Sanjula K A D L

24978

TUTORIAL 01

01.Briefly explain the need of a programming language?

Programming languages are used to write instructions for computers to follow. A language is essential for humans to work with computers. They are necessary because they allow humans to communicate with machines and tell them what to do.

02. Compare and contrast the differences between followings;

(a) Source code vs. Machine code

Source code is the human-readable version of a program, written in a high-level programming language like Python or Java. It can be compiled or interpreted into machine code, which is the binary code that computers can understand.

Machine code is the lowest level programming language that computers understand and is difficult for humans to read and write. Source code is easier for humans to read and write, but machine code is the only language that computers can run.

(b) High level language vs. Low level language

A high-level language is a language that is designed to be easy for humans to read and write. Examples of high-level languages include Python, Java, and C++. They are more abstract and less tied to the hardware of a specific computer.

Low-level languages are closer to the hardware and are more difficult for humans to read and write. Examples of low-level languages include assembly language and machine code. They are more tied to the specific hardware of a computer and are less portable than high-level languages.

(c) Compliers vs. Interpreters

Compilers and interpreters are used to translate high-level programming languages into machine code that computers can understand. A compiler translates the entire program into machine code before it is executed, while an interpreter translates the program line by line as it is executed. This means that compiled programs tend to run faster than interpreted programs, but interpreted programs can be more flexible and easier to debug.

(d) Structured language vs. Object oriented language

Structured programming languages are designed around the concept of procedures or functions that perform specific tasks. Examples of structured programming languages include C and Pascal.

Object-oriented programming languages are designed around the concept of objects that encapsulate data and behavior. Examples of object-oriented programming languages include Java, Python, and C++. Object-oriented programming allows for more modular and flexible code, as well as easier code reuse.

(e) C vs. C++

C and C++ are both programming languages that are widely used for system-level programming and application development. C is a procedural programming language known for its efficiency and low-level control over hardware.

C++ is an object-oriented programming language that is based on C but includes additional features such as classes, inheritance, and polymorphism. C++ is often used for large-scale application development and game development.

(f) C++ vs. Java

C++ and Java are both popular programming languages that are used for application development. C++ is a compiled language known for its efficiency and low-level control over hardware. It is often used for large-scale application development and game development. Java, on the other hand, is an interpreted language that is designed to be more portable and easier to write than C++. It is often used for web and mobile application development.

(g) Syntax error vs. logical error

Syntax errors occur when code breaks the rules of a programming language. This can include things like missing semicolons, incorrect parentheses, or misspelled keywords. Boolean errors occur when code runs without any syntax errors but does not produce the expected output. This can happen when the code contains an error in the logic or algorithm used to solve the problem. Debugging logical errors can be more difficult than debugging syntax errors because they may require careful examination of the code to identify the problem.

TUTORIAL 02

01. How do you write comments in a C program ? what is the purpose of comments in a program?

In C, comments can be added to code using double slashes (//) for single-line comments or /* and */ for multi-line comments. Comments are used to explain the purpose or function of code to make it easier to understand and maintain. They can also be used to temporarily disable a piece of code or to provide additional information to other developers who may be working on the same project.

02. Which is the function that is essential in a C program ?

- Main Function

In a C program, the main function is the main function. This is the entry point of the program where execution begins. The main function may contain other functions that perform specific tasks, but it is the first function that is executed when the program starts. The main function must return an integer value that indicates the state of the program after execution.

03. What is the purpose of " scanf " ?

The purpose of the "scanf" function in C is to read input data from the standard input stream (usually the keyboard) and store it in variables. The "scanf" function reads formatted input, meaning it expects the input to be in a specific format specified by the programmer using format specifiers. Format specifiers are used to indicate the type of data to be read, such as integers, floating-point numbers, or characters. The "scanf" function can be used to read input from the user or from a file, and is often used in conjunction with the "printf" function to create interactive programs.

04. Is "standard C" a case sensitive language ?

- Yes

"Standard C" is a case sensitive language. This means that variables, functions and keywords must be written in the correct case for the compiler to recognize them. For example, the variable "count" is not the same as "Count" or "COUNT". It is important to be consistent with the capitalization of identifiers in a C program to avoid errors and make the code more readable.

