GE23131-Programming Using C-2024-2025

Week 2 Assessment

Operators and Expressions , Managing Input and Output Operations

Arithmetic Operators

An operator is a special symbol used to **manipulate data**. The data items that the operators act upon are called operands.

The operator that works on a single operand is called a unary operator and that which works on two operands is known as a binary operator.

C provides many types of operators. They are: Arithmetic, Unary, Relational and equality, Logical, Assignment, Conditional, Bitwise and Special operators.

In C, we have 5 arithmetic operators:

Operator	Description
+	Used for addition
-	Used for subtraction
*	Used for multiplication
/	Used for division
%	Remainder/Modulus operator for finding remainder

Arithmetic operators are applied on **numeric operands**. Thus the operands can be **integers**, **floats** or **characters** (Since a character is internally represented by its numeric code).

The **remainder operator** (%) requires that both the operands be **integers** and the second operand be **non-zero**. Similarly the **division operator** (/) requires that the second operand be **non-zero**.

The format for usage of arithmetic operator is as follows:

```
operand1operatoroperand2
```

According to the coding conventions in C, a single space should be provided to the left and to the right of an operator.

The table given below demonstrates the use of various **arithmetic operators** using two variables num1 and num2 of type int with values 10 and 3 respectively:

Expression	Result
num1 + num2	13
num1 - num2	7
num1 * num2	30
num1 / num2	3
num1 % num2	1

Read the code given below to understand the usage of **arithmetic operators**. Retype in the space provided.

```
#include <stdio.h>
int main()
{
   int num1 = 10, num2 = 3;
   printf("Addition Result = %d\n", (num1 + num2));
   printf("Subtraction Result = %d\n", (num1 - num2));
   printf("Multiplication Result = %d\n", (num1 * num2));
   printf("Division Result = %d\n", (num1 / num2));
   printf("Remainder = %d", (num1 % num2));
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int num1=10,num2=3;
    printf("Addition Result = %d\n",(num1 + num2));
    printf("Subtraction Result = %d\n",(num1 - num2));
    printf("Multiplication Result = %d\n",(num1*num2));
    printf("Division Result = %d\n",(num1/num2));
    printf("Remainder = %d",(num1 % num2));
    return 0;
}
```

	Expected	Got	
✓	Addition Result = 13 Subtraction Result = 7 Multiplication Result = 30 Division Result = 3 Remainder = 1	Addition Result = 13 Subtraction Result = 7 Multiplication Result = 30 Division Result = 3 Remainder = 1	✓

Passed all tests! ✓

Arithmetic Operators

Division of one integer by another integer is referred to as **integer** division. This operation always results in an integer with truncated quotient.

If a **division** operation is carried out with two **floating point numbers** or with one **floating point number** and one **integer**, the result will be a **floating point quotient**.

The table given below demonstrates the usage of various **arithmetic operators** using two variables num1 and num2 of type float with values 12.5 and 2.0 respectively:

Expression	Result
num1 + num2	14.500000
num1 - num2	10.500000
num1 * num2	25.000000
num1 / num2	6.250000

Note that the **remainder operator** (%) is not applicable for **floating point numbers**.

In the program given below, type the missing code to find the **result** of applying different **arithmetic operators** on **floating point numbers**.

```c
#include <stdio.h>

int main()
{
    float num1 = 12.5, num2 = 2.0;
    printf("Result of addition = %f \n", (num1 + num2));
    printf("Result of subtraction = %f \n", (num1 - num2));
    printf("Result of multiplication = %f \n", (num1 * num2));
    printf("Result of division = %f \n", (num1 / num2));
    return 0;
}
```

	Expected	Got	
✓	Result of addition = 14.500000 Result of subtraction = 10.500000 Result of multiplication = 25.000000 Result of division = 6.250000	Result of addition = 14.500000 Result of subtraction = 10.500000 Result of multiplication = 25.000000 Result of division = 6.250000	✓

Passed all tests! ✓

Arithmetic Operators

The table given below demonstrates the use of various **arithmetic operators** using two variables c1 and c2 of type char with values 'A' and 'D' respectively:

Expression	Result
c1	65
c1 + c2	133
c1 + c2 + 5	138
c1 + c2 + '5'	186

In the above examples, the character 'A' is substituted with its ASCII value **65** and 'D' is substituted with 68.
The character '5' is substituted with its ASCII value 53. The integer value 5 is used as it is.

The following table demonstrates the usage of various **arithmetic operators** using two variables a and b of type int with values 11 and -3 respectively:

Expression	Result
a + b	8
a - b	14
a * b	-33
a / b	-3
a % b	2

In the program given below, type the missing code to find the **result** of applying different **arithmetic operators** on **char** data type values.

```c
#include <stdio.h>

int main()
{
    char c1 = 'A', c2 = 'D';
    printf("c1 = %d\n", c1);
    printf("c1 + c2 = %d \n", (c1 + c2));
    printf("c1 + c2 + 5 = %d \n", (c1 + c2 + 5));
    printf("Result = %d", (c1 + c2 + '5'));
    return 0;
}
```

	Expected	Got	
✓	c1 = 65 c1 + c2 = 133 c1 + c2 + 5 = 138 Result = 186	c1 = 65 c1 + c2 = 133 c1 + c2 + 5 = 138 Result = 186	✓

Passed all tests! ✓

Comments And Tokens

As mentioned earlier, a computer program is a collection of instructions or statements.

A **C** program usually consists of multiple statements.
Each statement is composed of one or more of the **three** given below:

1. Comments
2. Whitespace characters
3. Tokens

In a computer program, a comment is used to mark a section of code as non-executable.

Comments are mainly used for two purposes:

1. To mark a section of executable code as non-executable, so that the compiler ignores it during compilation.
2. To provide remarks or an explanation on the working of the given section of code in plain English, so that a fellow programmer can read and understand the code.

In **C**, there are two types of comments:

1. **end-of-line comment** : It starts with //. The content that follows the // and continues till the end of that line is a comment. It is also called as **single-line comment**.
2. **traditional comment** : It starts with /* and ends with */. The content between /* and */ is the comment. It is also called as **multi-line comment**.

The code given below shows the two types of comments:

```
/*
    C programming language was developed by Dennis Ritchie.
    This is called a header comment which is used to describe
    what this program would do. As you can notice the comment is
    spanning across multiple lines.
*/
```

```
#include <stdio.h>

int main()
{
     int num1 = 10, num2 = 20;
     printf("sum of two numbers = %d", num1 + num2);
     return 0;
}//end of the main() function - this is an example of a end-of-line comment
```

Read the code given below to understand the different types of comments. Retype in the space provided.

Given below are 3 important points regarding comments:

1. There **should not** be any space between the two forward slashes in //, i.e., / / is incorrect. Similarly, there should not be any space between the **slash** and **star** characters in /* and */, i.e., / * and * / are incorrect.
2. **Comments do not nest**, i.e., /* and */ comment has no special meaning inside a // comment.Similarly, a // comment has no special meaning inside a /* comment.
3. One should not write comments inside character literals (i.e., characters enclosed between single-quotes). Comments inside String literals (i.e., text enclosed between double-quotes) are treated as part of the String's content.

Content to be reproduced

```
/*
  This is a sample C program
  developed by REC
*/
#include <stdio.h>
int main()
{
   // this is an end of line comment
   printf("I love C Language!");
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
/*
 This is a sample C program
 developed by REC
*/
#include<stdio.h>
int main()
{
    // this is an end of line comment
    printf("I love C Language!");
    return 0;
}
```

	Expected	Got	
✓	I love C Language!	I love C Language!	✓

Passed all tests! ✓

Comments And Tokens

Make the following changes in the code given below:

1. Comment the statement which prints "**Mango**".
2. Remove the comment on the statement which prints "**Banana**".

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    printf("Orange\n");
    //printf("Orange")
    printf("Banana\n");
    //printf("Banana");
    return 0;
}
```

	Expected	Got	
✓	Orange Banana	Orange Banana	✓

Passed all tests! ✓

Comments And Tokens

The code given below contains text that prints "**DennisRitchieBrianKernighan**".

Make the suggested changes to the code so that it prints "**DennisRitchieBrianKernighan**" as shown below.

```
Dennis Ritchie
Brian Kernighan
```

To make the required changes, follow the steps given below to introduce the SPACE character and the \n new line character appropriately:

1. Insert a space between "Dennis" and "Ritchie". Make sure that no extra space or any other character apart from space are inserted.
2. Insert a \n between "Ritchie" and "Brian" Make sure that no extra space or any other character apart from \n are inserted.
3. Insert a space between "Brian" and "Kernighan". Make sure that no extra space or any other character apart from space are inserted.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    printf("Dennis Ritchie\nBrian Kernighan");
    return 0;
}
```

	Expected	Got	
✓	Dennis Ritchie Brian Kernighan	Dennis Ritchie Brian Kernighan	✓

Passed all tests! ✓

Comments And Tokens

In **C**, the backslash character \ is used to mark an escape sequence. An **Escape Sequence** is an escape character \ followed by a normal character. For example: \n or \t.

