\n",seclar);
}
```

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

Passed all tests! ✓

Triangle The Smallest Sides

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 %)

```c
#include<stdio.h>
int main()
{
    int X,Y,Z;
    scanf("%d %d %d",&X,&Y,&Z);
    if(X<=Y&&X<=Z){
        printf("%d\n",X);
    } else if(Y<=Z){
        printf("%d\n",Y);
    } else {
        printf("%d\n",Z);
    }
    return 0;
}
```

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

Passed all tests! ✓

Formal And Actual Agreements

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);
}
```

```c
#include<stdio.h>

int sum(int);

int main()
{
    int num;
    scanf("%d",&num);
    printf("Sum of %d natural numbers = %d\n",num,sum(num));
    return 0;
}
int sum(int value)

{
    int i,tot=0;
    for(i=1;i<=value;i++)
    {
        tot=tot+i;
    }
    return(tot);
}
```

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

Passed all tests! ✓

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:

```c
#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 %)

```
#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! ✓

Local And Global Variables

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);
}
```

```c
#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);
}
```

	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! ✓

Local And Global Variables

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;
}
```

```c
#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;

}
```

	Expected	Got	
✓	In main() function x = 10 In change1() function x = 10 In change2() function x = 15 In main() function x = 10	In main() function x = 10 In change1() function x = 10 In change2() function x = 15 In main() function x = 10	✓

Passed all tests! ✓

Different Categories Of Functions

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:

```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");
}
```

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

void india_capital(void);

int main()
{
    india_capital();
    return 0;
}

void india_capital(void)
{
    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! ✓

Different Categories Of Functions

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);
void hello(void);
// Write the functions

int main()
{
    print();
    hello();
    print();
    return 0;
}
void print(void)
{
    printf("...***...\n");
}
void hello(void){
    printf("Hello! REC\n");
}
```

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

Passed all tests! ✓

Different Categories Of Function

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); // Correct the code
    return 0;
}

void largest(int x, int y)
{
    // Correct the code
    if (x>y)
    {
        // Correct the code
        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! ✓

Different Categories Of 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:

```c
#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(void)
{
    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! ✓

Different Categories Of Function

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;
        }
}
```

```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\n", largest(a,b,c)); // Correct the code
    return 0;
}

int largest(int x, int y,int z)
{
    // Correct the code
    if ((x>y)&&(x>z))
    {
        // Correct the code
        return x; // Correct the code
    }
    else if (y>z)
    {
        // Correct the code
        return y ; // Correct the code
    }
    else
    {
        return z; // Correct the code
    }
}
```

	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! ✓

Different Categories Of Function

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

```c
#include <stdio.h>

int factorial(int);

int main()
{
    int number;
    scanf("%d", &number);
    printf("Factorial of a given number %d = %d\n", number, factorial(number));
    return 0;
}

int factorial(int n)
{
    int i, fact = 1;
    for (i=1;i<=n;i++)
    {
        fact*=i;// Write code to calculate the factorial of a given number
    }
    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! ✓

Different Categories Of Functions

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
```

```c
#include <stdio.h>

int prime();

int main()
{
    if (prime() == 0)
    {
        printf("The given number is a prime number\n");
    }
    else
    {
        printf("The given number is not a prime number\n");
    }
    return 0;
}
int prime()
{
    int num,i;
    scanf("%d",&num);
    if(num<=1)
    return 1;
    for(i=2;i*i<=num;i++){
        if (num%i==0){
            return 1;
        }
    }
    return 0;
}

// Write the function prime()

```

	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! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code which counts the number of vowels, consonants, digits and spaces are presented in a given string.

Initially, the variables vowels, consonants, digits and spaces are initialized to 0.

Iterate the string from the **first** character to **last** character to find all vowels, consonants, digits and spaces.

When a vowel character is found, vowel variable is incremented by 1. Similarly, consonants, digits and spaces are incremented when these characters are found in the string.

Finally, the count is displayed on the screen.

For example:

Input	Result
kohli hits 100 in every cricket match!	Vowels = 9 Consonants = 19 Digits = 3 White spaces = 6

