**Index no - 30251 Tutorial 01**

1. A programming language serves as a tool that enables humans to communicate instructions to computers effectively.It provides a set of rules and syntax for writing programs that can be understood and executed by a computer.

The need for a programming language arises from the fact that computers operate using binary code , which consists of sequences of 0 s and 1 s . While binary code is efficient for machines,it is highly complex and difficult for humans to work with directly.Programming languagea act as intermediaries,allowing humans to express their instructions in a more readable and manageable from.

2 a**) Source code Machine code**

Generated by a human. Generated by a machine.

High level code. Low level code.

Code exists in plain text. Code exists in binary format.

Human understandable. Machine understandable .

Easy to modify. Difficult to modify.

Written in various high level Written by an assembler,compiler or

Languages like Java,C,C++,Python any from of translator in to machine language

JavaScript or assembly language. from the source code.

Acts as the input to an assembler, Acts as the output to an assembler,compiler or

Compiler or a type of translator. a type of translator.

b**) High level language Low level language**

Progeammer friendly language. Machine friendly language.

Less memory efficient. High memory efficient.

Easy to understand. Tough to understand.

Simple to debug. Complex to debug comparatively.

Portable. Non -protable.

Can run on any platform. Machine dependent.

Needs compiler or interpreter Needs assembler for translation.

For translation.

C**) Compiler Interpreter**

Translate the source code entire Translate the source code line by line.

program.

More strict. More flexible.

Executes faster. Executes relatively slower.

Generates compiler independent Executes on the fly.

executable.

Detects many errors at the compile time . Detects many errors during the run time.

Need to compile program for each Relatively portable.

Supported platform.

d**) Structured language Object oriented language**

The program is represented as a The program is written as a collection of objects

Logical structure. Which communicate with each other.

The flow of execution of the programming The basic entity is object.each computation is

Is dependent on the structure of the performed using objects only.

program.

Code is given more importance. Data is given more importance.

Can handle up to moderately complex Can handle very complex programs.

programs.

Less data security. More data security.

Less code reusability. More reusability.

Flexibility is less. Flexibility is more.

Abstraction is less. Abstraction is more.

This is top-down approach. This is a bottom-up approach.

e**) C C++**

Function or procedure programming Object oriented programming language.

Language.

Top down approach is used in c. Bottom up approach is used in c++.

C is less secure than c++. C++ is secure , because oo’s concept is available.

Here Data is hidden and can’t be accessed by

External functions.

Variable declaration is possible only Can any where declare variable.

Top of program.

Namespace feature is not available. Namespace feature is present.

Middle level language. High level language.

Programs are divided into modules Programs are divided into classes and functions.

and functions.

Here no exception handling concept It support exception handling concept.

Is available.

C uses scanf() and printf() function for C++ uses cin>> and cout << for standard input

Standard input and output. and output.

Features like function overloading C++ supports function overloading and operator

and operator overloading is not present. overloading.

Program file is saved with C extension. Program file is saved with CPP extension.

f**) C++ Java**

Does’t support Pointer concept. Support pointer concept.

Does’t support multiple inheritances. Support multiple inheritance.

Does’t include structures or unions. Structure and union concept.

Automatic garbage collection. Requires explicit memory management.

Method overloding,but no operator Support both method overloading and operator

overloading. overloading.

Platform independent. Platfrom dependent.

Mainly used for design web based Used for design only desktop application like

application but also use for develop OS, Compiler etc.

application.

Uses compiler and interpreter both. Use only Compiler.

Supports header files. Does’t support header files. It uses the import key

word to include classes.

g**) Syntax error Logical error**

This is the error that occurs due to the This type of error causes unexpected and

violation of syntax rules of a particular unusual results for the program.

programming language.this usually

happens when we change the intended

syntax of writing the sequence of tokens

or characters.

The program terminates due to such errors

and cannot execute successfully until we rectify

It.

The cause is the fault in the program syntax. The cause is the fault in the logic or algorithm.

The compiler indicates the error and its The programmer has to find the error himself .

location itself in compiled languages.

It is easier to detect a syntax error. On the other hand , finding a logical error is

Comparatively more difficult.