The presence of the escape character changes the meaning of the character which follows it. For example, when the string literal "Hello\tWorld" is printed, the result is seen as

```
Hello	World
```

In the string literal "Hello**\t**World", \t represents the **TAB** character.

Similarly, if we want to print a **double quote** inside a double-quoted string literal, we need to escape the **double quote** by using the escape character \. For example :

```
printf("Hello \" (Quote)");
```

The code given above will produce the following output:

```
Hello " (Quote)
```

Given below are a few points regarding **escape sequences**:

- Each escape sequence has a unique ASCII value as shown in the table given below.
- Each and every combination of an escape sequence starts with backslash \.
- Although an escape sequence consists of two characters, it represents a single special character in the given context.

Escape sequences and their ASCII codes:

Character	Bell	Backspace	Horizontal tab	Vertical tab	New line	Form feed	Carriage return	Double Quotation	Single Quotation	Question mark	Backslash	Null
Escape Sequence	\a	\b	\t	\v	\n	\f	\r	\"	\'	\?	\	\0
ASCII value	007	008	009	011	010	012	013	034	039	063	092	000

Read the code given below and retype in the space provided. **Note** the effects of \t and \n in the resulting output when executed successfully.

Content to be reproduced

```
#include <stdio.h>

int main()
{
   printf("One Two");
   printf("Three\n");
   printf("Four\nFive\n");
   return 0;
}
```

Answer: (penalty regime: 0 %)

```
1  #include<stdio.h>
2  int main()
3  {
4      printf("One Two");
5      printf("Three\n");
6      printf("Four\nFive\n");
7      return 0;
8  }
```

	Expected	Got	
✓	One TwoThree Four Five	One TwoThree Four Five	✓

Passed all tests! ✓

Variables And Keyboards

Read the code given below to learn naming conventions in identifiers.

For example, consider the program given below:

```
#include <stdio.h>

int main()
{
        int age = 2; // age is an integer variable

        int firstNumber = 2; // firstNumber is an integer variable

        // If there are two or more words in an identifier/variable - User can also use "camel case" style to declare a variable.

        int second_number = 3; // second_number is an integer variable

        // Any space cannot be used between two words of an identifier/variable; User can use underscore (_) instead of space.

        int _i_am_also_a_valid_identifier = 4; // _i_am_also_a_valid_identifier is an integer variable

        // An identifier/variable name must be start with an alphabet or underscore (_) only, no other special characters, digits

        printf("age = %d\n", age);
        printf("firstNumber = %d\n", firstNumber);
        printf("second_number = %d\n", second_number);
        printf("_i_am_also_a_valid_identifier = %d\n", _i_am_also_a_valid_identifier);
        return 0;
}
```

Answer: (penalty regime: 0 %)

Reset answer

```
#include <stdio.h>

int main()
{
    int age = 2;
    int firstNumber = 2;
    int second_number = 3;
    int _i_am_also_a_valid_identifier = 4;
    printf("age = %d\n",age ); // Fill in the missing code
    printf("firstNumber = %d\n",firstNumber); // Fill in the missing code
    printf("second_number = %d\n",second_number ); // Fill in the missing code
    printf("_i_am_also_a_valid_identifier = %d\n",_i_am_also_a_valid_identifier ); // Fill in the missing code
    return 0;
}
```

	Expected	Got	
✓	age = 2 firstNumber = 2 second_number = 3 _i_am_also_a_valid_identifier = 4	age = 2 firstNumber = 2 second_number = 3 _i_am_also_a_valid_identifier = 4	✓

Syntax of Main Functions

Identify and correct the error in the code given below.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>
int main ()
{
    printf("Hello, float data type allocates 4 bytes in memory");
    return 0;
}
```

	Expected	Got	
✓	Hello, float data type allocates 4 bytes in memory	Hello, float data type allocates 4 bytes in memory	✓

Passed all tests! ✓

Syntax of Main Functions

Identify and correct the error in the code given below.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    printf("Hello, # is a preprocessor in C");
    return 0;
}
```

	Expected	Got	
✓	Hello, # is a preprocessor in C	Hello, # is a preprocessor in C	✓

Passed all tests! ✓

Syntax Of Main Function

Identify and correct the error in the code given below.

Answer: (penalty regime: 0 %)

Reset answer

```
#include <stdio.h>
int main()
{
    printf("Hello, I am learning C Language!");
    return 0;
}
```

	Expected	Got	
✓	Hello, I am learning C Language!	Hello, I am learning C Language!	✓

Passed all tests! ✓

Syntax Of Main Function

In **C** programming language, execution of the code starts with a function called main.

We shall learn more about functions in the later sections. For now, we can safely assume that **function** is the name given to a set of one or more executable statements. main() is a **user defined function**, i.e., a user (a programmer) writes the code for the main() function.

While executing a **C** program, the **Operating System (OS)** only calls the main() function in that program.

When the **OS** executes a program, the program usually returns an integer value 0 if the execution of that program is successful.

In **C**, **main()** can be written in such a way that it returns a an int.

```
#include <stdio.h>

int main()
{
    printf("Sample main() function with int as return type!");
    return 0;// 0 value indicates that the execution is successful

}
```

If the programmer does not specify any return type, the return type is by default considered as int.

The name of the main() function should always be in lowercase, i.e., if a function is written as Main(), it is not the main function which is called by the **OS**.

Read the code given below to familiarize yourself with the syntax of main() function. Retype in the space provided.

```
#include <stdio.h>

int main()
{
   printf("Impossible is nothing!");
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    printf("Impossible is nothing!");
    return 0;
}
```

	Expected	Got	
✓	Impossible is nothing!	Impossible is nothing!	✓

Passed all tests! ✓

Syntax Of Main Function

Click on **Check** without correcting the code.

This results in many errors because the main function is not defined correctly.

Now, correct the spelling of the main function and submit the program once again.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    printf("Correct Me!");
    return 0;
}
```

	Expected	Got	
✓	Correct Me!	Correct Me!	✓

Int Data Type

In the program given below, we shall learn how to assign values to int data type from binary, octal, hex and character literals.

Read the code given below and retype in the space provided.

```
#include <stdio.h>
int main()
{
   int binaryThree = 0b11;
   printf("binaryThree value = %d\n", binaryThree);
   int octalEight = 010;
   printf("octalEight value = %d\n", octalEight);
   int hexTen = 0xA;
   printf("hexTen value = %d\n", hexTen);
   int asciiValueOfOne = '1';
   printf("asciiValueOfOne value = %d\n", asciiValueOfOne);
   int asciiValueOfA = 'A';
   printf("asciiValueOfA value = %d\n", asciiValueOfA);
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int binaryThree=0b11;
    printf("binaryThree value = %d\n",binaryThree);
    int octalEight =  010;
    printf("octalEight value = %d\n",octalEight);
    int hexTen = 0xA;
    printf("hexTen value = %d\n",hexTen);
    int asciiValueOfOne = '1';
    printf("asciiValueOfOne value = %d\n",asciiValueOfOne);
    int asciiValueOfA = 'A';
    printf("asciiValueOfA value = %d\n",asciiValueOfA);
    return 0;
}
```

	Expected	Got	
✓	binaryThree value = 3 octalEight value = 8 hexTen value = 10 asciiValueOfOne value = 49 asciiValueOfA value = 65	binaryThree value = 3 octalEight value = 8 hexTen value = 10 asciiValueOfOne value = 49 asciiValueOfA value = 65	✓

Passed all tests! ✓

Int Data Type

In the program given below, fill in the missing code to add two integer numbers.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int num1 = 15, num2 = 25, sum;
    printf("Given integers are num1 = %d, num2 = %d\n", num1, num2);
    //Write the code to add num1 and num2 and place the result in the variable sum
    sum=num1+num2;
    printf("Sum of 2 given numbers = %d\n", sum);
    return 0;
}
```

	Expected	Got	
✓	Given integers are num1 = 15, num2 = 25 Sum of 2 given numbers = 40	Given integers are num1 = 15, num2 = 25 Sum of 2 given numbers = 40	✓

Passed all tests! ✓

Int Data Type

To print unsigned values on the console, use %u format character instead of %d in the **printf()** function.