05. Determine which of the following are valid identifiers. If invalid, explain why.

(a) record1	**(e) $tax**	**(h) name-and-address**
(b) 1record	**(f) name**	**(i) name_and_address**
(c) file-3	**(g) name and address**	**(j) 123-45-6789**
(d) return		

Valid :-

record1
$tax
name_and_address

Invalid :-

1record
file-3
Return
name and address
name-and-address
123-45-6789

- **1record :-**

  "1record" is an invalid identifier in C because it starts with a number. Identifiers must begin with a letter or an underscore.

- **file-3 :-**

  "file-3" is an invalid identifier in C because it contains a dash. Identifiers can only contain letters, numbers and underscores.

- **Return :-**

  "Return" is an invalid identifier in C because it is a keyword used to specify the value to be returned by the function. Identifiers cannot be the same as keywords in C.

- **name and address :-**

  In C, "name" and "address" can be valid identifiers because they consist of alphanumeric characters and do not start with a number.

- **name-and-address :-**

  "name-an-address" is an invalid identifier in C because it contains two dashes. Identifiers can only contain letters, numbers and underscores.

- **123-45-6789 :-**

  "123-45-6789" is an invalid identifier in C because it starts with a number and contains dashes. Identifiers must begin with a letter or an underscore and must not contain dashes.

06. State whether each of the following is true and false. If false , explain why.

(a) Function printf always begins printing at the beginning of a new line.

- False.

  By default, the `printf` function does not start printing at the beginning of a new line. Continues printing on the same line as the previous output unless a newline character is explicitly included in the output format string.

(b) Comments cause the computer to print the text enclosed between /* and */ on the screen when the program is executed.

- False.

  Comments in C are ignored by the compiler and do not affect the output of the program. Comments are used to provide information about the code to other programmers and to make the code more readable. They are not performed by a computer.

(c) The escape sequence \n when used in a printf format control string causes the cursor to position to the beginning of the next line on the screen.

- True.

  The escape sequence `\n` is used to represent a newline character in the control string of the printf format. When `printf` encounters a `\n` character in the format string, it causes the cursor to move to the beginning of the next line on the screen.

(d) All variables must be defined before they're used.

- True.

In C, variables must be declared before they are used. This means that the name and type of the variable must be specified before the program attempts to use the variable. The declaration can be made either at the beginning of the function or outside the function, at the top of the file**.**

(f) All variables must be given a type when they're defined.

- True.

In C, all variables must have a type when defined. The type determines the kind of data that the variable can hold and determines the amount of memory that will be allocated for the variable. If the type is not specified, the compiler will generate an error.

(e) C considered the variables, numbers and NuMbEr to be identical.

- False.

In C, variable names are case sensitive. This means that `NuMbEr` and `number` are two different variable names and will be treated as such by the compiler.

(g) A program that prints three lines of outputs must contain three printf statements.

- False.

A program that prints three lines of output can be created with a single printf statement with three newline characters (`\n`) or with multiple printf statements. It is up to the programmer which approach to use based on the requirements of the program.

07. What does the following code print ?

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

```
*

**

***

****

*****
```

08. Identify and correct the errors in each of the following statements. (Note : There may be more than one error per statement)

(a) scanf(“d”, value); :-

The error in the command is that the format specifier in the scanf command is incorrect. The format specifier for an integer value is "%d", so the correct command would be:

scanf("%d", &value);

Also, the "value" variable should be preceded by an ampersand character (&), which indicates the memory address of the variable where the input value will be stored.

(b)printf(“The product of %d and %d is %d”\n, x ,y);

The error in the command is that the newline character is outside the quotes, so it is not interpreted as a newline character. The correct statement would be:

printf("The product of %d and %d is %d\n", x, y, x*y);

Here the newline character is inside the quotes and will be interpreted as a newline character when printed

(c)scanf(“%d”, anInteger);

The error in the command is that the second argument of the scanf function, which is the variable that will store the input integer, is not preceded by the ampersand character (&), which indicates the memory address of the variable. The correct statement would be:

scanf("%d", &integer);

Here, the "%d" format specifier is correct for reading an integer value, and the "anInteger" variable is preceded by an ampersand (&), which indicates the memory address of the variable where the input value will be stored.

