CS101: Problem Solving through C Programming

Sachchida Nand Chaurasia

Assistant Professor

Department of Computer Science

Banaras Hindu University

Varanasi

Email id: snchaurasia@bhu.ac.in, sachchidanand.mca07@gmail.com

January 16, 2021

C Language Introduction I

✓ C is a procedural programming language.

✓ It was initially developed by Dennis Ritchie in the year 1972.

✓ It was mainly developed as a system programming language to write an operating system.

✓ The main features of C language include low-level access to memory, a simple set of keywords, and clean style, these features make C language suitable for system programmings like an operating system or compiler development.

C Language Introduction II

✓ Many later languages have borrowed syntax/features directly or indirectly from C language.

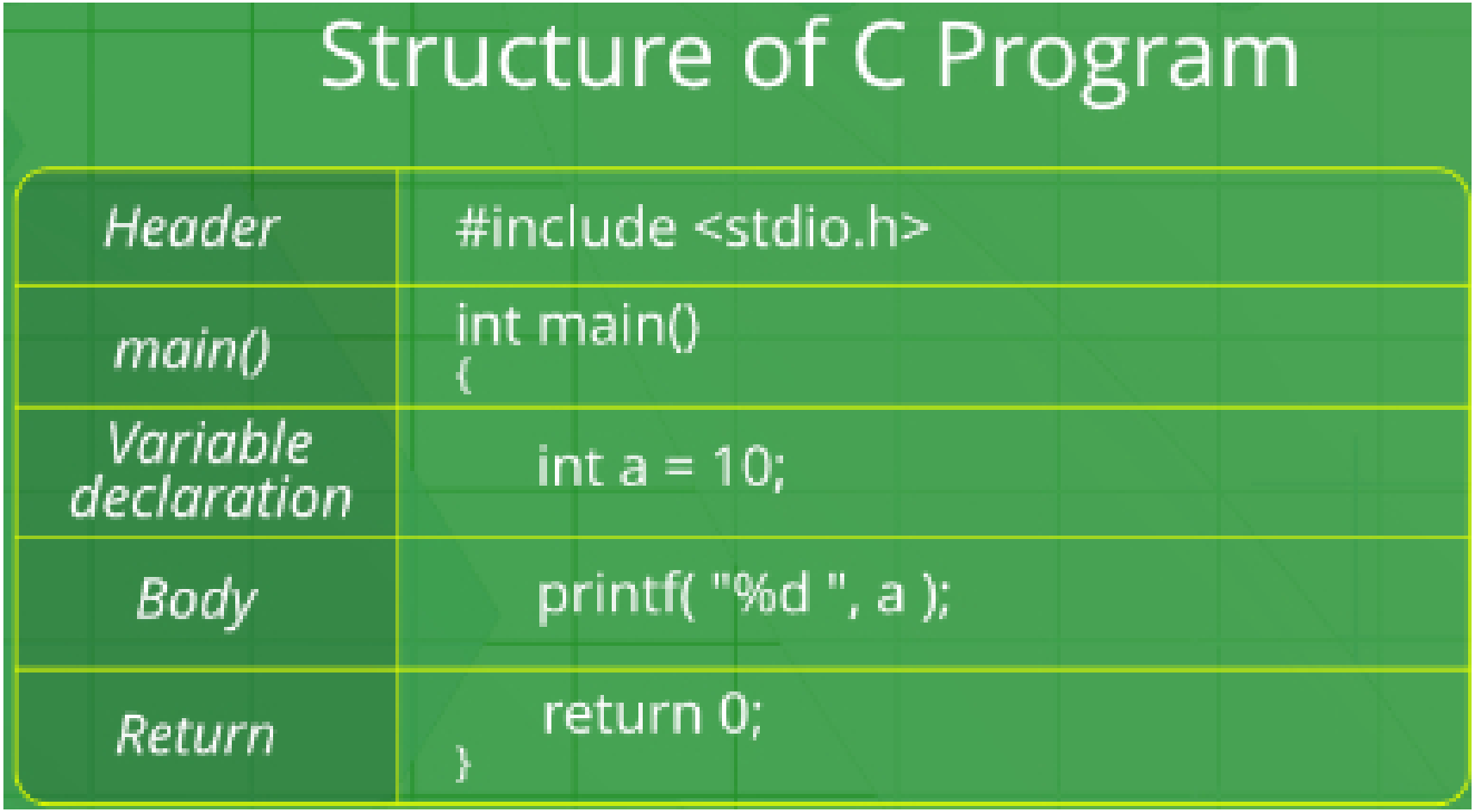
✓ Like syntax of Java, PHP, JavaScript, and many other languages are mainly based on C language.

```c
#include <stdio.h>
int main(void)
{
        /* my first program in C */
        printf("Hello, World !");
        return 0;
}
```