Whenever an attempt is made to assign a negative number to an **unsigned int** (For eg: unsigned int num = -1;) the compiler does not flag it as an **error**. Instead, it will automatically convert the negative number to a positive number as shown below:

```
unsigned int num = -1;
The value stored in num = unsigned int maximum_value + 1 - num;
The final value in num = 4294967295 (in a 32-bit processing system)
```

In the program given below, fill in the missing **format characters** to print **signed** and **unsigned** values.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    signed int number1 = -20, number2 = 20;
    unsigned int number3 = -1, number4 = 1;
    printf("Given signed values are %d and %d\n", number1, number2); // Fill the correct format character after %
    printf("Given unsigned values are %u and %u\n", number3, number4); // Fill the correct format character after %
    return 0;
}
```

	Expected	Got	
✓	Given signed values are -20 and 20 Given unsigned values are 4294967295 and 1	Given signed values are -20 and 20 Given unsigned values are 4294967295 and 1	✓

Passed all tests! ✓

Int Data Type

Identify the error and correct the code. [**Hint**: Verify if all variables are declared before they are first used.]

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int number1 = 20, number2 = 30;
    int sub;
    sub = number1 - number2;
    printf("The difference of the two given numbers = %d\n", sub);
    return 0;
}
```

	Expected	Got	
✓	The difference of the two given numbers = -10	The difference of the two given numbers = -10	✓

Passed all tests! ✓

Float Data Type

Identify and correct the errors in the code given below:

Expected Output:

Given float values are num1 = 5.340000, num2 = 125.789001
The result after dividing in float format = 23.555992
The result after dividing in exponential format = 2.355599e+01

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    float num1 = 5.340000, num2 = 125.789001, result;
    printf("Given float values are num1 = %f, num2 = %f\n", num1, num2);
    result = num2 / num1;
    printf("The result after dividing in float format = %f \n",result);
    printf("The result after dividing in exponential format = %e\n",result);
    return 0;
}
```

	Expected	Got	
✓	Given float values are num1 = 5.340000, num2 = 125.789001 The result after dividing in float format = 23.555992 The result after dividing in exponential format = 2.355599e+01	Given float values are num1 = 5.340000, num2 = 125.789001 The result after dividing in float format = 23.555992 The result after dividing in exponential format = 2.355599e+01	✓

Passed all tests! ✓

Float Data Type

Identify and correct the errors in the code given below:

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    float num1 = 5.345f, num2 = 12.4, result;
    printf("Given float values are num1 = %f, num2 = %f\n", num1, num2);
    result = num1 / num2;
    printf("Result of division = %f\n", result);
    return 0;
}
```

	Expected	Got	
✓	Given float values are num1 = 5.345000, num2 = 12.400000 Result of division = 0.431048	Given float values are num1 = 5.345000, num2 = 12.400000 Result of division = 0.431048	✓

Passed all tests! ✓

Relational and Operators

Relational and equality operators are used to **test** or **compare** two numeric values or numeric expressions.

In **C**, **Relational and equality operators** when applied on the operands, produce an **integer** value which is either 0 or 1 and these are often referred to as logical values. The value 0 represents false and the value 1 represents true.

In **C**, there are **four** relational and **two** equality operators as given below:

Operator	Description
>	Checks for greater-than condition
>=	Checks for greater-than-or-equals condition
<	Checks for less-than condition
<=	Checks for less-than-or-equals condition.
==	Checks if two values are equal
!=	Checks if two values are unequal

The format for usage of **relational** and **equality operators** is as follows:

`operand1operatoroperand2`

According to the coding conventions in C, a single space should be provided to the left and to the right of the operator.

The table given below demonstrates the use of various **relational and equality operators** using variables int num1 = 7;, float num2 = 5.5;, char ch = 'w':

Expression	Interpretation	Result Value
(num1 > 5)	true	1
((num1 + num2) <= 10)	false	0
(ch == 119)	true	1
(ch != 'p')	true	1
(ch >= 10 * (num1 + num2))	false	0

Read the code given below and retype in the space provided.

```c
#include <stdio.h>


int main()
{
   int num1 = 7;
   float num2 = 5.5;
   char ch = 'w';
   printf("Result1 = %d\n", (num1 > 5));
   printf("Result2 = %d\n", ((num1 + num2) <= 10));
   printf("Result3 = %d\n", (ch == 119));
   printf("Result4 = %d\n", (ch != 'p'));
   printf("Result5 = %d", (ch >= 10 * (num1 + num2)));
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int num1 = 7;
    float num2 =5.5;
    char ch = 'w';
    printf("Result1 = %d\n",(num1 > 5));
    printf("Result2 = %d\n",((num1 + num2) <= 10));
    printf("Result3 = %d\n",(ch == 119));
    printf("Result4 = %d\n",(ch !='p'));
    printf("Result5 = %d\n",(ch >= 10*(num1+ num2)));
}
```

	Expected	Got	
✓	Result1 = 1 Result2 = 0 Result3 = 1 Result4 = 1 Result5 = 0	Result1 = 1 Result2 = 0 Result3 = 1 Result4 = 1 Result5 = 0	✓

Passed all tests! ✓

Logical Operators

Logical operators are used to perform logical operations on the given expressions.

An expression containing a logical operator returns either 0 (or) 1 depending on the evaluation of the expression to either false or true respectively.

Note: In C, false is represented as 0 (zero) and all non-zero values can be treated as true.

Given below are the **three** logical operators in **C**:

Operator	Description	Meaning
&&	logical AND	It returns true when both conditions are true, else, it returns false
||	logical OR	It returns true if atleast one of the conditions is true
!	logical NOT	It returns true when the given expression is false and returns false when the given expression is true

According to the coding conventions in C, a single space should be provided to the left and to the right of the operator.

The below table demonstrates the use of various **relational and equality operators** using variables int num1 = 7;, float num2 = 5.5;, char ch = 'w':

Expression	Interpretation	Result Value
(num1 >= 6) && (ch == 'w')	true	1
(num2 < 11) && (num1 > 100)	false	0
(ch != 'p') || ((num1 + num2) <= 10)	true	1
!(num1 > (num2 + 1))	false	0
!(num1 <= 3)	true	1

Read the code given below and retype in the space provided.

```c
#include <stdio.h>

int main()
{
   int num1 = 7;
   float num2 = 5.5;
   char ch = 'w';
   printf("Result1 = %d\n", ((num1 >= 6) && (ch == 'w')));
   printf("Result2 = %d\n", ((num2 < 11) && (num1 > 100)));
   printf("Result3 = %d\n", ((ch != 'p') || ((num1 + num2) <= 10)));
   printf("Result4 = %d\n", !(num1 > (num2 +1)));
   printf("Result5 = %d\n", !(num1 <= 3));
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int num1 = 7;
    float num2 = 5.5;
    char ch = 'w';
    printf("Result1 = %d\n",((num1>=6)&&(ch == 'w')));
    printf("Result2 = %d\n",((num2 < 11)&&(num1 > 100)));
    printf("Result3 = %d\n",((ch != 'p')|| ((num1 + num2)<=10)));
    printf("Result4 = %d\n",!(num1 > (num2 + 1)));
    printf("Result5 = %d\n",!(num1 <= 3));
    return 0;
}
```

	Expected	Got	
✓	Result1 = 1 Result2 = 0 Result3 = 1 Result4 = 0 Result5 = 1	Result1 = 1 Result2 = 0 Result3 = 1 Result4 = 0 Result5 = 1	✓

Passed all tests! ✓

Unary Operators

Read the code given below to understand the working of unary operators. Retype in the space provided.

```c
#include <stdio.h>


int main()
{
   int x = 16;
   printf("+x = %d\n", (+x));
   printf("-x = %d\n", (-x));
   printf("x = %d\n", x);
   printf("++x = %d\n", (++x));
   printf("x = %d\n", x);
   printf("x++ = %d\n", (x++));
   printf("x = %d\n", x);
   printf("--x = %d\n", (--x));
   printf("x = %d\n", x);
   printf("x-- = %d\n", (x--));
   printf("x = %d", x);
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int x= 16;
    printf("+x = %d\n",(+x));
    printf("-x = %d\n",(-x));
    printf("x = %d\n",x);
    printf("++x = %d\n",(++x));
    printf("x = %d\n",x);
    printf("x++ = %d\n",(x++));
    printf("x = %d\n",x);
    printf("--x = %d\n",(--x));
    printf("x = %d\n",x);
    printf("x-- = %d\n",(x--));
    printf("x = %d",x);
}
```

	Expected	Got	
✓	+x = 16 -x = -16 x = 16 ++x = 17 x = 17 x++ = 17 x = 18 --x = 17 x = 17 x-- = 17 x = 16	+x = 16 -x = -16 x = 16 ++x = 17 x = 17 x++ = 17 x = 18 --x = 17 x = 17 x-- = 17 x = 16	✓

Passed all tests! ✓

Unary Operators

Read the code given below to understand the working of **increment** and **decrement** operators. Retype in the space provided.

```
#include <stdio.h>

int main()
{
   int x = 4, y;
   y = x++;
   printf("y = %d x = %d\n", y, x);
   y = ++x;
   printf("y = %d x = %d\n", y, x);
   y = x--;
   printf("y = %d x = %d\n", y, x);
   y = --x;
   printf("y = %d x = %d\n", y, x);
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int x = 4,y;
    y=x++;
    printf("y = %d x = %d\n",y,x);
    y = ++x;
    printf("y = %d x = %d\n",y,x);
    y= x--;
    printf("y = %d x = %d\n", y,x);
    y = --x;
    printf("y = %d x = %d\n",y,x);}
```

	Expected	Got	
✓	y = 4 x = 5 y = 6 x = 6 y = 6 x = 5 y = 4 x = 4	y = 4 x = 5 y = 6 x = 6 y = 6 x = 5 y = 4 x = 4	✓

Passed all tests! ✓

Assignment Operator

Read the code given below to understand the usage of the assignment operator. Retype in the space provided.

```
#include <stdio.h>

int main()
{
   int x = 24, y = 39, z = 45;
   z = x + y;
   y = z - y;
   x = z - y;
   printf("x = %d y = %d z = %d", x, y, z);
   return 0;
}
```

Answer: (penalty regime: 0 %)

```
#include<stdio.h>
int main()
{
    int x=24,y=39,z=45;
    z=x+y;
    y=z-y;
    x=z-y;
    printf("x = %d y = %d z = %d",x,y,z);
}
```

	Expected	Got	
✓	x = 39 y = 24 z = 63	x = 39 y = 24 z = 63	✓

Passed all tests! ✓

Ternary Operator

C language provides an operator to evaluate conditions. It is called a **conditional operator** (?:) or a **ternary operator**.

