Tutorial 01

01.A programming language is a system of notation used for writing computer programs. It serves as a communication medium between humans and computers, allowing programmers to provide instructions to computers to perform specific tasks

02.

1. source code is the higher-level, human-readable representation of a program, while machine code is the lower-level, machine-executable form of the program. Source code is translated into machine code through compilation or interpretation processes, enabling the computer to execute the instructions.

1. High-level language and low-level language are different types of programming and scripting languages. High-level language is more abstracted from the hardware and more readable for the programmer. Low-level language is closer to the hardware and more understandable for the machine. High-level language is useful for building desktop, web, and mobile applications. Low-level language is useful for developing hardware-related applications such as device drivers, operating systems, and embedded systems.

High-level languages are easier to learn and slower to execute than low-level languages¹. Examples of high-level languages are C, C++, Java, Python, etc.

1. A compiler is a program that takes the entire source code and converts it into machine code all at once. The output of a compiler is a standalone executable file that can be run on the computer without the need for any other software. A interpreter, on the other hand, reads and executes the source code line by line, translating each line into machine code as it goes.

Compilers are generally faster than interpreters because they translate the entire program at once, whereas interpreters translate each line of code as it is executed. However, interpreters are more flexible than compilers because they can execute code on any platform without the need for recompilation

1. 1. Structured Programming :

Structured Programming, as name suggests, is a technique that is considered as precursor to OOP and usually consists of well-structured and separated modules. In this programming, user can create its own user-defined functions as well as this methodology tries to resolve issues that are associated with unconditional transfers to allow programmers follow logic of programs. It also requires more discipline at the design and logical structuring stage.

Example : Pascal, ALGOL, C, Modula-2, etc.

2. Object-Oriented Programming :

Object-Oriented Programming, as name suggests, is a different approach to programming that brings together data and functions that execute on them. It basically supports encapsulation, abstraction, inheritance, polymorphism, etc. It also includes data hiding feature therefore it is more secure. This model is based on real life entities that focuses on by whom task is to be done rather than focusing on what to do.

Example : JAVA, C#, C++, etc.

1. C and C++ have similarities, C++ expands on C by adding OOP features and providing a more extensive standard library. The choice between the two depends on the project requirements and the developer's familiarity with the languages.

1. C++ and Java depends on factors such as project requirements, desired performance, platform compatibility, and individual preference. C++ is well-suited for performance-critical applications, while Java excels in cross-platform development and enterprise environments.

1. Syntax Error:

A syntax error occurs when the code violates the rules of the programming language's syntax. It means that the code is written in a way that the compiler or interpreter cannot understand. Syntax errors prevent the code from being executed or compiled successfully. Common examples of syntax errors include missing or mismatched punctuation, incorrect variable or function names, and incorrect use of language keywords.

Logical Error:

A logical error, also known as a semantic error, occurs when the code does not function as intended or produces incorrect results. It is a mistake in the logic or algorithm of the program. Unlike syntax errors, logical errors do not cause the code to fail to compile or execute. Instead, they result in undesired outputs or unexpected behavior during runtime. Logical errors can be challenging to identify and fix as they require careful analysis and debugging of the code's logic.

In summary, a syntax error is a violation of the language's rules, preventing successful compilation or execution, while a logical error is a mistake in the program's logic, resulting in incorrect outputs or behavior during runtime.

Tutorial 02

1. In C, you can write comments to add explanatory or descriptive text within your code. Comments are ignored by the compiler and do not affect the program's execution. They serve the purpose of providing information and making the code more understandable to developers.

1. In a C program, the `main()` function is essential. It is the entry point of the program, meaning that it is the first function that

is called when the program starts executing. The `main()` function is required for a C program to be executable.

The `main()` function has a specific signature and syntax in C:

```c int main() { // code statements

return 0;

}

```

The `main()` function typically contains the main logic of the program, where statements and function calls are written to achieve the desired functionality. It can receive command-line arguments as parameters if specified as `int main(int argc, char \*argv[])`.

When a C program is executed, the execution starts from the first statement within the `main()` function and proceeds sequentially until it reaches the end of the function or encounters a return statement. The return value of the `main()` function serves as the exit status of the program: 0 usually indicates successful execution, and non-zero values typically represent error conditions.

While it is possible to define other functions in a C program, the `main()` function is mandatory and acts as the starting point for the program's execution.

