C Language

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc.

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the **founder of the c language**.

It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.

C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible

1) Simple

C is a simple language in the sense that it provides a **structured approach** (to break the problem into parts), **the rich set of library functions**, **data types**, etc.

2) Machine Independent or Portable

Unlike assembly language, c programs **can be executed on different machines** with some machine-specific changes. Therefore, C is a machine independent language.

3) Mid-level programming language

Although, C is **intended to do low-level programming**. It is used to develop system applications such as kernel, driver, etc. It **also supports the features of a high-level language**. That is why it is known as mid-level language.

4) Structured programming language

C is a structured programming language in the sense that **we can break the program into parts using functions**. So, it is easy to understand and modify. Functions also provide code reusability.

5) Rich Library

C **provides a lot of inbuilt functions** that make the development fast.

6) Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) Pointer

C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array**, etc.

9) Recursion

In C, we **can call the function within the function**. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

10) Extensible

C language is extensible because it **can easily adopt new features**.

C programming is considered as the base for other programming languages, that is why it is known as mother language.

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

1) C as a mother language

C language is considered as the mother language of all modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

It provides the core concepts like the array, strings, functions, file handling, etc., used in many languages like C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, the Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.

A procedural language breaks the program into functions, data structures, etc.

C is procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of a procedural language. **Structure means breaking a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

5) C as a mid-level programming language

C is considered a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, and fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

C Program

In this tutorial, all C programs are given with a C compiler so that you can quickly change the C program code.

File: main.c

```c
#include <stdio.h>
int main() {
printf("Hello C Programming\n");
return 0;
}
```

Documentation

In a C program, single-line comments can be written using two forward slashes i.e., //, and we can create multi-line comments using /* */. Here, we've used multi-line comments.

```
/**
 * file: age.c
 * author: you
 * description: program to find our age.
 */
```

Link

All header files are included in this section.

A header file is a file that consists of C declarations that can be used between different files. It helps us in using others' code in our files. A copy of these header files is inserted into your code before compilation.

```
#include <stdio.h>
```

Definition

A preprocessor directive in C is any statement that begins with the "#" symbol.
The #define is a preprocessor compiler directive used to create constants. In simple terms, #define basically allows the macro definition, which allows the use of constants in our code.

```
#define BORN 2000
```

We've created a constant BORN which is assigned a value of 2000. Generally, uppercase letters are preferred for defining the constants. The above constant BORN will be replaced by 2000 throughout our code wherever used.

#define is typically used to make a source program easy to modify and compile in different execution environments.

The define statement **does not** ends with a semicolon.

Global Declaration

This section includes all global variables, function declarations, and static variables. The variables declared in this section can be used anywhere in the program. They're accessible to all the functions of the program. Hence, they are called global variables.

```
int age(int current);
```

We've declared our age function, which takes one integer argument and returns an integer.

Main() Function

In the structure of a C program, this section contains the main function of the code. The C compiler starts execution from the main() function. It can use global variables, static variables, inbuilt functions, and user-defined functions. The return type of the main() function can be void and also not necessarily int.

```
int main(void)
{
  int current = 2021;
  printf("Age: %d", age(current));
  return 0;
}
```

Here, we've declared a variable named current and assigned the value as 2021. Then we've called the printf() function, with calls the age() function, which takes only one parameter.

Subprograms

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.

When the user-defined function is called from the main() function, the control of the program shifts to the called function, and when it encounters a return statement, it returns to the main() function. In this case, we've defined the age() function, which takes one parameter, i.e., the current year.

```
int age(int current) {
    return current - BORN;
}
```

This function is called in the main function. It returns an integer to the main function.