(d)printf(“Remainder is %d divided by %d is\n”, x, y, x % y);

The error in the command is that the format specifier for outputting the result of the modulo operation is missing. The correct statement would be:

printf("Reminder %d divided by %d is %d\n", x, y, x % y);

Here the format specifier for the result of the modulo operation is "%d" and the newline character is inside the quotes and will be interpreted as a newline character when printed.

(e)printf(“The sum is %d\n,” x + y);

The error in the statement is that the second argument of the printf function, which is the sum of x and y, is not enclosed in parentheses, so it does not evaluate to an expression. The correct statement would be:

printf("Sum is %d\n", x + y);

Here, the sum of x and y is enclosed in parentheses and will be evaluated as an expression when printed. The newline character is inside the quotation marks and will be interpreted as a newline character when printed.

(f)printf(“The value you entered is: %d\n, &value);

The error in the command is that the ampersand (&) is not required in the second argument of the printf function, which is the memory address of the variable where the input value is stored. The correct statement would be:

printf("Entered value is %d\n", value);

Here, the variable "value" contains the input value entered by the user and is directly passed as an argument to the printf function. The newline character is inside the quotation marks and will be interpreted as a newline character when printed.

09. What,if anything, prints when each of the following statement is performed ? If nothing prints , then answer "Nothing". Assume x=2 and y=3.

a) printf("%d", x);	2
b) printf("%d", x + x);	4
c) printf("x =");	x=
d) printf("x=%d", x);	x=2
e) printf("%d = %d", x + y, y +x);	5=5
f) z = x + y;	nothing
g) scanf("%d%d", &x , &y);	nothing
h) /*printf("x+ y= %d", x + y);	*/x+ y= 5
i) printf("\n");	it will print a line break

10. State which of the following are true and which are false. If false explain your answer

a) **C operators are evaluated from left to right.**

This statement is generally true.
Operators in C are evaluated from left to right, except for a few operators such as the assignment operator (=), the conditional operator (?:), and the comma operator (,). These operators are evaluated from right to left.

b) **The following are all valid variable names: _under_bar_, m928134 ,t5 ,j7 , her_sales , his _account_total , a, b, c, z, z2.**

True

c) **The statement printf(" a=5 ;"); is a typical example of an assignment statement.**

False.

The statement 'printf(" a=5 ;");' Not a typical example of an assignment statement. This is a print statement where the string "a=5;" prints the To the console, an assignment command usually assigns a value to a variable, such as "a = 5;" assigns

d) A valid arithmetic expression containing no parentheses is evaluated from left to right.

False.

Arithmetic expressions without parentheses are evaluated in order of operation. The order of operations is the set of rules used to determine the order in which operations are performed. In C, the order of operations is as follows (highest to lowest priority):

1. Unary operators (such as ++ and --)
2. Multiplication, division, remainder
3. Addition and subtraction

At each level of precedence, operators are evaluated from left to right.

e) The following are all invalid variable names: 3g, 87 ,67h2 , h22 , 2h

False.

The variable names '3g', '87', '67h2', and '2h' are all invalid in C. Variable names in C cannot start with a number.

TUTORIAL 03

01.Write four different C statements that each add 1 to integer variable x.

Statement 1	int x=0; x=x+1;
Statement 2	int x=0; x++;
Statement 3	int x=0; x+=1;
Statement 4	int x; x=0+1;

02.Write a single C statement to accomplish each of the following:

a) **Assign the sum of x and y to z and increment the value of x by 1 after the calculation.**
z=x+ y; x++;

b) **Multiply the variable product by 2 using the *= operator.**
product*=2;

c) **Multiply the variable product by 2 using the = and * operators**
product=product*2

d) **Test if the value of the variable count is greater than 10. I fit is, print "Count is greater than 10."**
if(count>10) {
printf("Count is greater than 10");}

e) **Decrement the variable x by 1, then subtract it from the variable total.**
x++;y=total-x;

f) **Add the variable x to the variable total, then decrement x by 1.**
y=x+ total; x--;

g) **Calculate the remainder after q is divided by divisor and assign the result to q. Write this statement two different ways.**