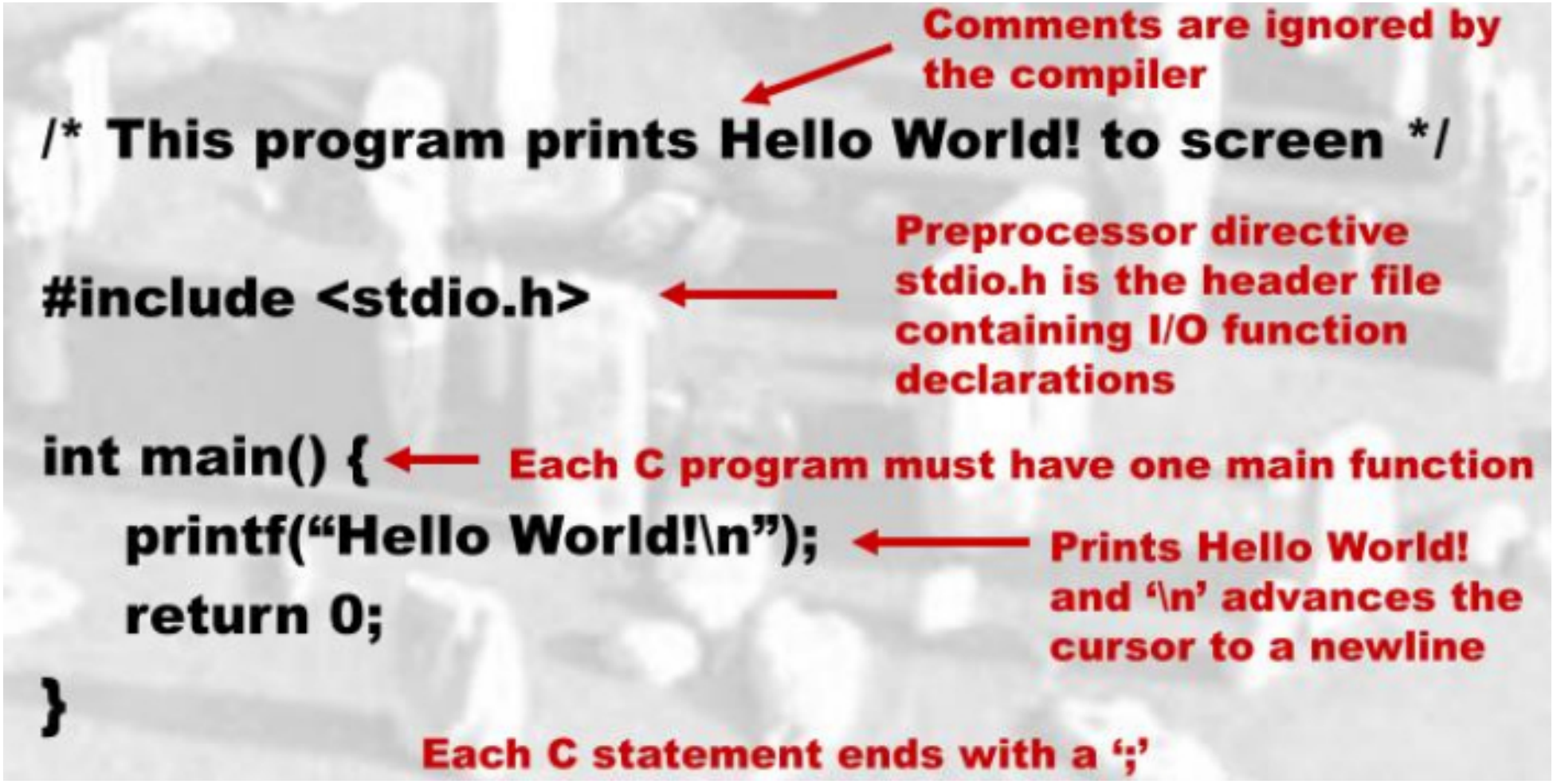
C Language Introduction III

```
1 #include<stdio.h> //include information about standard library
2 main(void) //define a Junction named main that receives no
      argument values statements oj main are enclosed in braces
3 {
4     printf("hello, world\n"); //main calls library function printf
       to print this sequence of characters; \n represents the
      newline character
5 }
```