Example Code:

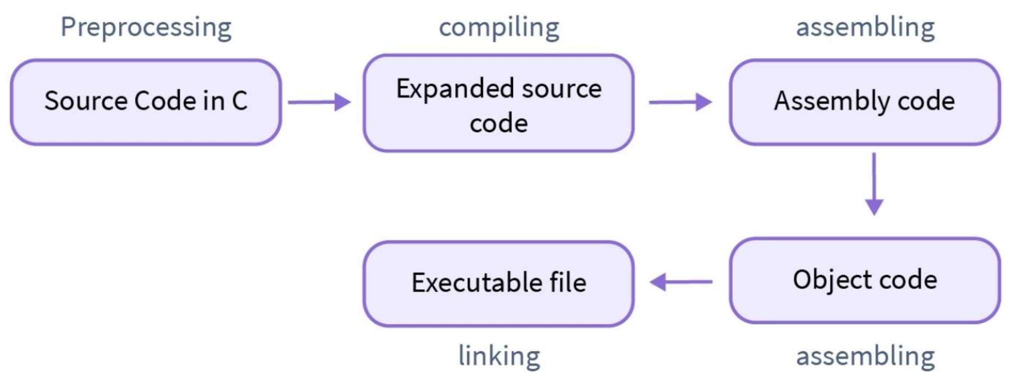
```c
/**                     //Documentation
 * file: age.c
 * author: you
 * description: program to find our age.
 */

#include <stdio.h>      //Link

#define BORN 2000       //Definition

int age(int current);   //Global Declaration

int main(void)          //Main() Function
{
  int current = 2021;
  printf("Age: %d", age(current));
  return 0;
}

int age(int current) {     //Subprograms
    return current - BORN;
}
```

Compilation Process in C :-

The compilation process in C transforms a human-readable code into a machine-readable format. For C programming language, it happens before a program starts executing to check the syntax and semantics of the code. The compilation process in C involves four steps: *pre-processing, compiling, assembling,* and *linking* then, we run the obtained executable file to get an output on the screen.

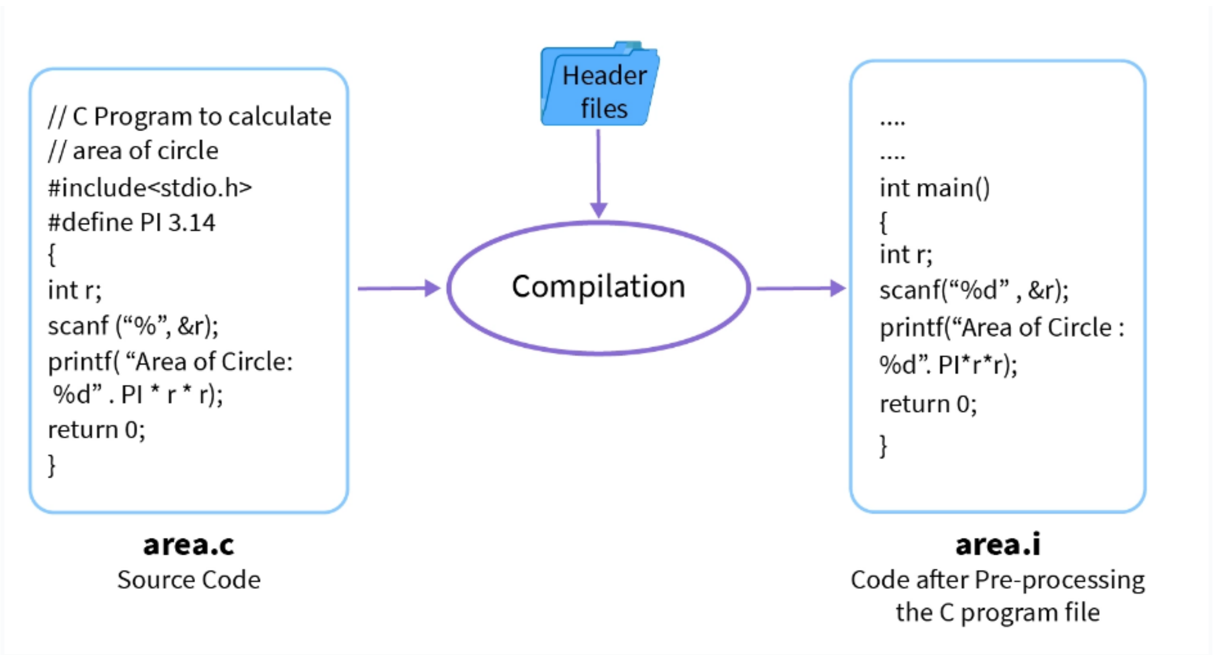
Compilation process in C involves four steps:

1. Preprocessing
2. Compiling
3. Assembling
4. Linking



a. Pre-Processing

Pre-processing is the first step in the compilation process in C performed using the *pre-processor tool* (A pre-written program invoked by the system during the compilation). All the statements starting with the # symbol in a C program are processed by the pre-processor, and it converts our program file into an intermediate file with no # statements.