1. The scanf() function in C is used to read input from the user or a file. Its purpose is to take formatted input and assign the extracted values to specified variables. scanf() allows the program to receive input during runtime, making it interactive and versatil

1. Yes, the "standard C" programming language is case sensitive. This means that distinctions are made between uppercase and lowercase letters in identifiers (variables, functions, keywords, etc.) within the C language.

1. (a) record1: Valid identifier.
2. 1record: Invalid identifier. It starts with a numeric digit, which is not allowed as the first character of an identifier in C. Identifiers must start with a letter or an underscore.
3. file-3: Invalid identifier. It contains a hyphen, which is not allowed in C identifiers. Only letters, digits, and underscores are permitted.
4. return: Valid identifier. Although "return" is a keyword in C, it is allowed as an identifier.
5. $tax: Valid identifier. It begins with a dollar sign, which is allowed in C identifiers. However, using a dollar sign as a prefix is not a common convention in C programming.
6. name: Valid identifier.
7. name and address: Invalid identifier. It contains spaces, and spaces are not allowed in C identifiers. Instead, you can use underscores to separate words, like "name\_and\_address".
8. name-and-address: Invalid identifier. It contains a hyphen, which is not allowed in C identifiers.
9. name\_and\_address: Valid identifier.
10. 123-45-6789: Invalid identifier. It starts with a numeric digit and contains hyphens, which are not allowed in C identifiers.

6.a) False. The `printf` function in C does not automatically begin printing at the beginning of a new line. It continues printing from the current cursor position on the same line unless specified otherwise.

1. False. Comments in C, denoted by `/\*` and `\*/`, are ignored by the compiler and do not affect the execution of the program. They are used for providing explanatory notes or disabling code temporarily. The text enclosed between `/\*` and `\*/` is not printed on the screen when the program is executed.

1. True. The escape sequence `\n` is used in C's `printf` function to insert a newline character. When used within a `printf` format control string, `\n` causes the cursor to position at the beginning of the next line on the screen.

1. False. In C, it is not required to define all variables before they are used. However, variables should be declared before they are used to inform the compiler about their existence and type. The actual definition (assignment of initial value) can occur later in the code.

1. True. In C, all variables must be given a type when they are defined. The type defines the nature of the variable, such as integer, floating-point, character, etc., and determines the memory allocated to it and the operations that can be performed on it.

1. False. C is case sensitive, so the variables `number` and `NuMbEr` would be considered as distinct and separate variables with different names.

1. False. A program that prints three lines of output does not necessarily require three `printf` statements. It could use a single `printf` statement with appropriate formatting, including escape sequences like `\n` to introduce line breaks within the output. Multiple lines of output can be generated with a single `printf` statement.

1. \*

\*\*

\*\*\*

\*\*\*\*

\*\*\*\*\*

1. a) Error: The format specifier in `scanf` is missing the `%` symbol before the `d`. Correction: `scanf("%d", &value);`

1. Errors: Missing a comma at the end of the format string. The closing double quote should be placed after the comma.

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

1. Error: The function `Scanf` is written with an uppercase 'S'. C is a case-sensitive language, so the correct function name should be `scanf`.

Correction: `scanf("%d", &anInteger);`

1. No errors found.

1. Error: The function `print` is undefined in C. The correct function for printing is `printf`. The comma is missing between the format string and the argument list.

Correction: `printf("The sum is %d\n", x + y);`

1. Error: The function `Printf` is written with an uppercase 'P'. C is a case-sensitive language, so the correct function name should be `printf`. The ampersand symbol should not be included before `value` since it's not necessary for `printf`.

Correction: `printf("The value you entered is: %d\n", value);`

9. a) printf("%d", x); prints "2".

1. printf("%d", x + x); prints "4".

1. printf("x="); prints "x=".

1. printf("x=%d", x); prints "x=2".

1. printf("%d = %d", x + y, y + x); prints "5 = 5".

1. Nothing is printed because it is an assignment statement (z = x + y;), not a printing statement.

1. The scanf function reads input from the user, so it waits for input. Nothing is printed immediately unless the user provides input.

1. The line is commented out, so nothing will be printed.

1. printf("\n"); prints a newline character. So, it prints a newline (blank line).

10. a) True. In C, operators are evaluated from left to right. This means that in an expression with multiple operators of the same precedence level, the operators are applied in the order they appear from left to right.

1. True. Each of the provided variable names is a valid variable name in C. Variable names can include letters (both uppercase and lowercase), digits, and underscores. However, they cannot start with a digit.