Ternary operator needs exactly **three** operands to compute the result.

The syntax for using a **ternary operator** is:

```
condition? expression1 : expression2
```

The condition should always evaluate to 1 (true) or 0 (false). If the condition evaluates to true, then **expression1** is evaluated and its value is returned, otherwise **expression2** is evaluated and its value is returned.

Given below is an example demonstrating the usage of a **conditional operator**:

```
int marks = 75, pass_marks = 50;
(marks > pass_marks) ? printf("Passed C Certification") : printf("Failed C Certification");
```

Since the condition marks > pass_marks evaluates to true, the **expression1** containing the **printf** statement in the above example prints "Passed C Certification"

Read the code given below and retype in the space provided.

```
#include <stdio.h>

int main()
{
   int marks = 75, pass_marks = 50;
   (marks > pass_marks) ? printf("Passed C exam.") : printf("Failed C exam.");
   return 0;
}
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int marks = 75,pass_marks = 50;
    (marks > pass_marks) ? printf("Passed C exam.") : printf("Failed C exam.");
    return 0;
}
```

	Expected	Got	
✓	Passed C exam.	Passed C exam.	✓

Passed all tests! ✓

Ternary Operator

In the program given below, fill in the missing code to find the **largest** of the two given numbers using **ternary operator**.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int num1 = 20, num2 = 25, large;
    large = (num1>num2) ? num1:num2; // Write the correct code
    printf("Largest number = %d", large);
    return 0;
}
```

	Expected	Got	
✓	Largest number = 25	Largest number = 25	✓

Cricket Stadium

There was a large ground in center of the city which is rectangular in shape. The Corporation decides to build a Cricket stadium in the area for school and college students, But the area was used as a car parking zone. In order to protect the land from using as an unauthorized parking zone, the corporation wanted to protect the stadium by building a fence. In order to help the workers to build a fence, they planned to place a thick rope around the ground. They wanted to buy only the exact length of the rope that is needed. They also wanted to cover the entire ground with a carpet during rainy season. They wanted to buy only the exact quantity of carpet that is needed. They requested your help. Can you please help them by writing a program to find the exact length of the rope and the exact quantity of carpet that is required?

Input format:

Input consists of 2 integers. The first integer corresponds to the length of the ground and the second integer corresponds to the breadth of the ground.

Output Format:

Output Consists of two integers. The first integer corresponds to the length. The second integer corresponds to the quantity of carpet required.

Sample Input:

50

20

Sample Output:

140

1000

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int len,wid;
    scanf ("%d %d",&len,&wid);
    printf("%d\n",2*(len + wid));
    printf("%d",len*wid);
}
```

	Input	Expected	Got	
✓	50 20	140 1000	140 1000	✓

Passed all tests! ✓

Sports Day Celebration

Training for sports day has begun and the physical education teacher has decided to conduct some team games. The teacher wants to split the students in higher secondary into equal sized teams. In some cases, there may be some students who are left out from the teams and he wanted to use the left out students to assist him in conducting the team games. For instance, if there are 50 students in a class and if the class has to be divided into 7 equal sized teams, 7 students will be there in each team and 1 student will be left out. That 1 student will assist the PET. With this idea in mind, the PET wants your help to automate this team splitting task. Can you please help him out?

INPUT FORMAT:

Input consists of 2 integers. The first integer corresponds to the number of students in the class and the second integer corresponds to the number of teams.

OUTPUT FORMAT:

The output consists of two integers. The first integer corresponds to the number of students in each team and the second integer corresponds to the students who are left out.

SAMPLE INPUT:

60

8

SAMPLE OUTPUT:

7

4

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int n,teams,leftout,perteam;
    scanf("%d %d",&n,&teams);
    perteam = n/teams;
    leftout= n%teams;
    printf("%d\n%d",perteam,leftout);
}
```

	Input	Expected	Got	
✓	60 8	7 4	7 4	✓

Passed all tests! ✓

The Newspaper Agency

Each Sunday, a newspaper agency sells w copies of a special edition newspaper for Rs.x per copy. The cost to the agency of each newspaper is Rs.y. The agency pays a fixed cost for storage, delivery and so on of Rs.100 per Sunday. The newspaper agency wants to calculate the profit which it obtains only on Sundays. Can you please help them out by writing a program to compute the profit if w, x, and y are given?

INPUT FORMAT:

Input consists of 3 integers: w, x, and y. w is the number of copies sold, x is the cost per copy and y is the cost the agency spends per copy.

OUTPUT FORMAT:

The output consists of a single integer which corresponds to the profit obtained by the newspaper agency.

SAMPLE INPUT:

1000

2

1

SAMPLE OUTPUT:

900

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int w,x,y,a;
    scanf("%d %d %d",&w,&x,&y);
    a=(w*x)-(w*y)-100;
    printf("%d",a);
}
```

	Input	Expected	Got	
✓	1000 2 1	900	900	✓

Passed all tests! ✓

The Chronicles Of Narnia

Four kids Peter, Susan, Edmond and Lucy travel through a wardrobe to the land of Narnia. Narnia is a fantasy world of magic with mythical beasts and talking animals.While exploring the land of narnia Lucy found Mr.Tumnus the two legged stag ,and she followed it, down a narrow path .She and Mr.Tumnus became friends and he offered a cup of coffee to Lucy in his small hut.It was time for Lucy to return to her family and so she bid good bye to Mr.Tumnus and while leaving Mr.Tumnus told that it is quite difficult to find the route back as it was already dark. He told her to see the trees while returning back and said that the first tree with two digits number will help her find the way and the way to go back to her home is the sum of digits of the tree and that numbered way will lead her to the tree next to the wardrobe where she can find the others. Lucy was already confused, so please help her in finding the route to her home....

Input Format:

Input consists of an integer corresponding to the 2-digit number.

Output Format:

Output consists of an integer corresponding to the sum of its digits.

SAMPLE INPUT:

87

SAMPLE OUTPUT:

15

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int n,d,ans,sum=0;
    scanf("%d",&n);
    d=n%10;
    sum=sum+d;
    n=n/10;
    ans=n+d;
    printf("%d",ans);
}
```

	Input	Expected	Got	
✓	87	15	15	✓
✓	54	9	9	✓

Passed all tests! ✓

Profit Calculator

Each Sunday, a newspaper agency sells X copies of a certain newspaper for Rs.A per copy. The cost to the agency of each newspaper is Rs.B . The agency pays a fixed cost for storage, delivery and so on of Rs.100 per Sunday. The newspaper agency wants to calculate the profit obtained on Sundays. Can you please help them out by writing a C program to compute the profit given X, A and B.

Input Format:

Input consists of 3 integers: X, A and B. X is the number of copies sold, A is the cost per copy and B is the cost the agency spends per copy.

Output Format:

Refer Sample Input and Output for exact formatting specifications.

Sample Input and Output:

Input

1000

2

1

Output

900

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int w,x,y,a;
    scanf("%d %d %d",&w,&x,&y);
    a=(w*x)-(w*y)-100;
    printf("%d",a);
}
```

	Input	Expected	Got	
✓	1000 2 1	900	900	✓

Passed all tests! ✓

Money With Baba

Baba is very kind to beggars and every day Baba donates half of the amount he has when ever a beggar requests him. The money M left in Baba's hand is passed as the input and the number of beggars B who received the alms are passed as the input. The program must print the money Baba had in the beginning of the day.

Input Format:

The first line denotes the value of M.
The second line denotes the value of B.

Output Format:

The first line denotes the value of money with Baba in the beginning of the day.

