Lecture 1: Overview of C Programming Languages
CSE115.12- AAN

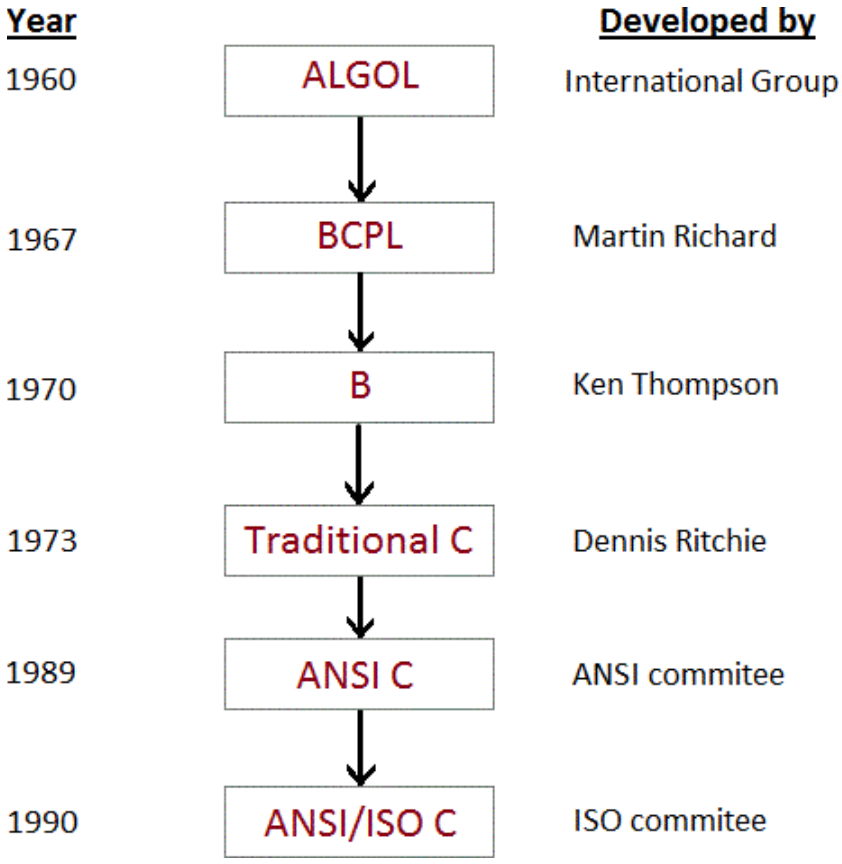
Overview of C Language

C is a structured programming language developed by Dennis Ritchie in 1973 at Bell Laboratories. It is one of the most popular computer languages today because of its structure, high-level abstraction, machine-independent feature, etc. C language was developed to write the UNIX operating system; hence it is strongly associated with UNIX, which is one of the most popular networks operating systems in use today and heart of internet data superhighway.

In the video below we have covered the complete introduction to the C language. If you want you can skip the video, as the concepts are covered in the topics below.

History of C language

C language has evolved from three different structured language ALGOL, BCPL and B Language. It uses many concepts from these languages while introduced many new concepts such as datatypes, struct, pointer etc. In 1988, the language was formalized by American National Standard Institute(ANSI). In 1990, a version of C language was approved by the International Standard Organization(ISO) and that version of C is also referred to as C89.



The idea behind creating C language was to create an easy language which requires a simple compiler and enables programmers to efficiently interact with the machine/system, just like machine instructions.

C language compiler converts the readable C language program into machine instruction.

Why C Language is so popular?

C language is a very good language to introduce yourself to the programming world, as it is a simple procedural language which is capable of doing wonders.

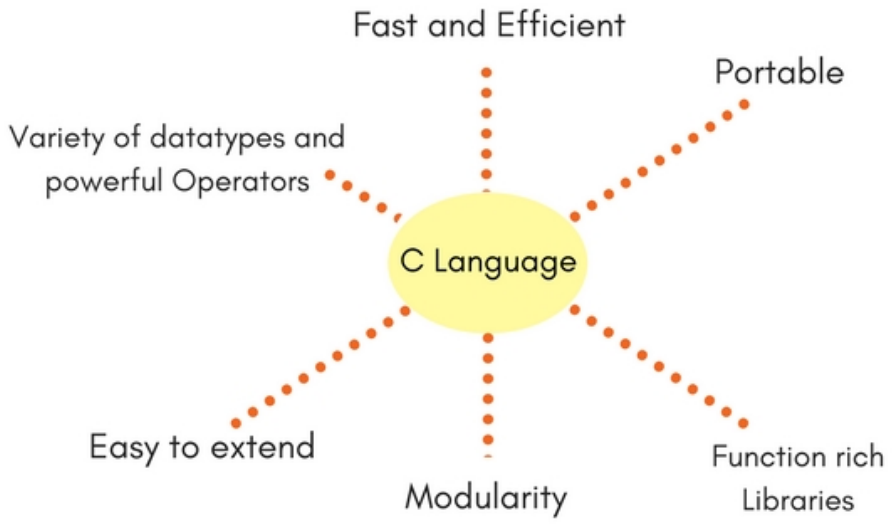
Programs written in C language takes very less time to execute and almost executes at the speed of assembly language instructions.