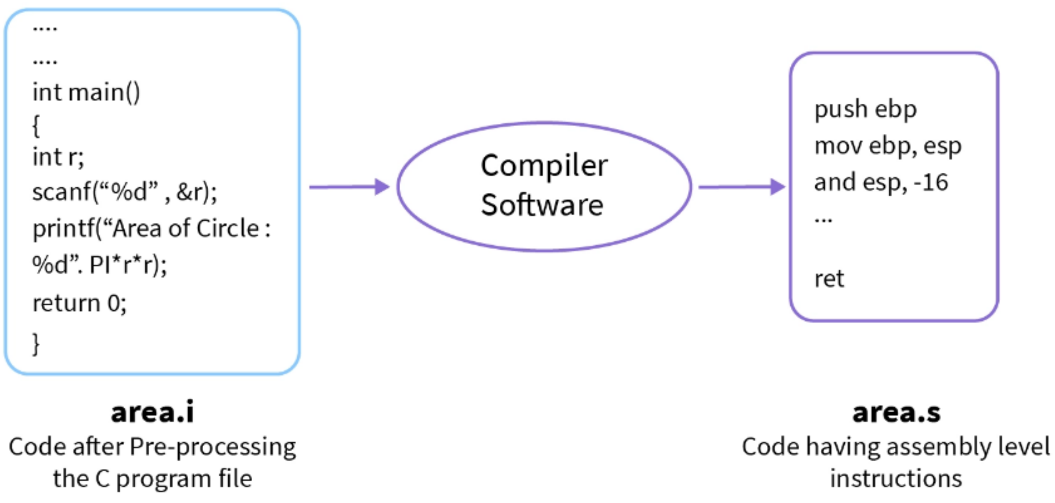


```
// C Program to calculate
// area of circle
#include<stdio.h>
#define PI 3.14
{
int r;
scanf (“%”, &r);
printf( “Area of Circle:
 %d” . PI * r * r);
return 0;
}
```

```
....
....
int main()
{
int r;
scanf(“%d” , &r);
printf(“Area of Circle :
%d”. PI*r*r);
return 0;
}
```

b. Compiling

Compiling phase in C uses an inbuilt *compiler software* to convert the intermediate (.i) file into an **Assembly file** (.s) having assembly level instructions (low-level code). To boost the performance of the program C compiler translates the intermediate file to make an assembly file.


Compiler Software
area.i
Code after Pre-processing the C program file
area.s
Code having assembly level instructions

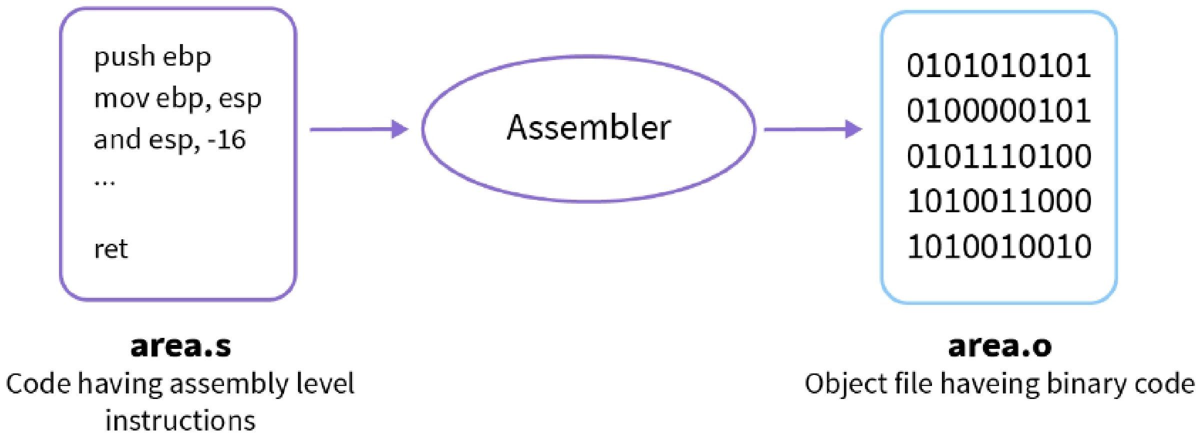
```
....
....
int main()
{
int r;
scanf(“%d” , &r);
printf(“Area of Circle :
%d”. PI*r*r);
return 0;
}
```

```
push ebp
mov ebp, esp
and esp, -16
...

ret
```