Example Input/Output:

Input:

100
2

Output:

400

Explanation:

Baba donated to two beggars. So when he encountered second beggar he had 100*2 = Rs.200 and when he encountered 1st he had 200*2 = Rs.400.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int M,B,amt;
    scanf("%d %d",&M,&B);
    amt=(M*B)*2;
    printf("%d\n",amt);

}
```

	Input	Expected	Got	
✓	100 2	400	400	✓

Passed all tests! ✓

Punctuality Incentive

The CEO of company ABC Inc wanted to encourage the employees coming on time to the office. So he announced that for every consecutive day an employee comes on time in a week (starting from Monday to Saturday), he will be awarded Rs.200 more than the previous day as "Punctuality Incentive". The incentive I for the starting day (ie on Monday) is passed as the input to the program. The number of days N an employee came on time consecutively starting from Monday is also passed as the input. The program must calculate and print the "Punctuality Incentive" P of the employee.

Input Format:

The first line denotes the value of I.
The second line denotes the value of N.

Output Format:

The first line denotes the value of P.

Example Input/Output:

Input:

500
3

Output:

2100

Explanation:

On Monday the employee receives Rs.500, on Tuesday Rs.700, on Wednesday Rs.900

```c
#include<stdio.h>
int main()
{
    int I,N,P;
    scanf("%d %d",&I,&N);
    P=((I+200)*N);
    printf("%d\n",P);
}
```

	Input	Expected	Got	
✓	500 3	2100	2100	✓
✓	100 3	900	900	✓

Passed all tests! ✓

Chocolates Remaining

Bajan Lal distributes C chocolates to school N students every Friday. The C chocolates are distributed among N students equally and the remaining chocolates R are given back to Bajan Lal.

As an example if C=100 and N=40, each student receives 2 chocolates and the balance 100-40*2 = 20 is given back.

If C=205 and N=20, then each student receives 10 chocolates and the balance 205-20*10 = 5 is given back.

Help the school to calculate the chocolates to be given back when C and N are passed as input.

Input Format:

The first line denotes C
The second line denotes N

Output Format:

The first line denotes R - the number of chocolates to be given back.

Example Input/Output:

Input:

300
45

Output:

30

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int C,N,given;
    scanf("%d %d",&C,&N);
    given=(C/N)*N;
    printf("%d",C-given);
}
```

	Input	Expected	Got	
✓	300 45	30	30	✓

Passed all tests! ✓

If Construct

The general format of if statement is

```
if (condition) {
   statement-1;
   statement-2;
   ....
   statement-n;
}
```

The if construct is a **selective statement**, the statements within the block are executed only once when the **condition evaluates to true**, otherwise the control goes to the first statement after the if construct.

If only one statement is presented in the if construct then there is no need to specify the braces { } i.e., if braces are not specified for the if construct, by default the next immediate statement is the only statement considered for the if construct.

Below code prints the number only when it is **divisible by 3**:

```
#include <stdio.h>
int main()
{
        int num;
        printf("Enter a number : ");
        scanf("%d", &num);
        if (num % 3 == 0)
        {

                printf("Given number %d is divisible by 3", num);
        }
        return 0;
}
```

In the above code, num % 3 == 0 is the **condition**, which verifies whether the **number is divisible by 3**. Only if the condition returns 1 (true) then the control enters in to the **if-block** and executes the statement.

Fill in the missing code in the below program to check whether the given number is divisible by **3** or not.

Answer: (penalty regime: 0 %)

Reset answer

```
#include <stdio.h>

int main()
{
    int num;
    scanf("%d", &num);
    if(num % 3 ==0)
    {
        printf("Given number %d is divisible by 3\n", num);
    }
    else
    {
        printf("Given number %d is not divisible by 3\n", num);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	9	Given number 9 is divisible by 3	Given number 9 is divisible by 3	✓
✓	7	Given number 7 is not divisible by 3	Given number 7 is not divisible by 3	✓

Passed all tests! ✓

If Else Construct

The if statement tells a program to execute a certain section of code only if a particular test evaluates to true. if (*expression*) {*statement*}.

Below is a sample code which uses a if statement:

```
int distinction_marks = 75;
if (marks > distinction_marks)
{
        printf("User secured distinction.\n");
}
```

An if statement will execute its block only when condition evaluates to 1 (**true**).

We can also conditionally execute another block when the condition evaluates to 0 (**false**) using the else construct. The else construct must be attached to an if, hence together they are referred to as if-else construct.

The if-else statement provides two different paths of execution depending on the result of the condition.

Below is the general syntax for the if-else statement :

```
if (expression)
{
        statement-1;
}
else
{
        statement-2;
}
```

Below is an example with code:

```
int distinction_marks = 75;
if (marks > distinction_marks)
{
        printf("User secured distinction.\n");
}
else
{
{
        printf("User did not secure distinction.\n");
}
```

Fill in the missing code in the below program to check whether the user secured distinction or not.

For example:

Input	Result
76	User secured distinction.
21	User did not secure distinction.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int marks, distinction_marks = 75;
    scanf("%d", &marks);
    if(marks> distinction_marks)
    { // Write the if condition
        printf("User secured distinction.\n");
    }
    else
    { // Write else part
        printf("User did not secure distinction.\n");
    }
}
```

	Input	Expected	Got	
✓	76	User secured distinction.	User secured distinction.	✓
✓	21	User did not secure distinction.	User did not secure distinction.	✓

Passed all tests! ✓

If Else Construct

Write code which uses an if-else statement to check whether a given account balance is greater or lesser than the minimum balance.

Use the if-else statement and print "Balance is low" if the balance is less than **1000**, otherwise print "Sufficient balance".

For example, if the user gives the **input** as 1500:

```
1500
```

then the program should **print** the result as:

```
Sufficient balance
```

Similarly, if the input is given as 700 then print

```
Balance is low
```

[**Hint:** Make sure to read the input as a float value.]

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int amt,suff_bal=1000;
    scanf("%d",&amt);
    if(amt>suff_bal)
     {
         printf("Sufficient balance");
     }
    else
     {
         printf("Balance is low");
     }
}
```

	Input	Expected	Got	
✓	1225	Sufficient balance	Sufficient balance	✓
✓	999.55	Balance is low	Balance is low	✓

Passed all tests! ✓

If Else Construct

Fill in the missing code in the below program to check whether the student secured first class or not.

Note-1: Read **6** subjects marks, find total and percentage, then print the student secured first class or not.

Note-2: If percentage is greater than or equal to **60** then print student secured first class and the percentage.

For example:

Input	Result
45 67 34 57 68 81	Student did not secure a first class with 58.67%
67 68 65 56 59 69	Student secured a first class with 64.00%

```c
#include <stdio.h>

int main()
{
    int maths,computers,physics,chemistry,english,spanish, total;
    float percentage;
    scanf("%d %d %d %d %d %d",&maths,&computers,&physics,&chemistry,&english,&spanish);
    total=maths+computers+physics+chemistry+english+spanish;
    percentage=(float)total/6;
    // Read marks

    // Calculate total and percentage

    if(percentage>=60)
    { // Write the condition
        printf("Student secured a first class with %5.2f%%\n", percentage);
    }
    else
    { // Write the else part
        printf("Student did not secure a first class with %5.2f%%\n", percentage);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	45 67 34 57 68 81	Student did not secure a first class with 58.67%	Student did not secure a first class with 58.67%	✓
✓	67 68 65 56 59 69	Student secured a first class with 64.00%	Student secured a first class with 64.00%	✓

Passed all tests! ✓

If Else Construct

Write a program which uses an if-else statement to verify and print if the given number is an odd or an even.

For example, if the user gives the **input** as 10:

```
10
```

then the program should **print** the result as:

```
The given number 10 is an even number
```

If the input is given as 35, then the program should print the result as :

```
The given number 35 is an odd number
```

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int number;
    scanf("%d",&number);
    // read the int value
    if(number%2==0)

    { // write if condition to check the given number is even or odd
     printf("The given number %d is an even number",number);// print even or odd

    }
    else
    {
        printf("The given number %d is an odd number",number);// print even or odd

    }
    return 0;
}
```

	Input	Expected	Got	
✓	35	The given number 35 is an odd number	The given number 35 is an odd number	✓
✓	10	The given number 10 is an even number	The given number 10 is an even number	✓

Passed all tests! ✓

If Else Construct

Write a program which uses an if-else statement to verify if the given character is an alphabet or not.

For example, if the user gives the **input** as W:

```
W
```

then the program should **print** the result as:

```
Given character W is an alphabet
```

If the input us given as 7, then print the result as:

```
Given character 7 is not an alphabet
```

[**Hint:** The ASCII values of alphabets '**A**' to '**Z**' are 65 to 90 and '**a**' to '**z**' are 97 to 122.]

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    char a;
    scanf("%c", &a);
    if((a>='A'&& a<='Z')||(a>='a'&& a<='z'))
     {
         printf ("Given character %c is an alphabet",a);
     }
    else
     {
         printf("Given character %c is not an alphabet",a);
     }
}
```

	Input	Expected	Got	
✓	W	Given character W is an alphabet	Given character W is an alphabet	✓
✓	7	Given character 7 is not an alphabet	Given character 7 is not an alphabet	✓

Passed all tests! ✓

Nested If Structure

When an if-else construct appear as a statement within another if-block or a else-block, it is referred to as nesting of if-else construct.