```c
#include <stdio.h>

int main()
{
    char s[100];
    int i, vowels = 0, consonants = 0, digits = 0, spaces = 0;
    fgets(s, sizeof(s), stdin);
    for (i=0;s[i]!='\0';i++)
    { // Complete the code in for
        if (s[i]=='a'||s[i]=='e'||s[i]=='i'||s[i]=='o'||s[i]=='u'||s[i]=='A'||s[i]=='E'||s[i]=='I'||s[i]=='O'||s[i
        { // Write the condition part
            ++vowels;
        }
        else if ((s[i]>='a'&&s[i]<='z')||(s[i]>='A'&&s[i]<='Z') )
        { // Write the condition part
            ++consonants;
        }
        else if ((s[i]>='0'&&s[i]<='9'))
        { // Write the condition part
            ++digits;
        }
        else if (s[i]==' '||s[i]=='\t' )
        { // Write the condition part
            ++spaces;
        }
    }
    printf("Vowels = %d\n", vowels);
    printf("Consonants = %d\n",consonants);
    printf("Digits = %d\n",digits);
    printf("White spaces = %d", spaces);
    return 0;
}
```

	Input	Expected	Got	
✓	kohli hits 100 in every cricket match!	Vowels = 9 Consonants = 19 Digits = 3 White spaces = 6	Vowels = 9 Consonants = 19 Digits = 3 White spaces = 6	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code which copies a given string into another string.

Initially, read a string from the standard input device and write a loop to copy each character of given string into another string till the end of the string is reached.

Place '\0' at the end of the copied string.

Finally, the copied string is displayed on the screen.

For example:

Input	Result
GangaRiver	The copied string = GangaRiver

```c
#include <stdio.h>

int main()
{
    char str1[50], str2[50];
    int i;
    scanf("%s", str1);
    for (i=0;str1[i]!='\0';i++)
    { //Complete the code in for
       str2[i]=str1[i];

    }
    str2[i] ='\0' ; //Complete the statement
    printf("The copied string = %s\n", str2);
    return 0;
}
```

	Input	Expected	Got	
✓	GangaRiver	The copied string = GangaRiver	The copied string = GangaRiver	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code which concatenates two given strings and store the result in another string.

Read two strings from the standard input device and write a loop to copy each character of the first string into third string till the end of the first string.

Write another loop to copy each character of the second string into third string till the end of second string.

Now place '\0' at the end of the third string.

Finally, display the third string.

For example:

Input	Result
Narendra Modi	NarendraModi

```c
#include <stdio.h>

int main()
{
    char a[20], b[20], c[20];
    int i, j;
    scanf("%s", a);
    scanf("%s", b);
    for (i=0;a[i]!='\0';i++)
    { // Complete the code in for
        c[i] =a[i]; //Complete the statement
    }
    for (j=0;b[j]!='\0';j++)
    { // Complete the code in for
        c[i] =b[j]; //plete the statement
        i++;
    }
    c[i] ='\0';//mplete the statement
    printf("%s\n", c);
    return 0;
}
```

	Input	Expected	Got	
✓	Narendra Modi	NarendraModi	NarendraModi	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code to check whether the given two strings are equal or not.

Read two strings from the standard input device and write a loop to check each character of the first string with second string till the end of the first string is reached.

If any character is not equal then break the loop and say "**Two strings are not equal**".

If all the characters are equal and the length of two strings is also equal then display "**Two strings are equal**".

For example:

Input	Result
Godavari Godavari	Two strings are equal
Narmada narmada	Two strings are not equal

```c
#include <stdio.h>

int main()
{
    char a[20], b[20];
    int i = 0, flag = 0;
    scanf("%s", a);
    scanf("%s", b);
    while (a[i]!='\0'&&b[i]!='\0')
    { //Complete the condition part
        if (a[i]!=b[i])
        { //Complete the condition part
            flag = 1; //Complete the statement
            break;
        }
        i++;
    }
    if (flag==0&&a[i]=='\0'&&b[i]=='\0')
    { //Complete the condition part
        printf("Two strings are equal\n");
    }
    else
    {
        printf("Two strings are not equal\n");
    }
    return 0;
}
```

	Input	Expected	Got	
✓	Godavari Godavari	Two strings are equal	Two strings are equal	✓
✓	Narmada narmada	Two strings are not equal	Two strings are not equal	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code to search the occurrence of a given character in a given string.

Read a string and a character from the standard input device and write a loop to check each character of the string with a given character.

If the given character is equal to a character in the string then increment the count with in the loop.

Finally, display the count variable which has the total number of occurrences of the given character.

For example:

Input	Result
CurrencyDemonitisation n	Occurence of character 'n' in the given string CurrencyDemonitisation = 3