**Tutorial 02**

1. **Comments can be used to explain code,and to make it more readable. It can also be used to prevent execution when testing alternative code.**

**Comments can be singled-lined or multi-lined.**

**Single-line comments start with two forward slashes (//).Any text between use // and the end of the compiler(will not be executed).**

**Ex- #include<stdio.h>**

**int main ()**

**{**

**// single line comment in c language**

**printf (“Hello world ! ‘’);**

**/\* multi line comment systax**

**comments help us to understand programme later easier.\*/**

**printf(‘’codebloks”);**

**return 0;**

**}**

1. **In a c program the main ‘main()’ function is essential . It serves as the entry point of the program, meaning that the execution of the program begins from this function.Every c program must have a main() function for it to be a valid executable program.**

**The’ main() ‘function has a specific signature.**

**#include<stdio.h>**

**Int main (void) {**

**// code satements**

**Return 0;**

**}**

1. **The scanf() function reads data from the standard input stream stdin in to the locations given by each entry in argument-list.Each argument must be a pointer to a variable with a type that corresponds to a type specifier in format string.**
2. **The C language is case – sensitive .This means that all language keyword,identifiers ,function names, and other variables must be entered with consistent letter capitalization.**

**Ex- NUMBER is a variable name in capital letters.In the same program two variables can be created with number simple letters.**

1. **a) record1 - Valid identifier**

**It starts with a letter and can be followed by letters,number,or uunderscores.**

**b) 1record – Invalid identifier.**

**Identifier can’t start with a number.**

**C) file - 3 – Invalid identifier.**

**Identifiers can’t contain hyphens. You can use underscores instead,like file\_3.**

**d) return – Valid identifier.**

**Although “return’’ is a keyword in many programming languages,it is still allowed as**

**An identifier in some contexts.**

**e) $tax – Valid identifier.**

**It starts with a dollar sign, which is allowed in many programming languages.**

**f) name – Valid identifier.**

**It consists of only letters and is not a reserved keyword.**

**g) name and address – Invalid Identifier .**

**Identifiers can’t contain spaces.You can use uderscores instead , like**

**name\_\_and\_\_address.**

**h) name-and-address – Invalid identifier.**

**Identifiers can’t contain hyphens.You can use underscores instead,like**

**name\_and\_address.**

**I) name\_and\_address – Valid identifier .**

**It consists of letters, underscores ,and numbers (though it is typically**

**recommended to avoid starting identifiers with numbers.)**

**j) 123-45-6789 - Invalid Identifier.**

**Identifiers can’t contain spaces or hyphens.**

**6) a) False.**

**The function ‘printf’ does’t automatically begin printing at the beginning of a new line. If you**

**want to start printing on a new line , you need to explicitly include the new line character ‘\n’ in**

**the format control string.**

**b) False .**

**Comments in programming are ignored by the compiler or interpreter and do not produce any**

**output when the program is executed. They are used to provide explanatory or descriptive text to**

**make the code more understandable to humans.**

**C) True.**

**The escape sequence ‘\n’ is used in a ‘printf’ format control string to insert a new line character,**

**which causes the cursor to move to the beginning of the next line when the output is displayed**

**on the screen.**

**d) True .**

**In most programming languages, including C, variables must be defined before they are used.**

**This means that you need to declare the variables and specifies their types before you can**

**perform any operations or assignments with them.**

**e) True.**

**In C ,all variables must be given a type when they are defined. The type specifies the kind of**

**data the variable can hold and the operations that can be performed on it. Without a type**

**The variable can’t be properly utilized in the program.**

**f) False.**

**In C , variable name are case – sensitive. Therefore, ‘number’ and ‘NuMbEr’ would be considered**

**as distinct and separate variables**

**g) False.**

**The number of ‘printf’ statements does’t necessarily correspond to the number of lines of**

**output. It is possible to print multiple lines of output using a single ‘printf’ statement by including**

**new line characters ‘\n’ or other appropriate formatting in the format control string.**

**7) #include<stdion .h>**

**int main ()**

**{**

**printf( “\*\n\*\*\n\*\*\*\n\*\*\*\*\n\*\*\*\*\*\n’’);**

**return o ;**

**}**

**8) a) scanf( ‘’d’’ , &value);**

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

**c) scanf ( ‘’%d’’ , &anInteger ) ;**

**d) The statement is correct.**

**e) The statement is correct.**

**f) Printf( "The value you entered is: %d\n ‘’, value ) ;**

**9) a) 2**

**b) 4**

**c) x=**

**d) x=2**

**e) 5 = 5**

**f) There will be no output**

**g) The output will depend on the user input**

**h****) There will be no output**

**i) There will be no output**

**Explanation: The new line character ‘\n’ is printed, which moves the cursor to a new line.**

**10) a) True.**

**In C, operators are evaluated from left to right, following the associativity rules. This means that if there are multiple operators of the same precedence level, they are evaluated in the order they appear from left to right.**