C Language Introduction IV



C Language Introduction V



The components of the above structure are:

C Language Introduction VI

1. Header Files Inclusion: The first and foremost component is the inclusion of the Header files in a C program.

   ✓ A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.

   Some of C Header files:

   - stddef.h – Defines several useful types and macros.
   - stdint.h – Defines exact width integer types.
   - stdio.h – Defines core input and output functions

C Language Introduction VII

- stdlib.h – Defines numeric conversion functions, pseudo-random network generator, memory allocation
- string.h – Defines string handling functions
- math.h – Defines common mathematical functions

Syntax to include a header file in C:

```
1 #include
```

2. Main Method Declaration: The next part of a C program is to declare the main(void) function. The syntax to declare the main function is: Syntax to Declare main method:

C Language Introduction VIII

```
1 int main(void)
2 {
3
4 }
```

3. Variable Declaration: The next part of any C program is the variable declaration. It refers to the variables that are to be used in the function. Please note that in the C program, no variable can be used without being declared. Also in a C program, the variables are to be declared before any operation in the function.

C Language Introduction IX

Example:

```
1 int main(void)
2 {
3         int a;
4     .
5     .
6 }
```

C Language Introduction X

4. Body: Body of a function in C program, refers to the operations that are performed in the functions. It can be anything like manipulations, searching, sorting, printing, etc.

   Example:

```
1 int main(void)
2 {
3         int a;
4         printf("%d", a);
5     .
6     .
7 }
```

C Language Introduction XI

5. Return Statement: The last part in any C program is the return statement. The return statement refers to the returning of the values from a function. This return statement and return value depend upon the return type of the function. For example, if the return type is void, then there will be no return statement. In any other case, there will be a return statement and the return value will be of the type of the specified return type. Example:

C Language Introduction XII

```c
#include <stdio.h>
int main(void)
{
        printf("Hello, World !");
        return 0;
}
```

C Language Introduction XIII

Let us analyze the program line by line.

Line 1: [#include <stdio.h>]:

✓ In a C program, all lines that start with # are processed by preprocessor which is a program invoked by the compiler.

✓ In a very basic term, preprocessor takes a C program and produces another C program.

✓ The produced program has no lines starting with #, all such lines are processed by the preprocessor.

✓ In the above example, preprocessor copies the preprocessed code of stdio.h to our file.

C Language Introduction XIV

The .h files are called header files in C. These header files generally contain declaration of functions. We need stdio.h for the function printf() used in the program.

C Language Introduction XV

Line 2 [int main(void)]:

✓ There must to be starting point from where execution of compiled C program begins.

✓ In C, the execution typically begins with first line of main(void).

✓ The void written in brackets indicates that the main doesn't take any parameter.

✓ main(void) can be written to take parameters also.

✓ The int written before main indicates return type of main(void).

