C PROGRAMMING NOTES

What is code?

* Code refers to a set of instructions written in a programming language that a computer can understand and execute.

What is coding?

* Coding is the process of writing code.

What is program?

* A program is a complete set of code that performs a specific task.

What is programming?

* Programming is the broader process of creating programs.
* It involves planning, designing, writing, testing, and debugging code.
* Coding is one of the part of programming.

Types of programming: -

1. Procedural Oriented Programming 2. Object-oriented Programming

What is programming language?

* A programming language is a special set of instructions with its own grammar and syntax, just like English or French.
* There are many different programming languages, each suited for different tasks.
* For example, C,C++,Java, Python etc

Levels of programming language: -

This refers to how close the programming language is to the computer's hardware.

There are two main Levels of programming based on abstraction: -

1. Low level language
2. High level language

1.Low level language: -

* Low level languages are closer to the machine’s native language.
* These languages are more difficult to read, write and debug.
* It offers great control over computer hardware resources.
* There are two categories in low level language.

1. Machine language 2. Assembly language
   * 1. Machine language: -

* It is in the form of Binary.
* Binary contains 0’s and 1’s called as bits.
* This is the most basic language that the computer can directly execute.
* Example: - 1010001001, 0010001001 etc.

ii)Assembly language: -

* This language uses Mnemonics (abbreviated codes) to represent machine code instruction.
* It is more readable than machine code instructions.
* To write this low level language, we have a deep understanding of the computer’s architecture.
* Example: - ADD R1, R2

SUB R2, R1

2.High level language: -

* These languages are designed to be easier to human to read and write.
* It is defined in English words and symbols.
* Writing code and debugging is easy.
* They requires less knowledge about the computer’s architecture.
* Example: - C, C++, Java, Python.

What is language Translator?

* A language translator is a software which is used to translate one language to another language.
* The programs are written mostly in high-level languages like Java, C++, Python etc. and are called source code. These source code cannot be executed directly by the computer and must be converted into machine language to be executed. Hence, a special translator system software is used to translate the program written in a high-level language into machine code
* There are generally three translators. They are: -
  + Assembler
  + Compiler
  + Interpreter

1. Assembler: -

It is a software that converts a assembly code to machine code.

Examples: - NASM, GAS

1. Compiler: -

* A compiler translates high level language (Source code) into intermediate form (or) machine code
* It converts complete code into machine code, if the code does not contain any errors.
* Compiler is faster than interpreter.
* Compiler gives more than one line error.
* Example of compiler languages are: - C, C++ etc.

1. Interpreter: -
   * Interpreter translate code line-by-line and execute it immediately.
   * It doesn’t generate any intermediate code.
   * Interpreter is slower than compiler.
   * Interpreter gives one line error.
   * Examples of interpreter languages are: - Python, Perl, Javascript.

|  |  |  |
| --- | --- | --- |
| **Feature** | **Compiler** | **Interpreter** |
| **Type Checking** | Types are determined during compilation, leading to early detection of type errors. | Types are determined at runtime, which can introduce type-related errors during execution. |
| **Translation Time** | Translates the entire program during compilation before execution. | Translates the program on-the-fly during runtime, interpreting code line by line. |
| **Output Storage** | Generates machine code stored as an executable file on disk. | Translated instructions are temporarily stored in RAM, without creating a separate executable. |
| **Optimization** | Can perform extensive compile-time optimizations for enhanced performance. | Limited optimization opportunities due to runtime translation, affecting performance. |
| **Application Suitability** | Ideal for performance-critical applications like operating systems and games. | Suited for scripting, rapid prototyping, and applications where immediate execution is beneficial. |
| **Security** | Offers better security through code obfuscation as the source is compiled into an executable. | Source code is more exposed, potentially posing higher security risks if left unprotected. |
| **Debugging** | Debugging can be complex as errors need to be traced back from the executable to the source code. | Facilitates easier debugging with errors reported in the context of the source code during execution. |
| **Execution Speed** | Generally faster due to direct execution of machine code. | Slower execution speed owing to on-the-fly interpretation and execution. |
| **Platform Dependency** | Produces platform-specific executables requiring recompilation for different platforms. | Highly portable, as the same code can run on any platform with a compatible interpreter. |
| **Memory Efficiency** | More memory-efficient as unnecessary code can be optimized out during compilation. | Less memory-efficient due to the need to keep the entire program in memory during interpretation. |
| **Development Cycle** | Longer development cycles due to the compile-link-execute steps. | Shorter development cycles beneficial for rapid testing and prototyping. |
| **Runtime Flexibility** | Less flexible at runtime since changes require recompilation. | Highly flexible, allowing for dynamic code execution and modifications at runtime. |
| **Language Examples** | C, C++, Rust are commonly compiled languages. | Python, JavaScript, and Ruby are popular interpreted languages. |