```c
#include <stdio.h>

int main()
{
    char str[20], ch;
    int count = 0, i;
    scanf("%s", str);
    scanf(" %c", &ch);
    for (i=0;str[i]!='\0';i++)
    { // Complete the code in for
        if (str[i]==ch )
        { // Write the condition part
            count++;
        }
    }
    if (count==0 )
    { // Write the condition part
        printf("The character '%c' is not presented in the string %s\n", ch, str);
    }
    else
    {
        printf("Occurence of character '%c' in the given string %s = %d\n", ch, str, count);
    }
    return 0;
}
```

	Input	Expected	Got
✓	CurrencyDemonitisation n	Occurence of character 'n' in the given string CurrencyDemonitisation = 3	Occurence of character 'n

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code to count total number of uppercase and lowercase characters from the accepted string.

Read a string from the standard input device and write a loop to check each character, whether it is uppercase or lowercase of the given string.

If the given character is uppercase then increment the upper_count with in the loop.

If the given character is lowercase then increment the lower_count with in the loop.

Finally display the upper_count and lower_count.

```c
#include<stdio.h>

int main()
{
    int upper_count = 0, lower_count = 0;
    char ch[80];
    int i;
    scanf("%s", ch); // Complete the statement
    i =0; // Complete the statement
    while (ch[i]!='\0')
    { // Write the condition part
        if (ch[i]>='A'&&ch[i]<='Z')
        { // Write the condition part
            upper_count++;
        }
        if (ch[i]>='a'&&ch[i]<='z')
        { // Write the condition part
            lower_count++;
        }
        i++;
    }
    printf("Number of uppercase letters = %d\n",upper_count );
    printf("Number of lowercase Letters = %d\n",lower_count );
    return 0;
}
```

	Input	Expected	Got	
✓	KrishnaAndGodavariAreRivers	Number of uppercase letters = 5 Number of lowercase Letters = 22	Number of uppercase letters = 5 Number of lowercase Letters = 22	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code to reverse the given string.

Hints
Step:1 Read a string from the standard input device.
Step:2 Write a loop to find the length of the string.
Step:3 Write another loop to interchange the characters from first to last of the string.
Step:4 Finally display the reverse of a string.

For example:

Input	Result
Software	The reverse of a given string : erawtfoS

```c
#include<stdio.h>

int main()
{
    char ch[80], temp;
    int i, j;
    scanf("%s", ch);
    i = j = 0;
    while (ch[j]!='\0')
    { // Write the condition part
        j++;
    }
    j--;
    while (i<j )
    { // Write the condition part
        temp =ch[i] ; // Complete the statement
        ch[i] =ch[j] ; // Complete the statement
        ch[j] =temp ; // Complete the statement
        i++;
        j--;
    }
    printf("The reverse of a given string : %s\n", ch);
    return 0;
}
```

	Input	Expected	Got	
✓	Software	The reverse of a given string : erawtfoS	The reverse of a given string : erawtfoS	✓

Passed all tests! ✓

Problem Solving With Strings

Fill in the missing code in the below sample code to check whether the given string is a palindrome or not.

Read a string from the standard input device and write a loop to check the characters of the given string with the reverse string.

If all the characters are equal then display "**The given string is a palindrome**", otherwise display "**The given string is not a palindrome**".

For example:

Input	Result
12321	The given string 12321 is a palindrome
amaravathi	The given string amaravathi is not a palindrome

```c
#include <stdio.h>

int main()
{
    char ch[80];
    int i, j, length, flag = 0;
    scanf("%s",ch ); // Complete the statement
    length = 0;
    while (ch[length]!='\0' )
    { //Write the condition part
        length++;
    }
    for (i=0,j=length-1;i<j;i++,j-- )
    { // Complete the code in for
        if (ch[i]!=ch[j] )
        { // Write the condition part
            flag++;
            break;
        }
    }
    if (flag==0 )
    { // Write the condition part
        printf("The given string %s is a palindrome\n",ch  ); // Complete the statement
    }
    else
    {
        printf("The given string %s is not a palindrome\n", ch ); // Complete the statement
    }
    return 0;
}
```

	Input	Expected	Got	
✓	12321	The given string 12321 is a palindrome	The given string 12321 is a palindrome	✓
✓	amaravathi	The given string amaravathi is not a palindrome	The given string amaravathi is not a palindrome	✓

Passed all tests! ✓

String Manipulation Function

In **C** language, we have four types of string functions that are used for performing **string operations**. They are strlen(), strcpy(), strcat(), strcmp().

The function strlen() is used to find the **length** of the given string. This function returns only the **integer data** (or) **numeric data**.

The function strlen() counts the number of characters in a given string and returns the integer value.

It stops counting the character when **NULL** character is found. Because, **NULL** character indicates the end of the string in **C**.

The syntax of strlen() is integer_variable = strlen(string);.

Here string is a group of characters, strlen() function finds the **length** of the string and the **integer** value will be stored in the integer_variable.

The string.h header file supports all the string functions in **C** language.

Fill in the missing code in the below program to find the **length** of a string using **strlen()** function.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char ch[20];
    int l;
    scanf("%s", ch);
    l=strlen(ch);
    printf("The length of the string %s is %d\n", ch,l ); //Correct the code
    return 0;
}
```