Initially C language was mainly used for writing system-level programs, like designing operating systems, but there are other applications as well which can be very well designed and developed using C language, like Text Editors, Compilers, Network Drivers etc.

Latest Version of C

The current latest version of C language is **C11**, which was introduced in 2011. It is supported by all the standard C language compilers.

Many new features have been introduced in this version and an overall attempt to improve compatibility of the C language with C++ language has been made. We will learn about the C11 edition, once we are done with learning C language, towards the end of this tutorial series.



First C Program

Let's see how to write a simple and most basic C program:

```
#include <stdio.h>

int main()

{

        printf("Hello,World");   //single line comment
/* multiline comments
/*

        return 0;

}
```

```
Output: Hello,World
```

Different parts of C program

1. Pre-processor Header file Function Variables
2. Statements & expressions Comments
3. All these are essential parts of a C language program.

Pre-processor

`#include` is the first word of any C program. It is also known as a **pre-processor**. The task of a pre-processor is to initialize the environment of the program, i.e. to link the program with the header files required.

So, when we say `#include <stdio.h>` it is to inform the compiler to include the **stdio.h** header file to the program.

STDIO = STandard Data Input Output

Header file

A Header file is a collection of built-in(readymade) functions, which we can directly use in our program. Header files contain definitions of the functions which can be incorporated into any C program by using pre-processor statement `#include` with the header file. Standard header files are provided with each compiler, and covers a range of areas like string handling, mathematical functions, data conversion, printing and reading of variables.

With time, you will have a clear picture of what header files are, as of now consider as a readymade piece of function which comes packaged with the C language and you can use them without worrying about how they work, all you have to do is include the header file in your program.

To use any of the standard functions, the appropriate header file must be included. This is done at the beginning of the C source file.

For example, to use the `printf()` function in a program, which is used to display anything on the screen, the line of `#include <stdio.h>` is required because the header file **stdio.h** contains the `printf()` function. All header files will have an extension `.h`

`main()` function

`main()` function is a function that must be there in every C program. Everything inside this function in a C program will be executed. In the above example, `main()` written before the `int` function is the return type of `main()` function. We will discuss about it in detail later. The curly braces `{ }` just after the `main()` function encloses the body of `main().`

Comments

We can add comments in our program to describe what we are doing in the program. These comments are ignored by the compiler and are not executed.

- To add a single line comment, start it by adding two forward slashes `//` followed by the comment.
- To add multiline comment, include it between `/* */`, just like in the program above.

Return statement - `return 0;`

A return statement is just meant to define the end of any C program. All the C programs can be written and edited in normal text editors like Notepad or Notepad++ and must be saved with a file name with extension as `.c`

If you do not add the extension `.c`, then the compiler will not recognize it as a C language program file.

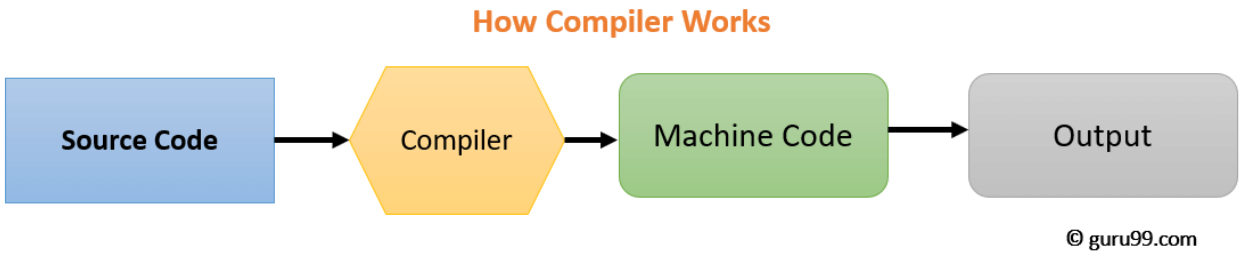
What is Compiler?