1. False. The statement `printf("a = 5;");` is not an assignment statement but a `printf` statement that will print the text "a = 5;" as it is. An assignment statement assigns a value to a variable, such as `a = 5;`, where the variable `a` is assigned the value 5.

1. True. In C, a valid arithmetic expression containing no parentheses is evaluated from left to right. The operators in the expression are applied in the order they appear, following the precedence and associativity rules of the operators.

1. True. The variable names "3g", "87", "67h2", "h22", and "2h" are all invalid variable names in C. Variable names cannot start with a digit; they must start with a letter or an underscore.

Tutorial 03

Q1. Sure! Here are four different C statements that each add 1 to an integer variable `x`:

1. `x = x + 1;`

This statement assigns the incremented value of `x` back to `x`.

1. `x += 1;`

This statement is a shorthand notation for adding 1 to `x`. It is equivalent to `x = x + 1;`.

1. `x++;`

This is the post-increment operator, which increments the value of `x` by 1 after the statement is executed.

1. `++x;`

This is the pre-increment operator, which increments the value of `x` by 1 before the statement is executed.

All four statements achieve the same result of adding 1 to the value of `x`.

Q2.

1. `z = x++ + y;`

This statement assigns the sum of `x` and `y` to `z`, then increments the value of `x` by 1.

1. `product \*= 2;`

This statement multiplies the variable `product` by 2 using the `\*=` operator. It is equivalent to `product = product \* 2;`.

1. `product = product \* 2;`

This statement multiplies the variable `product` by 2 using the `=` and `\*` operators.

d)

```c if (count > 10) { printf("Count is greater than 10.");

}

```

This statement tests if the value of the variable `count` is greater than 10. If it is, it prints the message "Count is greater than 10."

1. `total -= --x;`

This statement decrements the variable `x` by 1, then subtracts it from the variable `total`.

1. `total += x-- - 1;`

This statement adds the variable `x` to the variable `total`, then decrements `x` by 1.

1. ```c

q = q % divisor;

```

or ```c

q %= divisor;

```

These statements calculate the remainder after `q` is divided by `divisor` and assign the result back to `q`. Both statements are equivalent.

1. `printf("%.2f", 123.4567);`

This statement prints the value `123.4567` with 2 digits of precision. The value printed would be `123.46` because the precision rounds the number to two decimal places.

1. `printf("%.3f", 3.14159);`

This statement prints the floating-point value `3.14159` with three digits to the right of the decimal point. The value printed would be `3.142` because the precision rounds the number to three decimal places.

Q3.

1. `scanf("%d", &x);`

This statement inputs an integer value from the user and stores it in the variable `x` using the `scanf` function.

1. `scanf("%d", &y);`

This statement inputs an integer value from the user and stores it in the variable `y` using the `scanf` function.

1. `int i = 1;`

This statement initializes an integer variable `i` with the value 1.

1. `int power = 1;`

This statement initializes an integer variable `power` with the value 1.

1. `power \*= x;`

This statement multiplies the variable `power` by `x` and assigns the result back to `power`. It is equivalent to `power = power \* x;`.

1. `i++;`

This statement increments the variable `i` by 1. It is equivalent to `i = i + 1;`.

1. `while (i <= y)`

This statement tests if `i` is less than or equal to `y` as the condition of a `while` loop. The statements within the loop will execute as long as the condition is true.

1. `printf("%d", power);`

This statement outputs the value of the integer variable `power` using the `printf` function. The value will be printed as an integer.

Tutorial 04

01. The following if statement has multiple errors:

```c

if numNeighbors >= 3 || numNeighbors = 4

++numNeighbors;

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

--numNeighbors;

```

Errors:

1. Missing parentheses: The condition inside the if statement is missing parentheses around it.
2. Assignment instead of comparison: In the condition, `numNeighbors` is being assigned the value of 4 (`numNeighbors = 4`) instead of comparing it (`numNeighbors == 4`).

1. Inconsistent indentation: The indentation is inconsistent within the if statement, making it unclear which statements are intended to be inside the if block.

Corrected version:

```c

if (numNeighbors >= 3 || numNeighbors == 4)

{

++numNeighbors; printf("You are dead! \n");

}

else

{

--numNeighbors;

}

```

In the corrected version, the errors have been fixed:

1. The condition now has parentheses around it.
2. The equality comparison `==` is used instead of assignment `=` in the condition.
3. The statements within the if and else blocks are enclosed within curly braces for clarity.