Compiler vs Interpreter

Introduction to C

What is C?

* C is a general-purpose, high−level programming language developed by Dennis M. Ritchie at bell laboratory.
* Its applications are very diverse. It ranges from developing operating systems to databases and all. It is system programming language used to do low level programming (e.g., driver or kernel)
* C was invented to write an operating system called UNIX. The UNIX OS was totally written in C.

Features of C: -

* **Procedural Language: -** C follows a procedural paradigm because a C program is a series of instructions that explain the procedure of solving a given problem. It makes the development process easier.
* **Fast and Efficient:** - C is known for being high−performing and efficient. It can let you work with memory at a low level, as well as allow direct access to hardware, making it ideal for applications requiring speed and economical resource use.
* **Modularity: -** We can divide a large program into small modules i.e., functions.
* **Statically Type:** - C programming language is a statically typed language. Meaning the type of variable is checked at the time of compilation but not at run time. This means each time a programmer types a program they have to mention the type of variables used.
* **Portability:** - C programs are machine-independent which means that you can compile and run the same code on various machines with none or some machine-specific changes.
* **General-Purpose Language:** - C language hasn’t been developed with a specific area of application as a target. From system programming to photo editing software, the C programming language is used in various applications.
* **Rich set of built-in Operators:** - C is perhaps the language with the most number of built-in operators which are used in writing complex or simplified C programs. In addition to the traditional arithmetic and comparison operators, its binary and pointer related operators are important when bit-level manipulations are required.
* **Easy to Extend:** - C is an extensible language. It means if a code is already written, you can add new features to it with a few alterations. Basically, it allows adding new features, functionalities, and operations to an existing C program.
* **Libraries with Rich Functions:** - C compilers are bundled with an extensive set of libraries with several built-in functions. It includes OS-specific utilities, string manipulation, mathematical functions, etc.
* **Pointers in C: -** One of the unique features of C is its ability to manipulate the internal memory of the computer. With the use of pointers in C, you can directly interact with the memory.
* **High Language:** - C is a high level language. Sometimes it refers as middle level language because it is close to hardware.

Application of C: -

* Operating Systems
* Language Compilers
* Assemblers
* Text Editors
* Print Spoolers
* Network Drivers
* Modern Programs
* Databases
* Language Interpreters
* Utilities

Drawbacks of C: -

* **Manual Memory Management** − C languages need manual memory management, where a developer has to take care of allocating and deallocating memory explicitly.
* **No Object−Oriented Feature** − Nowadays, most of the programming languages support the OOPs features. But C language does not support it.
* **No Garbage Collection** − C language does not support the concept of Garbage collection. A developer needs to allocate and deallocate memory manually and this can be error-prone and lead to memory leaks or inefficient memory usage.
* **No Exception Handling** − C language does not provide any library for handling exceptions. A developer needs to write code to handle all types of expectations.

Installation of C compiler: -

For windows read this: - [Download and Install C/GCC Compiler for Windows - Scaler Topics](https://www.scaler.com/topics/c/c-compiler-for-windows/)

For Mac read this: - [C Compiler for Mac | Setup C Compiler in Mac - Scaler Topics](https://www.scaler.com/topics/c/c-compiler-for-mac/)

For Linux read this: - [How to Install C and GCC Compiler on Linux - Scaler Topics](https://www.scaler.com/topics/c/install-c-on-linux/)

And

[How to Compile C Program in Linux? - Scaler Topics](https://www.scaler.com/topics/c/how-to-compile-c-program-in-linux/)

Compilation process of C: -

The compilation is the process of converting the source code of the C language into machine code.