C Language Introduction XVI

✓ The value returned by main indicates status of program termination.

C Language Introduction XVII

Line 3 and 6: [{ and }]:

✓ In C language, a pair of curly brackets define a scope and mainly used in functions and control statements like if, else, loops.

✓ All functions must start and end with curly brackets.

C Language Introduction XVIII

Line 4 [printf("Hello, World !");]:

✓ printf() is a standard library function to print something on standard output.

✓ The semicolon at the end of printf indicates line termination.

✓ In C, semicolon is always used to indicate end of statement.

C Language Introduction XIX

Line 5 [return 0;]:

✓ The return statement returns the value from main(void).

✓ The returned value may be used by operating system to know termination status of your program.

✓ The value 0 typically means successful termination.

C Language Introduction XX

✓ The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.

✓ The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

C Language Introduction XXI

✓ The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if <stdio.h>, the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the 'stdio.h' file.

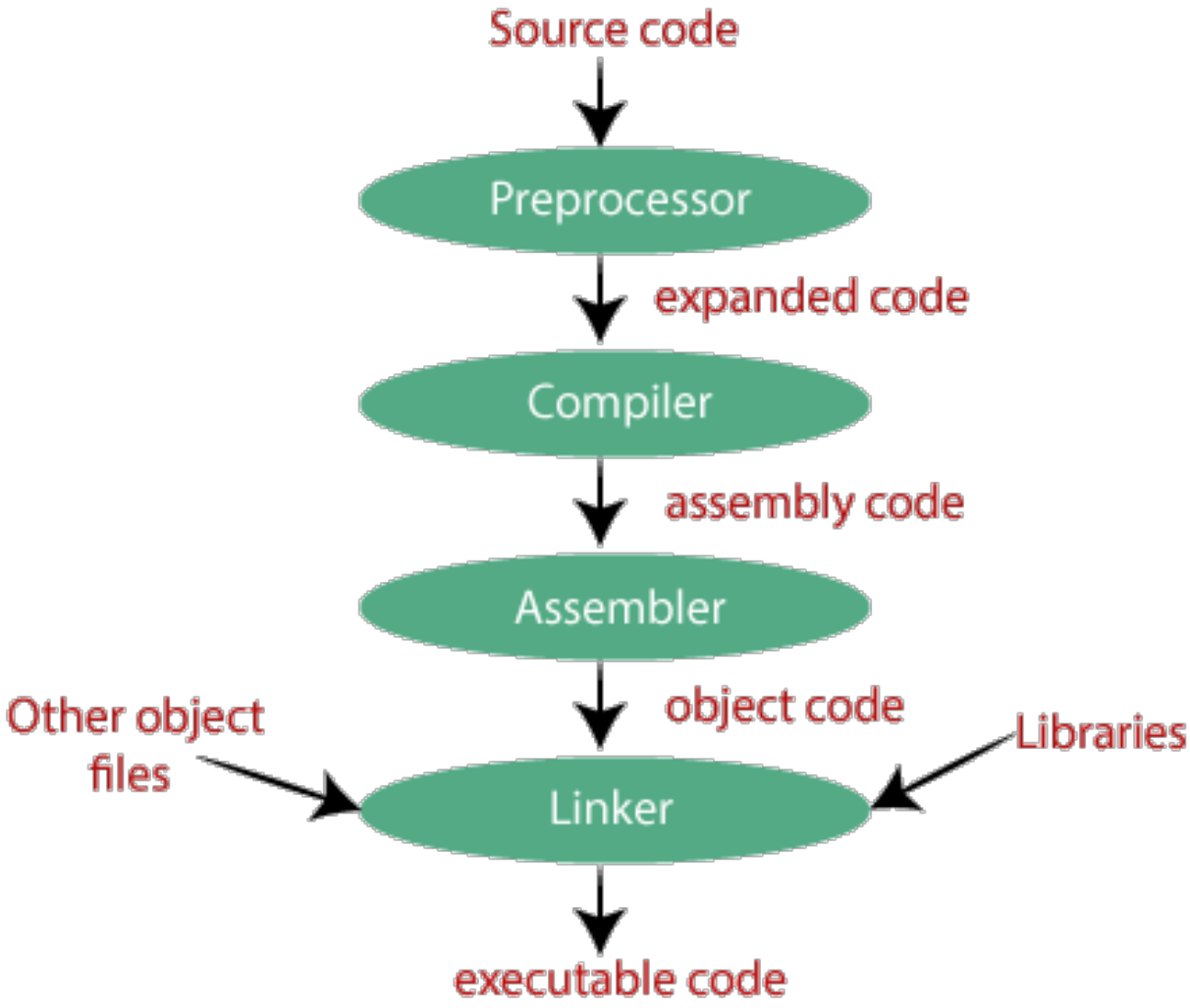
✓ The following are the phases through which our program passes before being transformed into an executable form:

