30418

Tutorial 1

1. People to communicate with computers to accomplished desired tasks

2. a)

Source code

- Using high level programming language which is human readable
- Portable across different platforms and operating systems
- High level of abstraction
- Translated into machine code through compilation
- Easy to modify
- Written in specific programming language
- Requires a compiler to execute
- Human readable and designed to understand and maintain the code

Machine code

- Using binary format which is not readable by human
- Specific to a certain targeted hardware and architecture
- Low level of abstraction
- Difficult to modify
- Not tied to a specific programming language
- Directly executable by the computer
- Not intended to understand or readable

b)

High level languages

- High level of abstraction
- Readable by humans

- Provides built in functions
- Abstracts memory management
- Slower execution speed

Low level languages

- Low level of abstraction
- Not readable by humans
- Lacks built in functions
- Requires manual memory management
- Higher execution speed
- Assembly language

c)

Compiler

- A compiler translates the entire source code in a single run.
- It consumes less time and it is faster than an interpreter.
- It is more efficient.
- CPU utilization is more.
- Both syntactic and semantic errors can be checked simultaneously.
- The compiler is larger.
- It is not flexible.

Interpreter

- An interpreter translates the entire source code line by line.
- It consumes much more time than the compiler i.e., it is slower than the compiler.
- It is less efficient.

- CPU utilization is less as compared to the compiler.
- Only syntactic errors are checked.
- Interpreters are often smaller than compilers.
- It is flexible.

d)

Structured languages

- Follows a procedural programming paradigm focusing on sequential execution and modular code structures
- Limited capabilities
- Does not support inheritance
- Does not support polymorphism

Object oriented languages

- Follows an object oriented programming paradigm organizing code around objects and their interactions
- Strong capabilities
- Supports inheritance
- Supports polymorphism

e)

C

- Data and functions are separated in C because it is a procedural programming language.
- C does not support information hiding.
- Built-in data types is supported in C.
- C is a function driven language because C is a procedural programming language.
- Function and operator overloading is not supported in C.
- C is a function-driven language.

C++

- Data and functions are encapsulated together in form of an object in C++.
- Data is hidden by the Encapsulation to ensure that data structures and operators are used as intended.
- Built-in & user-defined data types is supported in C++.
- C++ is an object driven language because it is an object oriented programming.
- Function and operator overloading is supported by C++.
- C++ is an object-driven language

f)

C++

- Platform dependent and needs to be compiled on every platform.
- Requires manual memory management.
- Does not have support for documentation comments.
- Procedural and object-oriented.
- C++'s source code, on the other hand, has no association with filenames.

Java

- Java is platform-independent. Once it's compiled into bytecode it can be executed on any platform.
- Java's memory management is system-controlled
- Has built-in support for comments which allows developers to provide documentation within their source files.
- A pure object-oriented programming language.
- Java's source code uses file names as classes, so file names should match any classes.

g)

Syntax error

- A syntax error is an error in the syntax of a sequence of characters or tokens that is intended to be written in a particular programming language.
- A syntax error occurs due to fault in the program syntax.
- In compiled languages, the compile indicates the syntax error with the location and what the error is.
- It is easier to identify a syntax error.

Logical error

- A logical error is an error in a program that causes it to operate incorrectly but not to terminate abnormally.
- A logical error occurs due to a fault in the algorithm.
- The programmer has to detect the error by himself.
- It is comparatively difficult to identify a logical error.

Tutorial 2

1.There are two ways to represent comments in C programming

- Single-line comments: To add a comment that spans a single line, you can use two forward slashes (//) followed by the comment text. Anything written after the double forward slashes will be considered a comment and will not be executed by the compiler.
- Multi-line comments: To add comments that span multiple lines, you can enclose the comment text between /* and */ . This allows you to write comments that can extend over multiple lines.

Uses of comments

- Use to describe the functionality of the programme and help the programmers to understand the code.

2.main () function

3.it use to take input from users, then store and process it.

4.yes

5. b, c and j are invalid, b-because digits are not allowed as the first character of an identifier c-because it contain hyphen which is not allowed in identifiers j-because numbers are not allowed as the first element and also hyphens are not allowed

6.a) false, if we want to begin with a new line we should use \n before the statement.

b) true

c)true

d)true

e)true

f)false,it should be identical

g)false,it can be written by one statement using \n

7.

*

**

***

****

*****

8.a) scanf("%d",&value);

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

c) scanf("%d",&aninteger); Multi-line comments: To add comments that span multiple lines, you can enclose the comment text between /* and */ . This allows you to write comments that can extend over multiple lines.