Below is an example of a **nested if-else** construct:

```
if (expression_1)
{
        if (expression_2)
        {
                if (expression_3)
                {
                        statement_1;
                }
                else
                {
                        statement_2;
                }
        }
        else
        {
                statement_3;
        }
}
```

In the above syntax, the **statement_2** will be executed only when the conditions in expression_1, expression_2 and expression_3 evaluates to 1 (true).

Fill in the missing code in the below program to find the **largest** of three numbers using nested if-else.

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int a, b, c;
    scanf("%d %d %d", &a, &b, &c);
    // Correct the below code
    if(a>b)
    {
        if(a>c)
        {
        printf("%d is greater than %d and %d\n",a,b,c);
        }
        else
        {
        printf("%d is greater than %d and %d\n",c,a,b);
        }
    }
    else
    {
        if(b>c)
        {
            printf("%d is greater than %d and %d\n",b,a,c);
        }
        {
            printf("%d is greater than %d and %d\n",c,a,b);
        }
    }
    return 0;
}
```

	Input	Expected	Got	
✓	23 56 77	77 is greater than 23 and 56	77 is greater than 23 and 56	✓

Passed all tests! ✓

If Else If Construct

The if-else-if construct extends the if-else construct by allowing to chain multiple if constructs as shown below:

```
if (expression_1)
{
        statement_1;
}
else if (expression_2)
{
        statement_2;
}
else if (expression_3)
{
        statement_3;
}
else if (expression_4)
{
        statement_4;
}
else
{
        statement_5;
}
```

As shown in the above syntax, multiple if constructs can be chained to any length. The else construct which appears at the end is optional, and if it is to be included it has to be only at the end.

The if-else-if construct is used whenever we have multiple mutually exclusive if conditions which work on the same input.

In a if-else-if construct the conditions are evaluated from top to bottom. Whenever a condition evaluates to **true** (1), the control enters into that if-block and after that the control comes out of the complete if-else-if construct ignoring all the remaining if and else constructs that may exist below the currently satisfied if-block.

For example, if the condition in the expression_2 is the first condition to evaluate to **true** after executing statement_2 the control comes out of the complete if-else-if construct.

The below program reads a character from the console and should print if the given character is an alphabet or a digit. Do not remove the existing code, add the missing lines of code which employs the if-else-if statement to produce appropriate output.

```c
#include <stdio.h>

int main()
{
    char ch;
    ch = getchar();
    //fill the appropriate if condition
    if((ch>='A'&& ch <= 'Z'))
      {
    printf("Given character %c is an alphabet\n", ch);
      }
    else if (ch >= '0' && ch <= '9')
      {
    //fill the appropriate else if condition
    printf("Given character %c is a digit\n", ch);
      }
    else
      {
    //fill the appropriate else condition
    printf("Given character %c is neither an alphabet nor a digit\n", ch);
      }
    return 0;
}
```

	Input	Expected	Got	
✓	A	Given character A is an alphabet	Given character A is an alphabet	✓
✓	8	Given character 8 is a digit	Given character 8 is a digit	✓
✓	%	Given character % is neither an alphabet nor a digit	Given character % is neither an alphabet nor a digit	✓

Passed all tests! ✓

If Else If Construct

The following code uses if-else statement to check whether the given integer number is a valid **leap year** or not.

Use if-else statement and print "__ is a leap year":

- if a year is divisible by **4** and should not be divisible by **100**.
- If a year is divisible by **400**.

Otherwise, print "__ is not a leap year".

Fill in the missing code in the below program to check whether the given year is a **leap year** or not..

For example:

Input	Result
1900	1900 is not a leap year

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int year;
    scanf("%d", &year);
    // Fill in the missing code
    if((year % 4 == 0 && year % 100 != 0) || (year % 400 ==0))
    {
        printf("%d is a leap year\n", year);
    }
    else
    {
        printf("%d is not a leap year\n", year);
    }

    return 0;
}
```

	Input	Expected	Got	
✓	1900	1900 is not a leap year	1900 is not a leap year	✓
✓	2000	2000 is a leap year	2000 is a leap year	✓

Passed all tests! ✓

If Else If Construct

Fill in the missing code in the below program to read an **integer value** for a variable age and use if-else statement to check the age and print appropriate ticket price.

If **age** is lessthan or equal to **infant_age** (3 years) or greaterthan or equal to **centenarian_age** (100 years) then print **Ticket Price: 0**.

Otherwise, If **age** is lessthan or equal to **child_age** (13 years) or greaterthan or equal to **senior_citizen_age** (60 years) then print **Ticket Price: 5**.

Otherwise, print **Ticket Price: 10**.

For example:

Input	Result
34	Ticket Price: 10
2	Ticket Price: 0
101	Ticket Price: 0
72	Ticket Price: 5

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int age, infant_age = 3, child_age = 13, senior_citizen_age = 60, centenarian_age = 100;
    scanf("%d", &age);
    if(age <= infant_age || age >= centenarian_age  )
    { // if condition
        printf("Ticket Price: 0\n");
    }
    else if(age <= child_age || age >= senior_citizen_age )
    { // else if condition
        printf("Ticket Price: 5\n");
    }
    else
    { // else
        printf("Ticket Price: 10\n");
    }
    return 0;
}
```

	Input	Expected	Got	
✓	34	Ticket Price: 10	Ticket Price: 10	✓
✓	2	Ticket Price: 0	Ticket Price: 0	✓
✓	101	Ticket Price: 0	Ticket Price: 0	✓
✓	72	Ticket Price: 5	Ticket Price: 5	✓

Switch Case Construct

A switch statement is used to change the control flow of a program execution through multiple paths depending on an expression's value.

The below code demonstrates how to use a switch-case construct to print the corresponding English words for the digits (**1** to **9**) read from the standard input.

One way is to write a long nested if-else-if for the **10** numbers or the other way is to use a switch-case statement.

See and retype the below code which demonstrates the usage of switch statement to print the English word of the given number between **1** to **9**.

```
#include <stdio.h>
int main()
{
   int value;
   scanf("%d", &value);
   switch (value)
   {
   case 1:
      printf("One");
      break;
   case 2:
      printf("Two");
      break;
      case 3:
      printf("Three");
      break;
      case 4:
      printf("Four");
         break;
      case 5:
      printf("Five");
      break;
      case 6:
      printf("Six");
      break;
      case 7:
      printf("Seven");
         break;
      case 8:
      printf("Eight");
         break;
      case 9:
         printf("Nine");
         break;
      case 10:
         printf("Ten");
         break;
      default:
         printf("Number %d is not in the range 1 to 10", value);
   }
   return 0;
}
```

For example:

Input	Result
2	Two
9	Nine
15	Number 15 is not in the range 1 to 10

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int  value;
    scanf("%d",&value);
    switch(value)
    {
        case 1:
         printf("One");
         break;
        case 2:
         printf("Two");
         break;
        case 3:
         printf("Three");
         break;
        case 4:
         printf("Four");
         break;
        case 5:
         printf("Five");
         break;
        case 6:
         printf("Six");
         break;
        case 7:
         printf("Seven");
         break;
        case 8:
         printf("Eight");
         break;
        case 9:
         printf("Nine");
         break;
        case 10:
         printf("Ten");
         break;
        default:
         printf("Number %d is not in the range 1 to 10",value);

    }
}
```

	Input	Expected	Got	
✓	2	Two	Two	✓
✓	9	Nine	Nine	✓
✓	15	Number 15 is not in the range 1 to 10	Number 15 is not in the range 1 to 10	✓

Passed all tests! ✓

Switch Case Construct

Assume that the weekdays are provided with the below numbers:

```
Sunday ⟹ 0
Monday ⟹ 1
Tuesday ⟹ 2
Wednesday ⟹ 3
Thursday ⟹ 4
Friday ⟹ 5
Saturday ⟹ 6
```

Write a program to read the **weekday number** from the standard input and print the **weekday name** using switch-case.