1. Preprocessor

C Language Introduction XXII

2. Compiler
3. Assembler
4. Linker

C Language Introduction XXIII



Preprocessor:

✓ The source code is the code which is written in a text editor and the source code file is given an extension ".c".

✓ This source code is first passed to the preprocessor, and then the preprocessor expands this code.

✓ After expanding the code, the expanded code is passed to the compiler.

Compiler:

✓ The code which is expanded by the preprocessor is passed to the compiler.

✓ The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler:

✓ The assembly code is converted into object code by using an assembler.

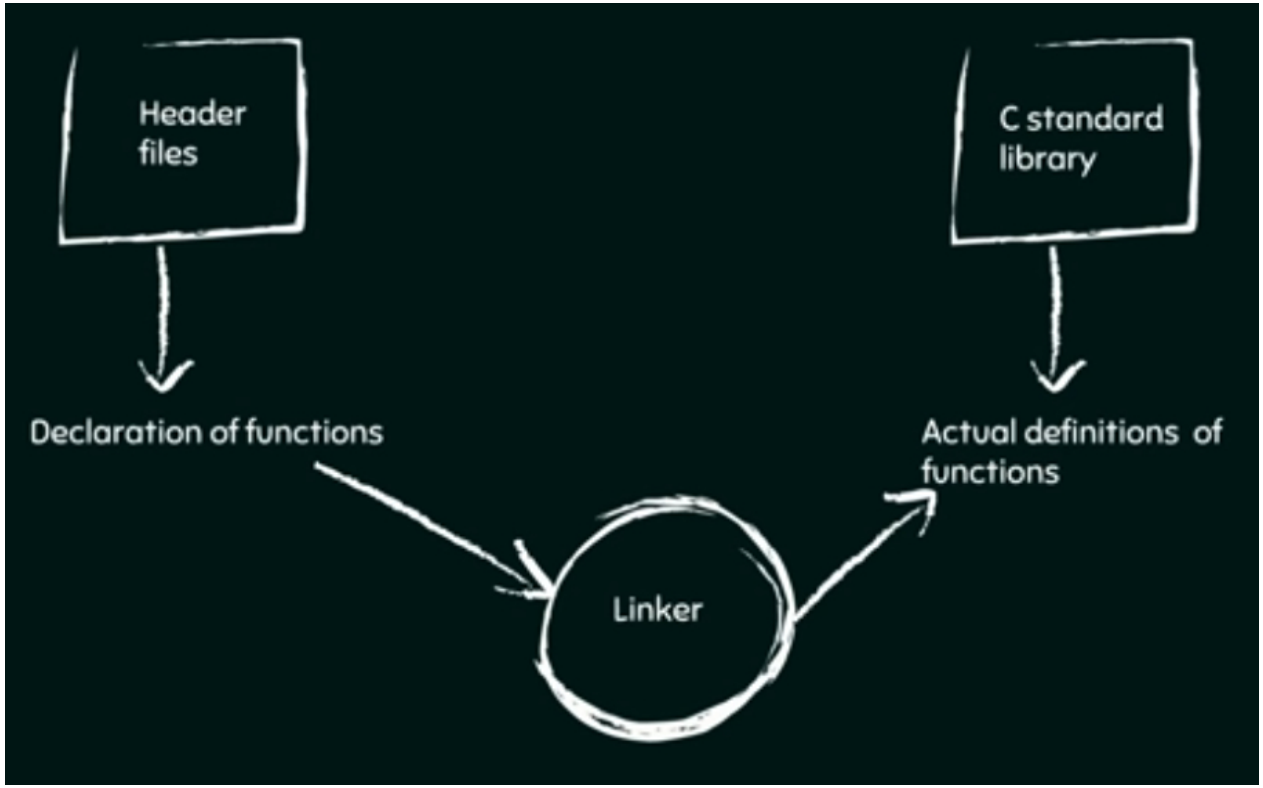
✓ The name of the object file generated by the assembler is the same as the source file.