Divisor	d
First way	q=q%d;
Second way	p=q%d; q=p;

h) **Print the value 123.4567 with 2 digits of precision. What value is printed.**
123.45

i) **Print the floating-point value 3.14159 with three digits to the right of the decimal point. What value is printed.**

3.141

03. Write single C statement that

a) **Input integer variable x with scanf.**
   Scanf(“%d”,&x);
b) **Input integer variable y with scanf.**
   Scanf(“%d”,&y);
c) **Initialize integer variable I to 1.**
   int i = 1;
d) **Initialize integer variable power to 1.**
   int power =1;
e) **Multiply variable power by x and assign the result to power.**
   power= power*x;
f) **Increment variable I by 1.**
   i =i+1
g) **Test i to see if it’s less than or equal to y in the condition of a while statement.**
   while(i<=y)
h) **Output integer variable power with printf.**
   printf(“%d”, &power);

TUTORIAL 04

01. What is wrong with the following if statement (there are at least 3 errors). The indentation indicates the desired behavior.

```
if numNeighbors >=3 || numNeighbors = 4
++numNeigbhors;
printf("you are dead ! \n");
else
--numNeighbors;
```

Errors;-

There's no () sign used to indicate the condition of the if statement.

The statements inside if and else isn't indented.

The condition 2 is already inside the condition 1 in above statement.

02. Describe the output produced by this poorly indented program segment:

```
Int number = 4;
double alpha = -1.0;
if (number > 0)
if (alpha > 0 )
printf("Here I am! \n");
else
printf("No, I'm here \n");
printf("No, actually, I'm here!\n");
```

output;-

```
No, I'm here!
No, actually, I'm here!
```

03. Consider the following if statement, where doesSignificantWork, makesBreakthrough, and nobelPrizeCandidate are all Boolean variables:

```
if (doesSignificantWork) {
if (makeBreakthrough)
nobelPrizeCandidate = true;
else
nobelPrizeCandidate = false;
}
else if (!doesSignificantWork)
nobelPrizeCandidate = false;
```

- If ‘ doesSignificantWork ’ is true and ‘ makeBreakthrough ’ is true, ‘ nobelPrizeCandidate ’ is also true.
- If ‘ doesSignificantWork ’ is true and ‘ makeBreakthrough ‘ is false, ‘nobelPrizeCandidate ‘ is become false.
- If ‘ doesSignificantWork ‘ is false, ‘ nobelPrizeCandidate ‘ is always become false.

04. Write if statements to do the following:

- If character variable taxCode is ‘T’, increase price by adding the taxRate percentage of price to it.

```
If(taxCode=’T’)
{Price =price+ taxRate;}
```

- If integer variable opCode has the value 1, read in double values for x and y and calculate and print their sum.

```
If(opCode=1) {
Printf(“The sum is %f”, %x+ y);}
```

- If integer variable currentNumber is odd, change its value so that it is now 3 times currentNumber plus 1, otherwise change its value so that it is now half of currentNumber (rounded down when currentNumber is odd).

```
if(currentNumber%2 =1) {currentNumber=3*currentNumber +1;}
else {currentNumber=currentNumber/2;}
```

- Assign true to the boolean variable leapYear if the integer variable year is a leap year.(A leap year is a multiple of 4, and if it is a multiple of 100, it must also be a multiple of 400.)

```
if(year%4=0)
{leapYear=true;}
```

- Assign a value to double variable cost depending on the value of integer variable distance as follows:

Distance	**cost**
O through 100	**5.00**
More than 100 but not more than 500	**8.00**
More than 500 but less than 1,000	**10.00**
1,000 or more	**12.00**

```
if(distance<=100) {cost=5.00;}
else if(distance<=500) {cost=8.00;}
else if(distance<1000) {cost=10.00;}
else{cost=12.00;}
```