For more read this article

[Compiling a C Program: Behind the Scenes - GeeksforGeeks](https://www.geeksforgeeks.org/compiling-a-c-program-behind-the-scenes/)

Or

[Compilation Process in C - Scaler Topics](https://www.scaler.com/topics/c/compilation-process-in-c/)

Structure of C program: -

Structure of C program indicates how it has to be written.

The basic structure of a C program is divided into 6 parts. They are: -

1. **Documentation Section: -** This section consists of the description of the program, the name of the program, and the creation date and time of the program. It is specified at the start of the program in the form of comments.
2. **Link Section:** - All the header files of the program will be declared in the preprocessor section of the program. Header files help us to access other’s improved code into our code. A copy of these multiple files is inserted into our program before the process of compilation. This section also called as Preprocessor section.

Example: #include<stdio.h>

#include<math.h>

1. **Definition Section: -** The define section comprises of different constants declared using the define keyword. It is given by:

#define PI = 3.14

1. **Global declaration Section:** - The global declaration section contains global variables, function declaration, and static variables. Variables and functions which are declared in this scope can be used anywhere in the program.

Example: int num = 18;

1. **Main Section: -** Every C program must have a main function. The main() function of the program is written in this section. Operations like declaration and execution are performed inside the curly braces of the main program. The return type of the main() function can be int as well as void too. void() main tells the compiler that the program will not return any value. The int main() tells the compiler that the program will return an integer value.

Example: void main()

or

int main()

1. **Sub program section:** - This includes the user-defined functions called in the main() function. User-defined functions are generally written after the main() function irrespective of their order.

**Example:** - void greet(){

printf(“Welcome!”);

}

**Example program to demonstrate Structure of C program. (see StructureOfC.c file)**

* **Important**
* Every C statement must end with a semicolon ( ; ). Thus ; acts as a statement terminator.
* Usually, each line contains one statement. However, you can write multiple statements in one line, provided each statement is terminated with a semicolon (;) **.**

**Character Set in C: -**

* In C programming, the character set refers to the collection of valid characters you can use to write your code.
* These characters are the building blocks for forming words, expressions, and even numbers within your program.
* These characters include letters, digits, special characters, and whitespace characters, all of which are used to form the tokens (keywords, identifiers, constants, strings, operators, and special symbols) in a C program.
* Categories of Characters in C
  + **Letters:** - Letters include both uppercase and lowercase English alphabets.
    - Uppercase letters: A, B, C, ..., Z
    - Lowercase letters: a, b, c, ..., z
  + **Digits: -** Digits include the numeric characters from 0 to 9.
    - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
  + **Special Characters: -** Special characters include various punctuation marks and symbols used in C for different purposes such as operators, separators, and other syntactical constructs.
    - **Arithmetic Operators:** +, -, \*, /, %
    - **Relational Operators:** <, >, ==, !=, <=, >=
    - **Logical Operators:** &&, ||, !
    - **Punctuation:** ;, :, ,, ., ?, !
    - **Brackets and Parentheses:** (, ), [, ], {, }
    - **Quotes:** ', "
    - **Backslash:** \
    - **Ampersand:** &
    - **Dollar Sign:** $
    - **Hash or Pound Sign:** #
  + **Whitespace Characters:** - Whitespace characters are used to separate tokens in the code. They do not carry any meaning but help in making the code more readable.
    - Space: ‘ ‘ (ASCII 32)
    - Horizontal Tab: \t (ASCII 9)
    - Vertical Tab: \v (ASCII 11)
    - Form Feed: \f (ASCII 12)
    - New Line: \n (ASCII 10)
    - Carriage Return: \r (ASCII 13)
* Each character in the set has a corresponding ASCII value, a unique number representing it in binary form.