For example, if the user gives the **input** as 1:

```
1
```

then the program should **print** the result as:

```
Monday
```

Note: If the given input number is not in the range i.e., other than **0** to **6**, the output should be as given below:

```
Invalid weekday number
```

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int day;
    scanf("%d",&day);
    switch(day)
     {
    case 0:
     printf("Sunday");
     break;
    case 1:
     printf("Mnday");
     break;
    case 2:
     printf("Tuesday");
     break;
    case 3:
     printf("Wednesday");
     break;
    case 4:
     printf("Thursday");
     break;
    case 5:
     printf("Friday");
     break;
    case 6:
     printf("Saturday");
     break;
    default:
     printf("Invalid weekday number");
     }
    return 0;
}
```

	Input	Expected	Got	
✓	6	Saturday	Saturday	✓
✓	0	Sunday	Sunday	✓
✓	7	Invalid weekday number	Invalid weekday number	✓

Passed all tests! ✓

While Loop

Most of the programming languages provide a special construct/statement using which we can repeatedly execute one or more statement as long as a condition is **true**. In C, we have while, do-while and for as the three main looping constructs or statements.

Below is a general syntax for using a while statement:

```
while (condition)
{
        statement_1;
        statement_2;
        ....
}
```

The block of code inside the opening and closing brace which follows the while-statement is called the **while-loop** body.

A while statement is used to execute some code repeatedly as long as a condition evaluates to true.

The condition is an expression which should always evaluate to either true or false.

- If it evaluates to true, the body containing one or more code statements is executed.
- If the expression evaluates to false, the control skips executing the **while-loop** body.

The while-loop construct is also referred to as an entry controlled loop. Meaning, first the condition is evaluated and only if the condition evaluates to true the body of the loop is executed. After executing the body the control is automatically transferred back to the condition and the process continues until the condition evaluates to false.

See and retype the below code which uses a while-loop to read multiple numbers from standard input and prints their sum when the **sum** exceeds 100.

```
#include <stdio.h>

int main()
{
   int total = 0;
   while (total <= 100)
   {
   int num;
   scanf("%d", &num);
      total += num;
```

```
    }
    printf("The total of given numbers is : %d", total);
    return 0;
}
```

For example:

Input	Result
34 62 24	The total of given numbers is : 120

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int total = 0;
    while (total <= 100)
    {
    int num;
    scanf("%d",&num);
     total += num;
    }
    printf("The total of given numbers is : %d",total);
    return 0;

}
```

	Input	Expected	Got	
✓	34 62 24	The total of given numbers is : 120	The total of given numbers is : 120	✓

Passed all tests! ✓

While Loop

The below sample code should print Ganga by number of times, where as the input is read by the programmer using **scanf()**.

Fill in the missing code so that it produces the desired output.

For example:

Input	Result
3	Ganga Ganga Ganga

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>
int main()
{
    int i = 0, n;
    scanf("%d",&n); //Fill the missing code in scanf() function
    while (i < n)
    { // complete the condition here
        printf("Ganga\n"); // Write the text to be printed here
        i++ ; // Complete the statement
    }
    return 0;
}
```

	Input	Expected	Got	
✓	3	Ganga Ganga Ganga	Ganga Ganga Ganga	✓

Passed all tests! ✓

While Loop

Write a **C** program to print first n **natural numbers**.

For example, if the user gives the **input** as :

```
3
```

then the program should **print** the result as:

```
The natural numbers from 1 - 3 : 1 2 3
```

For example:

Input	Result
3	The natural numbers from 1 - 3 : 1 2 3
9	The natural numbers from 1 - 9 : 1 2 3 4 5 6 7 8 9

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int n,i=1;
    scanf("%d",&n);
    printf("The natural numbers from 1 - %d : ",n);
    while(i<=n)
    {
        printf("%d ",i);
        i++;
    }

    return 0;
}
```

	Input	Expected	Got	
✓	3	The natural numbers from 1 - 3 : 1 2 3	The natural numbers from 1 - 3 : 1 2 3	✓
✓	9	The natural numbers from 1 - 9 : 1 2 3 4 5 6 7 8 9	The natural numbers from 1 - 9 : 1 2 3 4 5 6 7 8 9	✓

Passed all tests! ✓

While Loop

The below sample code should find the sum of **even numbers** between any two numbers.

[**Hint:** The numbers should be read by using scanf()].

Fill in the missing code so that it produces the desired output.

For example:

Input	Result
3 6	The sum of even integers between the given limits = 10

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int num1, num2, sum = 0;
    scanf("%d %d",&num1,&num2); // Fill the missing code in the scanf()
    if (num1 % 2 != 0)
    { // If it is an odd number then add 1
        num1 = num1 + 1;
    }
    while (num1 <= num2  )
    { // Write the condition part
        sum = sum + num1;
        num1 = num1+2 ;
    }
    printf("The sum of even integers between the given limits = %d\n",sum  );
    return 0;
}
```

	Input	Expected	Got	
✓	3 6	The sum of even integers between the given limits = 10	The sum of even integers between the given limits = 10	✓

Passed all tests! ✓

While Loop

Fill in the missing code in the below program to read an **integer number** and find the reverse of the given number.

For example if the input is 1234, then the output will be 4321.

Hints

The logic of reversing of any number is pretty simple if you know how to find last digit of any number. Initially the variable reverse contains zero(0), the process of reversing involves four basic steps:

- Multiply the reverse variable by 10.
- Find the last digit of the given number by applying % 10.
- Add the last digit just found to reverse.
- Divide the original number by 10 to eliminate the last digit, which is not needed anymore.

Repeat the above four steps till the original number becomes 0 and finally we will be left with the reversed number in reverse variable.

For example:

Input	Result
1234	The reverse number of a given number = 4321
765	The reverse number of a given number = 567

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>

int main()
{
    int n, digit, reverse = 0;
    scanf("%d", &n);
    while ( n!=0 )
    { // Write the condition
    digit = n%10 ; // Fill the correct code
    reverse = reverse*10+digit ; // Fill the correct code
    n = n/10 ; // Fill the correct code
    }
    printf("The reverse number of a given number = %d" , reverse);
    return 0;
}
```

	Input	Expected	Got	
✓	1234	The reverse number of a given number = 4321	The reverse number of a given number = 4321	✓
✓	765	The reverse number of a given number = 567	The reverse number of a given number = 567	✓

Passed all tests! ✓

While Loop

Fill in the missing code in the below sample program which finds the factorial of a given number.

Factorial of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n.
For example, 5! = 5 * 4 * 3 * 2 * 1 = 120.

The below sample code computes the factorial of a given non-zero integer.

The main() function declares an integer variable factorial and initializes it to 1, which it will use to store the computed factorial value.

It uses a **while-loop** to iterate from 2 to n multiplying the loop counter in each iteration with the factorial and storing the product again in factorial.

```c
#include <stdio.h>

int main()
{
    int i, n, factorial = 1;
    scanf("%d", &n);
    i = 2;
    while ( i<=n)
    { // Write the condition
    factorial =  factorial*i; // Fill the correct code
    i++;
    }
    printf("Factorial of given number %d = %d\n", n, factorial);
    return 0;
}
```

	Input	Expected	Got	
✓	2	Factorial of given number 2 = 2	Factorial of given number 2 = 2	✓
✓	4	Factorial of given number 4 = 24	Factorial of given number 4 = 24	✓

Passed all tests! ✓

While Loop

Below partial code is to verify if the given number is a prime number or not.

A prime number is a positive integer greater than 1, which is not divisible by any other number other than 1 and itself. Examples of a few prime numbers are 2, 3, 5, 7, 11, 13, 17, 19, etc.

Fill in the missing code so that it produces the desired output.

For example:

Input	Result
7	The given number 7 is a prime number
119	The given number 119 is not a prime number

```c
#include <stdio.h>

int main()
{
    int n, i = 1, count = 0; // initialize i and count with appropriate values
    scanf("%d", &n);
    while (i<=n )
    { // complete the condition to iterate the loop
    if ( n%i==0)
        { // complete the condition to check the remainder is 0 or not
        count++;
    }
    i++;
    }
    if (count == 2  )
    { // complete the condition to check the count
        printf("The given number %d is a prime number\n", n);
    }
    else
    {
    printf("The given number %d is not a prime number\n", n);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	7	The given number 7 is a prime number	The given number 7 is a prime number	✓
✓	119	The given number 119 is not a prime number	The given number 119 is not a prime number	✓

Passed all tests! ✓

While Loop

Below partial code is to verify if the given number is an armstrong number or not.

An armstrong number is a number that is the sum of its own digits raised to the power of number of digits that make up the original number.

For example, if the given number is 153, the total number of digits are 3, and the sum of cubes of each digit ($1^3 + 5^3 + 3^3$) is equal to the same number 153. Such a number is known as an armstrong number.

Let us take another example, if the given number is 9474, the total number of digits are 4, and the sum of the power of 4 of each digit ($9^4 + 4^4 + 7^4 + 4^4$) is equal to the same number 9474. Such a number is known as an armstrong number.

Similarly,
$9 = 9^1 = 9$
$371 = 3^3 + 7^3 + 1^3 = 27 + 343 + 1 = 371$
$3\ 8208 = 8^4 + 2^4 + 0^4 + 8^4 = 4096 + 16 + 0 + 4096 = 8208$

Fill in the missing code so that it produces the desired output.