**b) True.**

**All the variable names listed are valid in C. Variable names can start with an underscore or a letter and can contain letters, numbers, and underscores. However, it is worth noting that starting variable names with an underscore is generally not recommended to avoid potential naming conflicts.**

**c) False.**

**The statement printf("a = 5;"); is not an assignment statement. It is a print statement that displays the text "a = 5;" on the console. An assignment statement would involve assigning a value to a variable, such as a = 5;.**

**d) True.**

**In the absence of parentheses, a valid arithmetic expression without any explicit grouping will be evaluated following the left-to-right order. The order of evaluation follows the precedence and associativity rules of the operators involved.**

**e) False.**

**The variable names "3g", "87", and "67h2" are invalid in C. Variable names cannot start with a number and should not contain any special characters except for underscores. Variable names must start with a letter or an underscore. The variable names "h22" and "2h" are valid since they start with a letter and only contain letters and/or numbers after the initial character.**

**Tutorial 03**

**1) #include <stdio.h>**

**int main()**

**{**

**int x=1;**

**printf("%d\n",x+1);**

**printf("%d\n",x+=1);**

**printf("%d\n",x++);**

**printf("%d\n",++x);**

**return 0;**

**}**

**2) a ) z = x++ + y ;**

**d ) product \*= 2 ;**

**c ) product = product \* 2 ;**

**d ) #include <stdio.h>**

**int main()**

**if (count > 10) ;**

**{**

**printf("Count is greater than 10 ");**

**return 0;**

**}**

**e ) total -= - - x ;**

**f ) total += x - - ;**

**g )**

**h ) #include <stdio.h>**

**int main()**

**{**

**printf("%.2f",123.4567);**

**return 0;**

**}**

**I ) #include <stdio.h>**

**int main()**

**{**

**printf****("%.3f",3.14159);**

**return 0;**

**}**

**3 ) a** **) scanf ("%d",&x) ;**

**b ) scanf ("%d",&y) ;**

**c ) int i = 1 ;**

**d ) int power = 1 ;**

**e ) power \*= x ;**

**f ) int i = i ++ ;**

**g ) while ( i < = y ) {**

**// code inside the while loop**

**}**

**h ) printf ("%d",power) ;**

**Tutorial 04**

**1 ) if (numNeighbors >= 3 || numNeighbors = = 4 )** **{**

**++numNeighbors ;**

**printf("You are dead! \n " ) ;**

**}**

**else {**

**-- numNeighbors ;**

**}**

**2 ) No, I'm here!**

**No, actually, I’m here!**

**3 ) if doesSignificantWork is ture , the code enters the first if block.**

**if** **makesBreakthrough is ture ,** **it sets** **nobelPrizeCandidate to ture.**

**if makesBreakthrough is false , it sets nobelPrizeCandidate to false.**

**if doSignificantWork is false , the code skips the first if block and enters another if block.**

**it sets nobelPrizeCandidate to false . In summary, if dosignificantWork is ture , the value of**

**nobelPrizeCandidate depends on the value of makesBreakthrough .**

**if doSignificantWork is false , nobelPrizeCandidate is always set to false.**

**4 ) #include <stdio.h>**

**int main()**

**{**

**char taxCode = 'T';**

**double price = 100.0,taxRate = 0.1,x,y,sum ,cost;**

**int opCode = 1 ,currentNumber = 7 , year = 2024, leapYear = 0,distance = 250;**

**if(taxCode == 'T'){**

**price += taxRate \* price;**

**}**

**printf("price : %.2f\n", price);**

**if(opCode == 1){**

**printf("Enter x value:");**

**scanf("%1f",&x);**

**printf("Enter y value:");**

**scanf("%1f",&y);**

**sum = x + y;**

**printf("sum : %.2f\n",sum);**

**}**

**if (currentNumber %2 !