**What are Tokens in C?**

* Tokens in C language are the minor elements or the building blocks used to construct or develop together a C program.
* These tokens in C are meaningful to the compiler.
* Think of tokens like words in a sentence - they come together to form instructions the compiler can understand.
* **Types of Tokens**: C tokens can be categorized into different types based on their function: (KISSCO)

Keywords Identifiers Strings

Special Symbols Constants Operators

* **Keywords:** - Keywords are reserved words that have a special meaning in C. They are predefined and cannot be used as identifiers (names of variables, functions, etc.).
  + There are 32 keywords in C. They are: -

|  |  |  |  |
| --- | --- | --- | --- |
| auto | break | case | char |
| const | continue | default | do |
| double | else | enum | extern |
| float | for | goto | if |
| int | long | register | return |
| short | signed | sizeof | static |
| struct | switch | typedef | union |
| unsigned | void | volatile | while |

* All the keywords are in small letters.
* **Identifiers:** - Identifiers are names given to various program elements such as variables, functions, arrays, etc.
  + There are some defined rules in C language for declaring identifiers:
    1. The first character of a identifier must be either an underscore or an alphabet.
    2. Identifiers shouldn't begin with any numerical digit
    3. Identifiers are case-sensitive and hence, both lowercase and uppercase letters are distinct.
    4. The length of identifiers shouldn't be more than 31 characters.
    5. Commas and blank spaces are not allowed while declaring an identifier.
    6. Except underscore (\_) no other special symbols are not allowed in identifiers.
    7. we can't use keywords as identifiers because keywords in C language are reserved words for special purposes only.
  + Some valid identifiers are: -

count, \_name, Start123, sTart\_123, count1\_, Double

* + Some invalid identifiers are: -

100days //started with a numerical digit

\_hello,morning //can't use comma operator

int //keyword

My name //can’t use space

Delhi(100) //circular brackets can't be used

**Difference between Keyword and Identifier**

|  |  |
| --- | --- |
| **Keyword** | **Identifier** |
| Keywords are special predefined reserved words which have special meanings and cannot be used elsewhere in the program. | Identifiers are used to define some variables, structures, unions, integers and other things. |
| It specifies the type of some particular variable. | It specifies the name of a particular variable. |
| It only consists of lowercase alphabets and starts with a lowercase. | It can have both uppercase and lowercase, and it can start with the alphabet or underscore. |
| No special symbol can be used. | Underscore can be used. |
| Example-: int, float, for, do, class etc. | Example-: Name, Roll\_Number etc. |

What is Variable in C?

* Variable is a name given to a location in memory that holds a value.
* This value can changed during program execution.
* Variables are fundamental building blocks because they allow us to store, manipulate, and retrieve data.
* **Rules for Constructing Variable Names**
* We cannot use any keyword as a Variable.
* All the Variables should have a unique name in the same scope.
* Variables cannot start with a digit.
* The first character of an Variable should always start with an alphabet or underscore, and then it can be followed by any of the Letter, digit, or underscore.
* The special characters (other than underscore \_ ) such as '\*','#','@','$' are not allowed within an Variable.
* All the Variables are case sensitive means the Variables "hello" and "Hello" will be treated differently. However, both names are identical, but one has a lowercase alphabet, and the other has an uppercase alphabet.
* Length of an Variable should not exceed 31 characters.
* Any blank spaces or commas are not allowed within an Variable.

**Declaration and Initialization of a variable: -**

First we have to declare a variable in order to use it.

1. Syntax to declare a variable: -

data\_type variable\_name;

Example: - int age;

Char grade;

float salary;

1. Syntax to declare a variable and initialize a value: -

initializing a variable means assigning it an initial value at the time of declaration.

data\_type variable\_name = value;

Example: - int age = 25;

Char grade = ‘A’;

float salary = 15000.00;

* **Strings: -** Strings are sequences of characters enclosed within double quotes. In C, strings are actually arrays of characters ending with a null character (\0).
  + Examples:

"Hello, World!"