TUTORIAL 05

switch

Input two numbers and display the outputs of the basic mathematic operations. The output screen should be displayed as follows;

```
Enter two numbers _ _

1.  +

2.  -

3.  *

4.  /

Please enter your choice _
```

```
#include<stdio.h>
int main () {
    int choice;
    float no1, no2;

    printf("enter two numbers");
    scanf("%f %f, &no1 , &no2");

    printf("1. +\n");
    printf("2. -\n");
    printf("3, *\n");
    printf("4. /\n");
    printf("please enter your choice");
    scanf("%f, &choice");

    switch (choice) {
        case 1:
           printf("result %.2f\n", no1 + no2);
           break;
        case 2:
           printf("result %.2f\n", no1 - no2);
           break;
       case 3:
           printf("result %.2f\n ", no1 * no2);
           break;
       case 4:
```

```c
     if (no2 != 0) {
      printf("result %.2f\n", no1 / no2);
     } else {
      printf("error: division by zero is not allowed here\n");
     }
     break;
    default:
     printf("invalid choice\n");
     break;
  }

}
```

While loop

01. **Input 10 numbers and display the total count of odd & even numbers in the entered number series.**

```c
#include<stdio.h>

 int main () {
    int i = 1, evenc = 0, oddc = 0, no;

    printf("Enter 10 numbers:\n");

    while ( i <= 10) {
          printf("Enter number %d: ", i);
    scanf("%d", &no);

 if (no % 2 == 0) {
    evenc++;
  } else {
           oddc++;
          }

          i++;
            }
           printf("Total count of even numbers: %d\n", evenc);
           printf("Total count of odd numbers: %d\n", oddc);


      }
```

02. **Modify the above program in to enter series of numbers terminates when the user enter -99 and display the same expected output.**

```
#include<stdio.h>
int main () {
      int no , evenc = 0 , oddc = 0;
      printf("Enter numbers (terminate with -99):\n");;

     while(1) {
          printf("Enter a number: ");
         scanf("%d", &no);
      if (no == -99) {
              break; // terminate the loop when -99 is entered
             }
     if  (no % 2 == 0) {
           evenc++;
       } else {
          oddc++;
       }
     }
           printf("Total count of even numbers: %d\n", evenc);
           printf("Total count of odd numbers: %d\n", oddc);



}
```

Do while loop

Rewrite the programs for the above while loop question 1 & 2 using do while loop.

1.

```
#include<stdio.h>
int main () {
    int i = 1,evenc = 0, oddc = 0, no;

    printf("Enter 10 numbers:\n");

    do {
        printf("enter number %d: ", i);
        scanf("%d", &no);
```

```c
        if (no % 2 == 0) {
            evenc++;
        }else {
            oddc++;
        }
        i++;
    } while (i <=10 );
    printf("total count of even numbers: %d\n", evenc);
    printf("total count of odd numbers: %d\n", oddc );

}
```

2.

```c
#include<stdio.h>
int main () {
    int no, evenc = 0,oddc = 0;

    printf("enter numbers (terminate with -99):\n");

    do {
        printf("enter a number:");
        scanf("%d", &no);

        if (no== -99){
             break;//terminate the loop when -99 is entered
        }
         if (no % 2 ==-99) {
            evenc++;
         } else {
             oddc++;
         }
        } while (1);


         printf("total count of even numbers: %d\n", evenc);
         printf("total count of odd numbers: %d\n", oddc);


}
```

For loop

1.Input 10 numbers and display the average value using the for loop.

```
#include<stdio.h>
          int main () {
              int total = 0 , count = 0 ,num;
              for( int i = 0; i < 10; i++ ) {
                          printf("Enter number: ");
                                 scanf("%d", &num);

                         total += num;
                         count++;
             }
           float average = (float) total / count;

          printf("The average is: %.2f\n", average);

     }
```

2.Display the following output using the for loop.

```
*

**

***

****

*****
```

```
#include<stdio.h>
    int main () {
         for (int i = 1; i <= 5; i++) {
  for (int j = 1; j <= i; j++) {
       printf("*");
  }
     Printf("\n");
}

}
```