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

e)printf("the sum is %d\n",x+y);

f)printf("the value you entered is :%d\n",value);

9.a)2 b)4

c)x=

d)x=2

e)5=5

f)nothing

g)nothing

h)nothing

i)(a blank line)

10.a)true

b)true

c)false,it is a printf statement

d)false,it is according to the “BODMAS” theory

e)true

Tutorial 3

1.x++; ++x; x=x+1; x+=1;

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

b)x=2;

c)x=x*2;

d)if(count>10) {printf(“count is greater than 10\n”);}

e)total=--x;

f)total=x--;

g)q=q%divisor; q%=divisor;

h)printf(“%2f”,num);

123.46

i)printf(“%.3f”,num);

3.142

3.a)scanf(“%d”,&x); b)scanf(“%d”,&y);

c)int i=1;

d)int power=1;

e)power=x;

f)i++;

g)while(i<=y)

h)printf(“%d”,power);

Tutorial 4

1.ASSIGNMENT OPERATOR ERROR(not using equal comparison operator) Missing curly braces in if statement

2.output:-NO,actually I'm here!

The first if statement is true and the second if statement is false.

But there's no curly brackets.therefore it only prints what is after that if statement.

3.(question is not clear)

4.a)if(tax code==”T”) Price+=(taxrate/100)*price;

b)if(opcode=0) printf(“enter a number:”); scanf(“%f”,&x); printf(“enter a number”);

scanf(“%f”,&y); sum=x+y; printf(“%f”,sum);

c)if(currentnumber/2==1) currentnumber=3*(currentnumber)+1;

else

currentnumber=currentnumber/2;

d) if (year % 4 == 0)

{ if (year % 100 == 0)

{ if (year

% 400 == 0) { leapYear = true;

}

} else { leapYear = true;

}

}

if (leapYear) { printf("%d is a leap year.\n", year);

} else { printf("%d is not a leap year.\n", year); }

e) if (distance >= 0 && distance <= 100) { cost = 5.00;

} else if (distance > 100 && distance <= 500) { cost = 8.00;

} else if (distance > 500 && distance < 1000) { cost = 10.00; } else if (distance >= 1000) { cost = 12.00; } else { printf("Invalid distance entered!\n");

Tutorial 5

1. #include <stdio.h>
int main() { double num1, num2; int choice;
printf("Enter two numbers: "); scanf("%lf %lf", &num1, &num2);
printf("1. +\n"); printf("2. -\n"); printf("3. *\n"); printf("4. /\n"); printf("Please enter your choice: "); scanf("%d", &choice);
switch (choice) {
case 1:
printf("Result: %.2lf\n", num1 + num2); break; case 2: printf("Result: %.2lf\n", num1 - num2); break; case 3: printf("Result: %.2lf\n", num1 * num2); break; case 4:
if (num2 != 0) { printf("Result: %.2lf\n", num1 / num2);
} else { printf("Error: Division by zero!\n");
} break; default: printf("Invalid choice!\n"); break;
}
return 0;
}
WHILE LOOP
1. #include <stdio.h>
#include <stdlib.h>
int main() { int num,odd=0,even=0,count;
for (count=1;count<=10;count++)
{
printf("ENTER A NUMBER:"); scanf("%d",&num); if (num%2==0) even=even+1; else odd=odd+1;
}
printf("even=%d\n",even); printf("odd=%d",odd);

```
}
2. #include <stdio.h>
int main() { int num; int evenCount = 0, oddCount = 0;
printf("Enter numbers terminated by -99:\n");
while (1) { printf("Enter a number (or -99 to terminate): "); scanf("%d", &num);
if (num == -99) { break;
}
if (num % 2 == 0) { evenCount++; } else { oddCount++;
}
}
printf("Total even numbers: %d\n", evenCount); printf("Total odd numbers: %d\n", oddCount);
return 0; }
```

DO WHILE

```
1. #include <stdio.h>
int main() { int count = 0; int num; int evenCount = 0, oddCount = 0;
printf("Enter 10 numbers:\n");
do { printf("Enter number %d: ", count + 1); scanf("%d", &num);
if (num % 2 == 0) { evenCount++; } else { oddCount++;
}
count++;
} while (count < 10);
printf("Total even numbers: %d\n", evenCount); printf("Total odd numbers: %d\n", oddCount);
return 0; }
2. #include <stdio.h>
int main() { int num; int evenCount = 0, oddCount = 0;
printf("Enter numbers terminated by -99:\n");
do { printf("Enter a number (or -99 to terminate): "); scanf("%d", &num);
if (num == -99) { break;
}
if (num % 2 == 0) { evenCount++; } else { oddCount++;
}
```

```
} while (1);
printf("Total even numbers: %d\n", evenCount); printf("Total odd numbers: %d\n", oddCount);
return 0;
}
```

For loop

```
1. #include <stdio.h>
int main() { int num; int sum = 0; double average;
printf("Enter 10 numbers:\n");
for (int i = 0; i < 10; i++) { printf("Enter number %d: ", i + 1); scanf("%d", &num); sum +=
num;
}
average = (double) sum / 10;
printf("Average value: %.2lf\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; }
```