"C programming"

**Note: -** We will learn about string in future.

* **Special symbols:** - Special symbols in C have specific meanings and are used to structure the code and define its syntax.
  + **Example: -** Square Brackets [ ]
  + The opening and closing square brackets represent single and multi-dimensional subscripts and they are used as array element reference for accessing array elements.
  + Example:

int arr[10]; //For declaring array, with size defined in square brackets

* + Special Symbols list are: -
    1. Square Brackets [] (used for arrays)
    2. Parenthesis () (used for functions)
    3. Curly Braces {} (used for defining a block)
    4. Comma , (used for separation)
    5. Pre-processor/Hash #
    6. Astrik \* (for pointers)
    7. Period/dot . (used for access element in structure)
    8. Colon : (used in terinary operator)
    9. Semicolon ; (to represent the end of statement)
    10. And more like ? <> \ / “ ” ‘ ’
* **Constants:** - A constant is an entity that doesn’t change.
  + In programming languages, constants are often called literals
  + **Types of C Constants:** - Constants in C can be divided into two major categories:
    1. Primary Constants – Integer, Real, Character
    2. Secondary Constants – Pointer, Array, String, Structure, Union, Enum.
* At this stage, we would restrict our discussion to only Primary constants, namely, Integer, Real and Character constants.
* Following Rules have been laid down for constructing these different types of constants:
* **Rules for Constructing Integer Constants**

(a) An integer constant must contain at least one digit.

(b) It must not contain a decimal point.

(c) Its value can be zero, positive or negative. If no sign precedes an integer constant, it is assumed to be positive.

(d) Commas or blanks are not allowed within an integer constant.

(e) The allowable range for integer constants is - 2147483648 to +2147483647.

Ex.: 426 +782 -8000 -7605 6

* The range of an Integer constant depends upon the compiler. For compilers like Visual Studio, GCC, it is -2147483648 to +2147483647, whereas for compilers like Turbo C or Turbo C++, the range is -32768 to +32767.
* The integer constant can be decimal numbers, octal numbers and hexa-decimal numbers.
* **Rules for Constructing Real Constants**
* Real constants are often called Floating Point constants. Real constants could be written in two forms—Fractional form and Exponential form.
* Following rules must be observed while constructing real constants expressed in fractional form:

(a) A real constant must contain at least one digit.

(b) It must contain a decimal point.

(c) It can be zero, positive or negative. Default sign is positive.

(d) Commas or blanks are not allowed within a real constant.

Ex.: +325.34 426.0 -32.76 -48.5792

* The exponential form is usually used if the value of the constant is either too small or too large. It, however, doesn’t restrict us from using exponential form for other real constants.
* In exponential form, the real constant is represented in two parts. The part appearing before ‘e’ is called mantissa, whereas the part following ‘e’ is called exponent.
* Thus 0.000342 can be written in exponential form as 3.42e-4 (which in normal arithmetic means 3.42 x 10-4).
* Following rules must be observed while constructing real constants expressed in exponential form:

(a) The mantissa part and the exponential part should be separated by a letter e or E.

(b) The mantissa part may have a positive or negative sign. Default sign is positive.

(c) The exponent must have at least one digit, which may be a positive or negative integer. Default sign is positive.

(d) Range of real constants expressed in exponential form is -3.4e38 to +3.4e38.

Ex.: +3.2e-5 4.1e8 -0.2E+3 -3.2e-5

* **Rules for Constructing Character Constants**

(a) A character constant is a single alphabet, digit or special symbol enclosed within single inverted commas.

(b) Both the single inverted commas should point to the left. For example, ’A’ is a valid character constant, whereas ‘A’ is not.

Ex.: 'A' 'I' '5' '='

|  |  |
| --- | --- |
| **Constant** | **Example** |
| Integer constant | 10, 20, 30 etc. |
| Floating-point constant | 10.2, 20.5, 30.6 etc. |
| Character constant | 'x', 'y', 'z' etc. |
| Octal constant | 011, 022, 088 etc. |
| Hexadecimal constant | 0x1a, 0x4b, 0x6b, etc. |

* **Operators:** - Operators are symbols that perform operations on variables and values.
  + C supports various types of operators, including arithmetic, relational, logical, bitwise, assignment etc.

What is Datatypes in C?

* The data type specifies the size and type of data the variable will store.
* A data type is the type of a variable can store.
* Data types in C tells the C compiler which type of data a variable holds.
* Data types are classified into 3 types. They are: -
  + Primary data type: - int, char, float, double
  + Derived data type: - Arrays, Strings, Pointer
  + User defined data type: - Structure, Union, enum, typedef
* Each data type has range and size.
* Range and size of memory allocate by each data type is depends on compilers architecture.