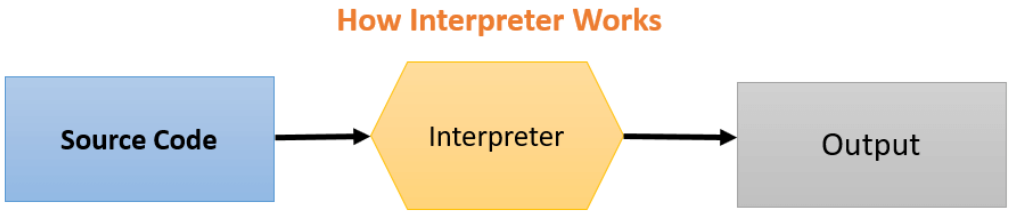
A compiler is a computer program that transforms code written in a high-level programming language into the machine code. It is a program which translates the human-readable code to a language a computer processor understands (binary 1 and 0 bits). The computer processes the machine code to perform the corresponding tasks.

A compiler should comply with the syntax rule of that programming language in which it is written. However, the compiler is only a program and cannot fix errors found in that program. So, if you make a mistake, you need to make changes in the syntax of your program. Otherwise, it will not compile.

What is Interpreter?

An interpreter is a computer program, which coverts each high-level program statement into the machine code. This includes source code, pre-compiled code, and scripts. Both compiler and interpreters do the same job which is converting higher-level programming language to machine code. However, a compiler will convert the code into machine code (create an exe) before program run. Interpreters convert code into machine code when the program is run.





Difference Between Compiler and Interpreter

Basis of difference	Compiler	Interpreter
Programming Steps	Create the program. Compile will parse or analyses all of the language statements for its correctness. If incorrect, throws an error If no error, the compiler will convert source code to machine code. It links different code files into a runnable program(known as exe) Run the Program	Create the Program No linking of files or machine code generation Source statements executed line by line DURING Execution
Advantage	The program code is already translated into machine code. Thus, its code execution time is less.	Interpreters are easier to use, especially for beginners.
Disadvantage	You can't change the program without going back to the source code.	Interpreted programs can run on computers that have the corresponding interpreter.
Machine code	Store machine language as machine code on the disk	Not saving machine code at all.
Running time	Compiled code run faster	Interpreted code run slower
Model	It is based on a language translation linking-loading model.	It is based on Interpretation Method.
Program generation	Generates output program (in the form of exe) which can be run independently from the original program.	Do not generate an output program. So, they evaluate the source program at every time during execution.
Execution	Program execution is separate from the compilation. It performed only after the entire output program is compiled.	Program Execution is a part of the Interpretation process, so it is performed line by line.
Memory requirement	The target program executes independently and does not require the compiler in the memory.	The interpreter exists in the memory during interpretation.
Best suited for	Bounded to the specific target machine and cannot be ported. C and C++ are the most popular programming language which uses compilation model.	For web environments, where load times are important. Due to all the exhaustive analysis is done, compiles take a relatively larger time to compile even small code that may not be run

		multiple times. In such cases, interpreters are better.
Code Optimization	The compiler sees the entire code upfront. Hence, they perform lots of optimizations that make code run faster	Interpreters see code line by line, and thus optimizations are not as robust as compilers
Dynamic Typing	Difficult to implement as compilers cannot predict what happens at turn time.	Interpreted languages support Dynamic Typing
Usage	It is best suited for the Production Environment	It is best suited for the program and development environment.
Error execution	The compiler displays all errors and warning at the compilation time. Therefore, you can't run the program without fixing errors	The interpreter reads a single statement and shows the error if any. You must correct the error to interpret the next line.
Input	It takes an entire program	It takes a single line of code.
Output	Compliers generates intermediate machine code.	Interpreter never generates any intermediate machine code.
Errors	Display all errors after, compilation, all at the same time.	Displays all errors of each line one by one.
Pertaining Programming languages	C,C++,C#, Scala, Java all use complier.	PHP, Perl, Ruby uses an interpreter.

Role of Compiler

- Compliers reads the source code, outputs executable code
- Translates software written in a higher-level language into instructions that a computer can understand. It converts the text that a programmer writes into a format the CPU can understand.
- The process of compilation is relatively complicated. It spends a lot of time analyzing and processing the program.
- The executable result is some form of machine-specific binary code.

Role of Interpreter

- The interpreter converts the source code line-by-line during RUN Time.
- Interpret completely translates a program written in a high-level language into machine level language.
- The interpreter allows the evaluation and modification of the program while it is executing.
- Relatively less time spent on analyzing and processing the program
- Program execution is relatively slow compared to the compiler.

High-Level Languages

High-level languages, like C, C++, JAVA, etc., are very near to English. It makes the programming process easy. However, it must be translated into machine language before execution. This translation process is either conducted by either a compiler or an interpreter. Also known as source code.

Machine Code

Machine languages are very close to the hardware. Every computer has a machine language. A machine language programs are made up of series of binary patterns. (E.g. `110110`) It represents the simple operations which should be performed by the computer. Machine language programs are executable so that they can be run directly.