c. Assembling

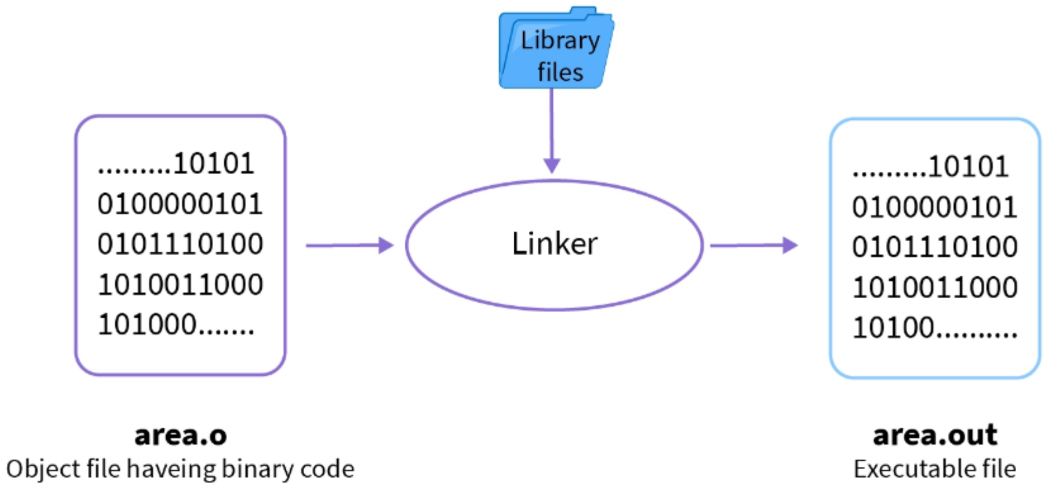
Assembly level code (.s file) is converted into a machine-understandable code (in binary/hexadecimal form) using an *assembler*. Assembler is a pre-written program that translates assembly code into machine code. It takes basic instructions from an assembly code file and converts them into binary/hexadecimal code specific to the machine type known as the object code.

The file generated has the same name as the assembly file and is known as an **object file** with an extension of **.obj** in DOS and **.o** in UNIX OS.