```c
#include <stdio.h>
#include<math.h>
int main()
{
    int number, temp, remainder, i, power, digits = 0, sum = 0;
    scanf("%d", &number);
    temp = number;
    while ( temp !=0 )
    { // complete the condition to iterate the loop
    digits ++; // increment the digits
    temp = temp/10; //calculate the temp value
    }
    temp = number;
    while (temp!=0 )
    { // complete the condition to iterate the loop
    remainder = temp % 10;
    i = 1;
    power = 1;
    while(i<=digits )
        { // find the powers of each digit
        power *=remainder ; // calculate power value
            i++ ; // increment i value
        }
        sum +=power ; // calculate sum value
        temp =temp/10 ; // calculate number value
    }
    if (sum==number )
    { // write the condition
        printf("The given number %d is an armstrong number\n", number);
    }
    else
    {
    printf("The given number %d is not an armstrong number\n", number);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	777	The given number 777 is not an armstrong number	The given number 777 is not an armstrong number	✓
✓	9	The given number 9 is an armstrong number	The given number 9 is an armstrong number	✓

Passed all tests! ✓

For Loop

Fill in the missing code in the below program to calculate the value of a^n, given two positive non-zero integers a and n.

The code in the main() function reads two integers from standard input and stores them in the variables a and n.

It uses a for-loop to multiply a with itself n number of times.

Variable a_power_n is used to store the computed value of a^n.

After the execution of for-loop is completed, the final value of a_power_n is printed to the standard output.

```c
#include <stdio.h>
int main()
{
    int i, a, n, a_power_n;
    scanf("%d %d", &a, &n);
    a_power_n=1;
    for (  i=0; i<n ;i++  )
    { // Write the initialization, condition and increment part
        a_power_n *= a   ; // Calculate the value
    }
    printf("%d\n", a_power_n);
    return 0;
}
```

	Input	Expected	Got	
✓	2 3	8	8	✓

Passed all tests! ✓

For Loop

Write a program to find **sum** and **mean** of **n** numbers.

Constraints:

- 1 <= n <= 10^6
- 10^{-3} <= elements <= 10^3
- Result of mean should print upto **2 decimal places**.

Sample test case:

4----------> First line of input is the value on n.

3 5 7 8----------> Second line of input is n space separated integer values.

Sum: 23---------->Third line prints the Sum as required.

Mean: 5.75---------->Fourth line prints the Mean as required.

Instruction: To run your custom test cases strictly map your input and output layout with the visible test cases.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    float sum=0,c,n;
    float avg;
    scanf("%f",&n);
    for(c=0;c<n;c++)
    {
        float num;
        scanf("%f",&num);
        sum+=num;
    }
    printf("Sum: %.0f\n",sum);
    avg=sum/c;
    printf("Mean: %.2f",avg);
}
```

	Input	Expected	Got	
✓	4 3 5 7 8	Sum: 23 Mean: 5.75	Sum: 23 Mean: 5.75	✓

Passed all tests! ✓

For Loop

Fill in the missing code in the below program to print the Fibonacci series i.e., 0 1 1 2 3 5 8 13 21....., up to the limit.

The code in the main() function reads one integer variable n. It uses a for loop to iterate from 0 to n and print the series.

By definition, the first two numbers in the Fibonacci sequence are 0 and 1, and each subsequent number is the sum of the previous two.

For example:

Input	Result
25	The Fibonacci series is : 0 1 1 2 3 5 8 13 21

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>
int main()
{
    int fib1 = 0, fib2 = 1, fib3, n;
    scanf("%d", &n);
    printf("The Fibonacci series is : %d %d", fib1, fib2);
    for (fib3=1 ; fib3<n  ; fib3=fib1+fib2  )
    { // Write the initialization, condition and increment part
        printf(" %d", fib3);
        fib1 =fib2 ; // Assign a value
        fib2 = fib3 ; // Assign a value
    }
    return 0;
}
```

	Input	Expected	Got	
✓	25	The Fibonacci series is : 0 1 1 2 3 5 8 13 21	The Fibonacci series is : 0 1 1 2 3 5 8 13 21	✓

Passed all tests! ✓

For Loop

Write a program that will print all the **English alphabets** from A to Z, each in a new line.

Hints

1. The code in the main() function can use a for loop to iterate over the characters 'A' to 'Z'.
2. Note that char data type is a numeric type and can be used in a for loop as a loop counter.
3. You can declare and initialize a loop counter char i and initialize it to 'A' (eg: char i = 'A';). The condition can similarly be i <= 'Z'; and the update statement can be i++.
4. You can then print i directly which is of type char, using the **printf()** function with a newline character (\n).

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    char c='A';
    int i;
    for(i=0;i<=25;i++)
    {
        printf("%c\n",c+i);
    }
}
```

	Expected	Got	
✓	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	A B C D E F G H I J K L M N O P Q R S T U V W X Y Z	✓

Passed all tests! ✓

For Loop

Write a program to read **n** numbers from the user and then count number of "**Odd**" and "**Even**" numbers.

Constraints:

- $1 <= n <= 10^6$
- $10^{-3} <= \text{elements} <= 10^3$

Sample test case:

3----------> First line of input is n i.e. 3.

5 6 7-------> Second line of input is n space separated integer values/elements.

Even: 1-----> Third line prints the output (the count of even elements).

Odd: 2------->Fourth line prints the output (the count of odd elements).

Note: Do use the **printf()** function with a **newline** character (**\n**) to print your results on newline.

Instruction: To run your custom test cases strictly map your input and output layout with the visible test cases.

Answer: (penalty regime: 0 %)

```c
#include<stdio.h>
int main()
{
    int t,evec=0,oddc=0;
    scanf("%d",&t);
    for(int i=0;i<t;i++)
    {
        int num;
        scanf("%d",&num);
        if(num%2==0)
        {
            evec++;
        }
        else
        {
            oddc++;
        }
    }
    printf("Even: %d\nOdd: %d\n",evec,oddc);
}
```

	Input	Expected	Got	
✓	3 5 6 7	Even: 1 Odd: 2	Even: 1 Odd: 2	✓

Passed all tests! ✓

For Loop

Fill in the missing code in the below program to verify whether the given number is perfect, abundant or deficient.

A number is said to be perfect if it equals the sum of its proper divisors. For example, **6** and **28** can be called **perfect numbers** as : **6 = 1 + 2 + 3** and **28 = 1 + 2 + 4 + 7 + 14**.

Alternatively, if the sum of a number's proper divisors **exceeds** the number itself, it is said to be abundant, while if the sum of a number's proper divisors is **less-than** the number itself, it is said to be deficient.

For example:

Input	Result
6	The given number 6 is a perfect number
10	The given number 10 is a deficient number
12	The given number 12 is an abundant number

```c
#include <stdio.h>
int main()
{
    int n, i, sum = 0;
    scanf("%d", &n);
    for (i=1 ; i<n ; i++ )
    { //Write the initialization, condition and increment part
        if (n%i==0 )
        { // Fill the condition
            sum = sum + i;
        }
        else
        {
            continue;
        }
    }
    if (sum==n )
    { // Fill the condition
        printf("The given number %d is a perfect number", n);
    }
    else if (sum<n )
    { // Fill the condition
        printf("The given number %d is a deficient number", n);
    }
    else
    {
        printf("The given number %d is an abundant number", n);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	6	The given number 6 is a perfect number	The given number 6 is a perfect number	✓
✓	10	The given number 10 is a deficient number	The given number 10 is a deficient number	✓
✓	12	The given number 12 is an abundant number	The given number 12 is an abundant number	✓

Passed all tests! ✓

For Loop

Fill in the missing code in the below program to check whether the given number is a strong number or not.

A number is called strong number if sum of the **factorials** of its digit is equal to number itself. For example: **145** is considered a strong number since **1! + 4! + 5! = 1 + 24 + 120 = 145**.

The code in the below main() function reads a number from standard input and performs the verification for a strong number by extracting the individual digits and calculating their factorials.

For example:

Input	Result
145	The given number 145 is a strong number
123	The given number 123 is not a strong number

Answer: (penalty regime: 0 %)

Reset answer

```c
#include <stdio.h>
int main()
{
    int rem, n, i, sum = 0, temp, fact = 1;
    scanf("%d", &n);
    temp=n;
    for (temp = n;temp>0; temp = temp / 10)
    { // Write the condition part
        rem =temp % 10  ; // Calculate remainder value
        fact = 1;
        for (i =1; i<=rem  ; i++  )
        { // Write the initialization, condition and increment part
            fact = fact * i;
        }
        sum = sum + fact;
    }
    if ( sum==n )
    { // Fill the condition
        printf("The given number %d is a strong number\n", n);
    }
    else
    {
        printf("The given number %d is not a strong number\n", n);
    }
    return 0;
}
```

	Input	Expected	Got	
✓	145	The given number 145 is a strong number	The given number 145 is a strong number	✓
✓	123	The given number 123 is not a strong number	The given number 123 is not a strong number	✓

Passed all tests! ✓