02. The poorly indented program segment will produce the following output:

```

No, actually, I’m here!

```

Explanation:

The program segment consists of three lines of code, and there is an issue with the indentation that affects the control flow and the relationship between the if statements and the corresponding print statements.

Let's improve the indentation for better understanding:

```c

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");

```

Now, let's analyze the code:

1. The variable `number` is assigned the value 4, and the variable `alpha` is assigned the value -1.0.
2. The first `if` statement checks if `number` is greater than 0. Since `number` is indeed greater than 0 (it has a value of 4), the control flow enters the body of the outer `if` statement.
3. Inside the body of the outer `if` statement, there is a nested `if` statement that checks if `alpha` is greater than 0. However, `alpha` has a value of -1.0, so this inner `if` condition evaluates to `false`.
4. Since the inner `if` condition is `false`, the corresponding `printf` statement inside the `else` block is executed. It displays the message "No, I'm here!".
5. Next, there is a `printf` statement outside of both `if` statements. This statement is not related to any specific condition and will always execute. It displays the message "No, actually, I'm here!".

Therefore, the output produced by the program will be:

```

No, actually, I'm here!

```

The "No, actually, I'm here!" message is printed regardless of the values of `number` and `alpha` because it is outside both `if` statements.

03. The given if statement can be represented as follows:

```c

if (doesSignificantWork) { if (makesBreakthrough) nobelPrizeCandidate = true;

else nobelPrizeCandidate = false;

}

else if (!doesSignificantWork) nobelPrizeCandidate = false;

```

Let's break down the logic:

1. The outer `if` statement checks the condition `doesSignificantWork`.
2. If `doesSignificantWork` is `true`, the control flow enters the body of the outer `if` statement.
3. Inside the body of the outer `if` statement, there is an inner `if` statement that checks the condition `makesBreakthrough`.
4. If `makesBreakthrough` is `true`, it assigns `true` to `nobelPrizeCandidate`.
5. If `makesBreakthrough` is `false`, it assigns `false` to `nobelPrizeCandidate`.
6. If the condition `doesSignificantWork` is `false`, the control flow skips the body of the outer `if` statement and moves to the `else if` condition `!doesSignificantWork`.
7. If `doesSignificantWork` is `false`, it assigns `false` to `nobelPrizeCandidate`.

In summary, the following assignments are made based on the given conditions:

* If `doesSignificantWork` is `true` and `makesBreakthrough` is `true`, `nobelPrizeCandidate` is assigned `true`.
* If `doesSignificantWork` is `true` and `makesBreakthrough` is `false`, `nobelPrizeCandidate` is assigned `false`.
* If `doesSignificantWork` is `false`, `nobelPrizeCandidate` is always assigned `false`.

Therefore, the value of `nobelPrizeCandidate` depends on the values of `doesSignificantWork` and `makesBreakthrough`. If `doesSignificantWork` is `false`, `nobelPrizeCandidate` will always be `false`. Otherwise, it will depend on the value of `makesBreakthrough`.

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 integer variable opCode has the value 1, read in double values for X and Y and calculate and print their sum.
* 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).

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

Distance Cost

----------------------------------- ----------

0 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

Tutorial 05

Switch-

#include <stdio.h>

int main() {

int first, second, add, subtract, multiply; float divide;

printf("Enter two numbers:\n"); scanf("%d %d", &first, &second);

add = first + second; subtract = first - second; multiply = first \* second; divide = (float)first / second; // Typecasting to float

printf("1. Addition: %d\n", add); printf("2. Subtraction: %d\n", subtract); printf("3. Multiplication: %d\n", multiply); printf("4. Division: %.2f\n", divide);

return 0;

}

While loop-

01.

#include <stdio.h>

int main() { int numbers[10]; int evenCount = 0; int oddCount = 0;

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

for (int i = 0; i < 10; i++) { scanf("%d", &numbers[i]); if (numbers[i] % 2 == 0) { evenCount++;

} else {

oddCount++;

}

}

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

return 0;

}

02.

Do while loop-

For loop-

01.

#include <stdio.h>

int main() { int numbers[10]; int sum = 0;

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

for (int i = 0; i < 10; i++) { printf("Number %d: ", i + 1); scanf("%d", &numbers[i]); sum += numbers[i];

}

double average = (double)sum / 10;

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

return 0;

}

02.

#include <stdio.h>

int main() {

int i, j;

for (i = 1; i <= 5; i++) { for (j = 1; j <= i; j++) { printf("\*");

}

printf("\n");

}

return 0;

}