✓ The extension of the object file in DOS is '.obj,' and in UNIX, the extension is 'o'.

✓ If the name of the source file is 'hello.c', then the name of the object file would be 'hello.obj'.

C Language Introduction XXVII

Linker:



C Language Introduction XXVIII

✓ Mainly, all the programs written in C use library functions.

✓ These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension.

✓ The main working of the linker is to combine the object code of library files with the object code of our program.

✓ Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this.

C Language Introduction XXIX

✓ It links the object code of these files to our program.

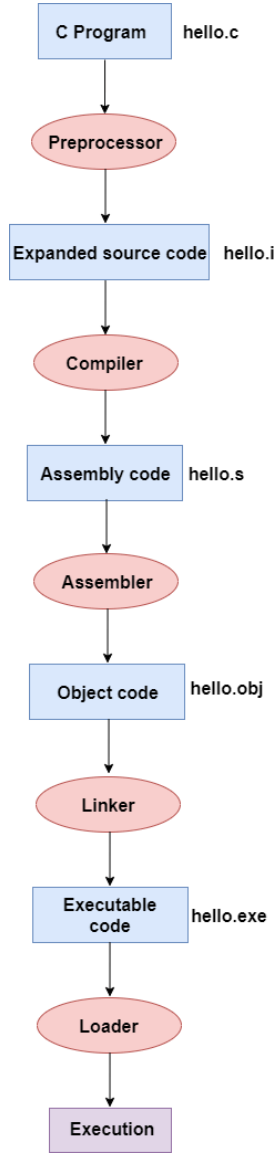
✓ Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files.

✓ The output of the linker is the executable file.

✓ In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as 'a.out'.

✓ For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.

C Language Introduction XXX



C Language Introduction XXXI

In the above flow diagram, the following steps are taken to execute a program:

✓ Firstly, the input file, i.e., hello.c, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code.

✓ The extension of the expanded source code would be hello.i.

✓ The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code.