d. Linking

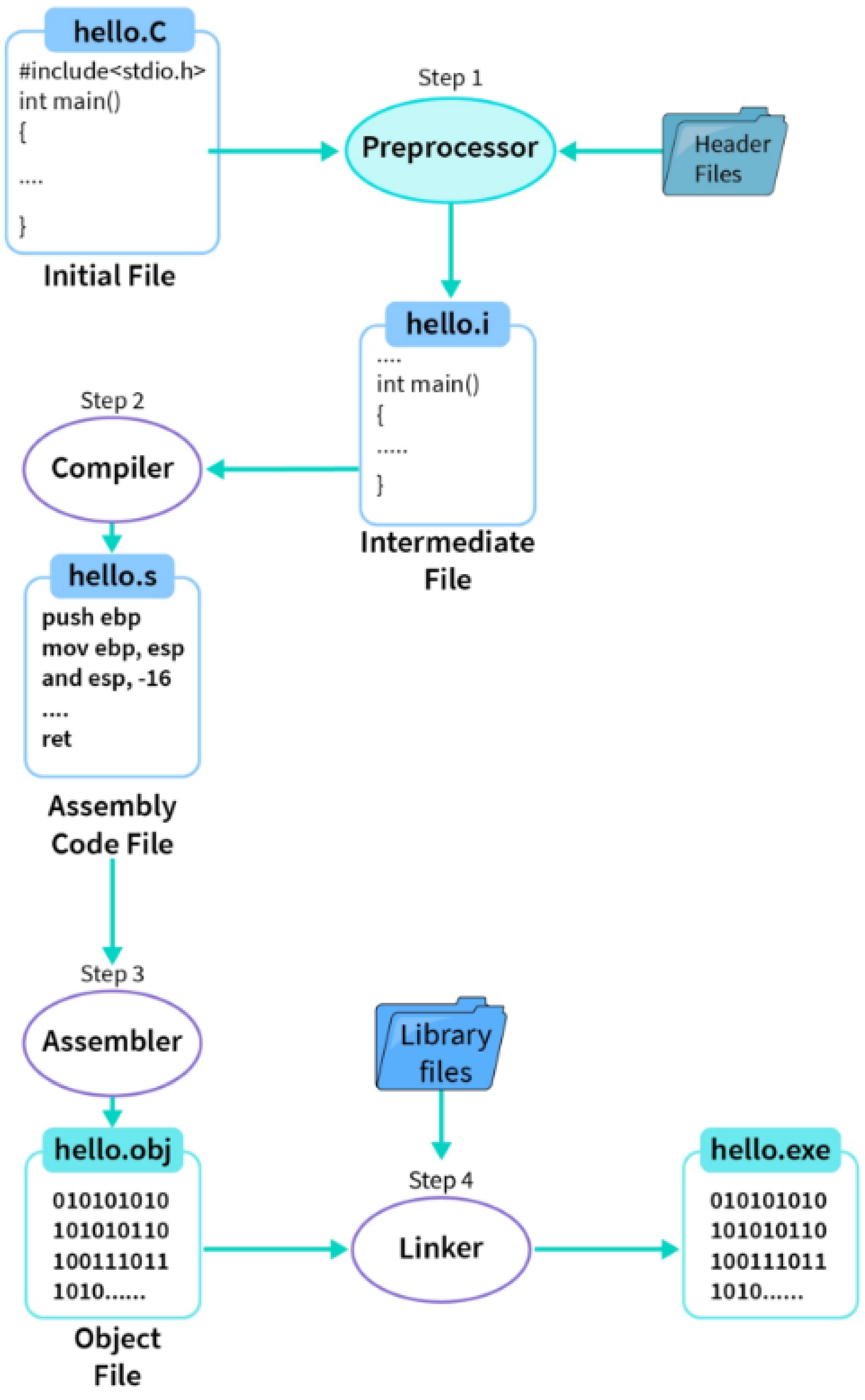
Linking is a process of including the library files into our program. *Library Files* are some predefined files that contain the definition of the functions in the machine language and these files have an extension of .lib. Some unknown statements are written in the object (.o/.obj) file that our operating system can't understand. You can understand this as a book having some words that you don't know, and you will use a dictionary to find the meaning of those words. Similarly, we use *Library Files* to give meaning to some unknown statements from our object file. The linking process generates an **executable file** with an extension of **.exe** in DOS and **.out** in UNIX OS.



Exampels :-

- We have a C Program file with an extension of .c i.e. hello.c file.
- **Step 1** is **preprocessing of header files**, all the statements starting with # (hash symbol) and comments are replaced/removed during the pre-processing with the help of a pre-processor. It generates an intermediate file with .i file extension i.e. a hello.i file.
- **Step 2** is a compilation of hello.i file. Compiler software translates the hello.i file to hello.s with assembly level instructions (low-level code).
- **Step 3**, assembly-level code instructions are converted into machine-understandable code (binary/hexadecimal form) by the assembler. The file generated is known as the object file with an extension of .obj/.o i.e. hello.obj/hello.o file.
- **Step 4**, *Linker* is used to link the library files with the object file to define the unknown statements. It generates an executable file with .exe/.out extension i.e. a hello.exe/hello.out file.
- Next, we can run the hello.exe/hello.out executable file to get the desired output on our output window, i.e., Hello World!.

Flow Diagram of the Program


hello.C
#include<stdio.h>
int main()
{
....
}
Initial File
Step 1
Preprocessor
Header Files
hello.i
....
int main()
{
.....
}
Intermediate File
Step 2
Compiler
hello.s
push ebp
mov ebp, esp
and esp, -16
....
ret
Assembly Code File
Step 3
Assembler
Library files
hello.obj
010101010
101010110
100111011
1010......
Object File
Step 4
Linker
hello.exe
010101010
101010110
100111011
1010......