Object Code

On compilation of source code, the machine code generated for different processors like Intel, AMD, an ARM is different. To make code portable, the source code is first converted to Object Code. It is an intermediary code (similar to machine code) that no processor will understand. At run time, the object code is converted to the machine code of the underlying platform.

C Input Output (I/O)

C Output

In C programming, `printf()` is one of the main output functions. The function sends formatted output to the screen. For example,

Example 1: C Output

```c
#include <stdio.h>
int main()
{
    // Displays the string inside quotations
    printf("C Programming");
    return 0;
}
```

Output

```
C Programming
```

How does this program work?

- All valid C programs must contain the `main()` function. The code execution begins from the start of the `main()` function.
- The `printf()` is a library function to send formatted output to the screen. The function prints the string inside quotations.
- To use `printf()` in our program, we need to include `stdio.h` header file using `#inclue <stdio.h>` statement.
- The `return 0;` statement inside the `main()` function is the "Exit status" of the program. It's optional.

Example 2: Integer Output

```c
#include <stdio.h>
int main()
{
    int testInteger = 5;
    printf("Number = %d", testInteger);
    return 0;
}
```

Output

```
Number = 5
```

We use `%d` format specifier to print `int` types. Here, the `%d` inside the quotations will be replaced by the value of `testInteger`.

Example 3: float and double Output

```
#include <stdio.h>
int main()
{
    float number1 = 13.5;
    double number2 = 12.4;

    printf("number1 = %f\n", number1);
    printf("number2 = %lf", number2);
    return 0;
}
```

Output

```
number1 = 13.500000
number2 = 12.400000
```

To print `float`, we use `%f` format specifier. Similarly, we use `%lf` to print `double` values.

Example 4: Print Characters

```
#include <stdio.h>
int main()
{
    char chr = 'a';
    printf("character = %c.", chr);
    return 0;
}
```

Output

```
character = a
```

To print `char`, we use `%c` format specifier.

C Input

In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

Example 5: Integer Input/Output

```c
#include <stdio.h>
int main()
{
    int testInteger;
    printf("Enter an integer: ");
    scanf("%d", &testInteger);
    printf("Number = %d",testInteger);
    return 0;
}
```

Output

```
Enter an integer: 4
Number = 4
```

Here, we have used `%d` format specifier inside the `scanf()` function to take `int` input from the user. When the user enters an integer, it is stored in the `testInteger` variable.

Notice, that we have used `&testInteger` inside `scanf().` It is because `&testInteger` gets the address of `testInteger,` and the value entered by the user is stored in that address.

Example 6: Float and Double Input/Output

```c
#include <stdio.h>
int main()
{
    float num1;
    double num2;

    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);

    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);

    return 0;
}
```

Output

```
Enter a number: 12.523
Enter another number: 10.2
num1 = 12.523000
num2 = 10.200000
```

We use `%f` and `%lf` format specifier for `float` and `double` respectively.

Example 7: C Character I/O

```c
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c",&chr);
    printf("You entered %c.", chr);
    return 0;
}
```

Output

```
Enter a character: g
You entered g.
```

When a character is entered by the user in the above program, the character itself is not stored. Instead, an integer value (ASCII value) is stored. And when we display that value using `%c` text format, the entered character is displayed. If we use `%d` to display the character, its ASCII value is printed.

Example 8: ASCII Value

```c
#include <stdio.h>
int main()
{
    char chr;
    printf("Enter a character: ");
    scanf("%c", &chr);

    // When %c is used, a character is displayed
    printf("You entered %c.\n",chr);

    // When %d is used, ASCII value is displayed
    printf("ASCII value is % d.", chr);
    return 0;
}
```

Output

```
Enter a character: g
You entered g.
ASCII value is 103.
```

I/O Multiple Values

Here's how you can take multiple inputs from the user and display them.

```c
#include <stdio.h>
int main()
{
    int a;
    float b;

    printf("Enter integer and then a float: ");

    // Taking multiple inputs
    scanf("%d %f", &a, &b);

    printf("You entered %d and %f", a, b);
    return 0;
}
```

Output

```
Enter integer and then a float: -3
3.4
You entered -3 and 3.400000
```

Format Specifiers for I/O

As you can see from the above examples, we use

- `%d` for `int`
- `%f` for `float`
- `%lf` for `double`
- `%c` for `char`

Here's a list of commonly used C data types and their format specifiers.

DATA TYPE	FORMAT SPECIFIER
`int`	`%d`
`char`	`%c`
`float`	`%f`
`double`	`%lf`
`short int`	`%hd`
`unsigned int`	`%u`
`long int`	`%li`
`long long int`	`%lli`
`unsigned long int`	`%lu`
`unsigned long long int`	`%llu`
`signed char`	`%c`
`unsigned char`	`%c`
`long double`	`%Lf`