=0){**

**currentNumber = 3 \* currentNumber + 1;**

**} else {**

**currentNumber = currentNumber / 2;**

**}**

**printf("currentNumber : %d\n",currentNumber );**

**if (year %4 == 0){**

**if(year %100 == 0){**

**if(year %400 == 0){**

**leapYear = 1;**

**}**

**}else {**

**leapYear = 1;**

**}**

**}**

**printf("leapYear : %s\n",leapYear ? "true" : "false" );**

**if(distance <=100){**

**cost = 5.00;**

**}else if(distance <=500){**

**cost = 8.00;**

**}else if(distance < 1000){**

**cost = 10.00;**

**}else{**

**cost = 12.00;**

**}**

**printf("cost: %.2f\n",cost );**

**return 0;**

**}**

**Tutorial 05**

**1) switch**

**#include <stdio.h>**

**int main()**

**{**

**char op;**

**int n1,n2;**

**printf("Enter an operator (+,-,\*,/):");**

**scanf("%c",&op);**

**printf("Enter two numbers:");**

**scanf("%d %d",&n1,&n2);**

**switch(op){**

**case '+':**

**printf("%d + %d = %d",n1,n2 ,n1+n2);**

**break;**

**case '-':**

**printf("%d - %d = %d",n1,n2 ,n1-n2);**

**break;**

**case '\*':**

**printf("%d \* %d = %d",n1,n2, n1\*n2);**

**break;**

**case '/':**

**if (n2 != 0)**

**printf("%d / %d = %d",n1,n2,n1/n2);**

**else**

**printf("Error! Division by zero is not allowed.\n");**

**break;**

**default :**

**printf("Error! operator is not correct\n");**

**break;**

**}**

**return 0;**

**}**

**2 ) 1) while**

**#include <stdio.h>**

**int main()**

**{**

**int count = 1, odd = 0, even = 0 ,number;**

**printf("Enter ten numbers:\n");**

**while(count <= 10){**

**printf("Number %d :",count);**

**scanf("%d",&number);**

**if (number %2 == 0){**

**even++;**

**}else{**

**odd++;**

**}**

**count++;**

**}**

**printf("Total count of odd numbers: %d\n",odd);**

**printf("Total count of even numbers: %d\n",even);**

**return 0;**

**}**

**2 ) #include <stdio.h>**

**int main()**

**{**

**int number,numbers[100],count = 0;**

**printf("Enter a series of numbers(-99 to terminate):\n");**

**while(1){**

**scanf("%d",&number);**

**if (number == -99)**

**break;**

**numbers[count] = number;**

**count++;**

**}**

**printf("Entered numbers:\n");**

**for (int i = 0; i < count; i++){**

**printf("%d",numbers[i]);**

**}**

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

**return 0;**

**}**

**3) 1) dowhile**

**#include <stdio.h>**

**int main()**

**{**

**int numbers[10],count = 0,odd = 0,even = 0, number;**

**printf("Enter ten numbers:\n");**

**do{**

**scanf("%d",&number);**

**numbers[count] = number;**

**count++;**

**}while (count < 10);**

**for (int i = 0; i < 10; i++){**

**if (numbers[i] %2 == 0){**

**even++;**

**}else{**

**odd++;**

**}**

**}**

**printf("Total count of odd numbers: %d\n",odd);**

**printf("Total count of even numbers: %d\n",even);**

**return 0;**

**}**

**2 ) #include <stdio.h>**

**int main()**

**{**

**int numbers[100],count = 0,odd = 0,even = 0, number;**

**printf("Enter a series of numbers (-99 to terminate):\n");**

**do{**

**scanf("%d",&number);**

**if (number == -99);**

**break;**

**numbers[count] = number;**

**count++;**

**}while(1);**

**for ( i = 0; i < count; i++){**

**if (numbers[i] %2 == 0){**

**even++;**

**}else{**

**odd++;**

**}**

**}**

**printf("Total count of odd numbers: %d\n",odd);**

**printf("Total count of even numbers: %d\n",even);**

**return 0;**

**}**

**for loop**

**1) #include <stdio.h>**

**int main()**

**{**

**int i, number;**

**double sum = 0.0,average;**

**printf("Enter ten numbers:\n");**

**for (i = 0; i < 10; i++){**

**printf("number %d:",i);**

**scanf("%d",&number);**

**sum += number;**

**}**

**average = sum / 10;**

**printf("Average: %.2f\n",average);**

**return 0;**

**}**

**2) #include <stdio.h>**

**int main()**

**{**

**int rows = 5;**

**for (int i = 1; i <= rows; i++){**

**for(int j =1 ; j <= i; j++){**

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

**}**

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

**}**

**return 0;**

**}**