	Input	Expected	Got	
✓	NarendraModi	The length of the string NarendraModi is 12	The length of the string NarendraModi is 12	✓

Passed all tests! ✓

String Manipulation Function

The function strcpy() is used to **copy** one string into another string including the NULL character (terminator char '\0').

The syntax of strcpy( ) is strcpy(string1, string2);.

Where string1, string2 are two strings and the string2 is copied into string1. In this case the copied string is available in string1 and both strings contains the same data.

If the length of string1 is less than the length of string2 then entire string2 value will not be copied into string1.

For example, consider the length of string1 is **20** and the length of string2 is **30**. Then, only the first **20** characters from string2 will be copied into string1, the remaining **10** characters will not be copied and will be **truncated**.

Understand and retype the below code which demonstrates the usage of **strcpy()** function.

```
#include <stdio.h>
#include <string.h>

int main()
{
   char str1[20], str2[20];
   scanf("%s", str2);
   strcpy(str1, str2);
   printf("The copied string = %s", str1);
   return 0;
}
```

```c
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20],str2[20];
    scanf("%s",str2);
    strcpy(str1,str2);
    printf("The copied string = %s",str1);
    return 0;
}
```

	Input	Expected	Got	
✓	Rose	The copied string = Rose	The copied string = Rose	✓

Passed all tests! ✓

String Manipulation Function

The function strcat() is used to concatenate two strings into a single string.

The syntax of strcat() is strcat(string1, string2);.

where string1, string2 are two different strings. Here string2 is concatenated with string1, and the **concatenated string** is stored in string1.

In strcat() operation, **NULL character ('\0')** of string1 is **overwritten** by first character of string2 and **NULL character ('\0')** is appended (added) at the end of **new** string1 which is created after strcat() operation.

Fill the missing code in the below program to display the **concatenated** string using **strcat()** function.

Fill the missing code in the below program to display the **concatenated** string using **strcat()** function.

For example:

Input	Result
REC Chennai	RECChennai

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char str1[20], str2[20];
    scanf("%s", str1);
    scanf("%s", str2);
    strcat(str1,str2);
    //Concat str2 with str1
    printf("%s\n",str1); // Correct the code
    return 0;
}
```

	Input	Expected	Got	
✓	REC Chennai	RECChennai	RECChennai	✓

Passed all tests! ✓

String Manipulation Function

The function strcmp() is used for comparison of two strings and it always returns the numeric data. This function compares strings character by character using their ASCII values.

The syntax of strcmp() is variable_name = strcmp (string1, string2);.

Where string1, string2 are two strings and the variable is of **integer** datatype.

The comparison of two strings is dependent on the **alphabets (characters)** and not on the size (length) of the strings.

If the function strcmp() returns zero, both strings are **equal**.
If the function strcmp() returns a value which is less than zero, **string2** is higher than **string1** (because the ASCII value of first unmatched character of **string1** is less than the ASCII value of the corresponding character in **string2**)
If the function strcmp() returns a value which is greater than zero, **string1** is higher than **string2** (because the ASCII value of first unmatched character of **string1** is greater than the ASCII value of the corresponding character in **string2**)

Fill the missing code in the below program to compare two strings using **strcmp()** function.

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char a[20], b[20];
    int i;
    scanf("%s", a);
    scanf("%s", b);
    i=strcmp(a,b);//Compare two strings

    if (i==0)
    { // Correct the code
        printf("The given two strings are equal\n");
    }
    else if ( i>0)
    { // Correct the code
        printf("The string %s is higher than the string %s\n", a, b);
    }
    else
    {
        printf("The string %s is higher than the string %s\n", b, a);
    }
    return 0;
}
```

	Input	Expected	Got
✓	NarendraModi narendramodi	The string narendramodi is higher than the string NarendraModi	The string narendramodi is higher than the str
✓	Krishna Godavari	The string Krishna is higher than the string Godavari	The string Krishna is higher than the string G
✓	REC REC	The given two strings are equal	The given two strings are equal

Passed all tests! ✓