C Language Introduction XXXII

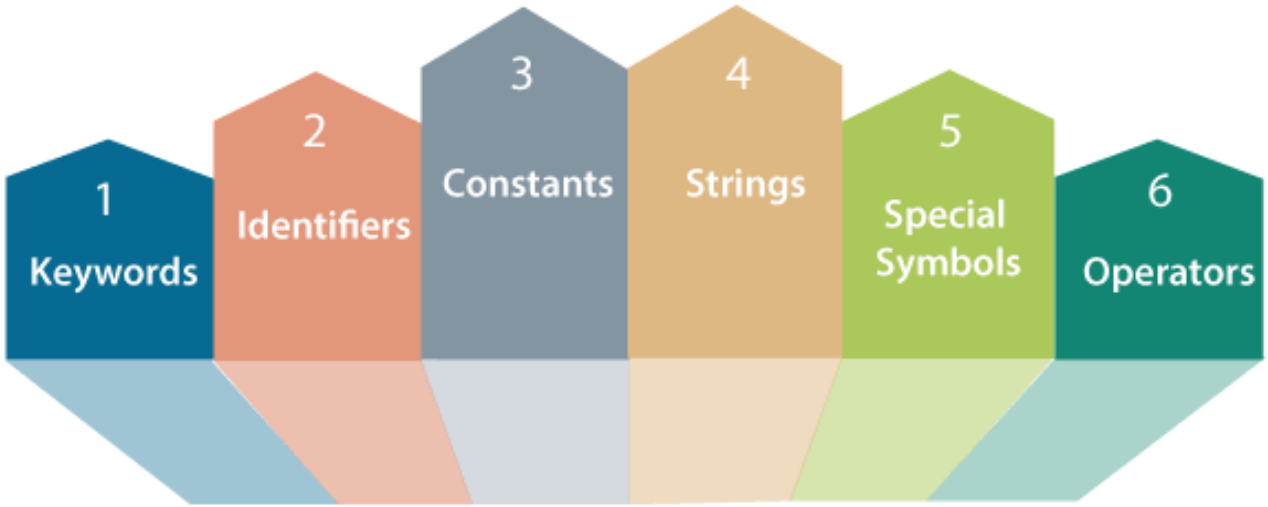
✓ The extension of the assembly code would be hello.s. This assembly code is then sent to the assembler, which converts the assembly code into object code.

✓ After the creation of an object code, the linker creates the executable file.

✓ The loader will then load the executable file for the execution.

C Tokens I

A token is the smallest element of a program that is meaningful to the compiler. Tokens can be classified as follows:



Classification of C Tokens

C Tokens II

1. Keywords
2. Identifiers
3. Constants
4. Strings
5. Special Symbols
6. Operators

C Tokens III

Keywords:

✓ Keywords are pre-defined or reserved words in a programming language.

✓ Each keyword is meant to perform a specific function in a program.

✓ Since keywords are referred names for a compiler, they can't be used as variable names because by doing so, we are trying to assign a new meaning to the keyword which is not allowed.

C Tokens IV

✓ You cannot redefine keywords. However, you can specify the text to be substituted for keywords before compilation by using C preprocessor directives.

C Tokens V

- ✓ C language supports 32 keywords which are given below:

C Tokens VI

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

Identifiers:

✓ Identifiers are used as the general terminology for the naming of variables, functions and arrays.

✓ These are user-defined names consisting of an arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character.

✓ Identifier names must differ in spelling and case from any keywords.

✓ You cannot use keywords as identifiers; they are reserved for special use.

C Tokens VIII

✓ Once declared, you can use the identifier in later program statements to refer to the associated value.

✓ A special kind of identifier, called a statement label, can be used in goto statements.

There are certain rules that should be followed while naming C identifiers:

- They must begin with a letter or underscore(_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.

C Tokens IX

- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only the first 31 characters are significant.
- main: method name.
- a: variable name.

Constants:

✓ Constants are also like normal variables. But, the only difference is, their values can not be modified by the program once they are defined.

✓ Constants refer to fixed values. They are also called literals.

✓ Constants may belong to any of the data type.

✓ const can be used to declare constant variables. Constant variables are variables which, when initialized, can't change

C Tokens XI

their value. Or in other words, the value assigned to them cannot be modified further down in the program. **Syntax:**

```
1 const data_type var_name = var_value; (or)
2 const data_type *variable_name;
```

C Tokens XII

Types of Constants:

1. Integer constants – Example: 0, 1, 1218, 12482
2. Real or Floating-point constants – Example: 0.0, 1203.03, 30486.184
3. Octal & Hexadecimal constants – Example: octal: $(013)_8 = (11)_{10}$, Hexadecimal: $(013)_{16} = (19)_{10}$
4. Character constants -Example: 'a', 'A', 'z'
5. String constants -Example: "BSCPMKSMK"

Strings:

✓ Strings are nothing but an array of characters ended with a null character ('\0'). This null character indicates the end of the string.

✓ Strings are always enclosed in double-quotes. Whereas, a character is enclosed in single quotes in C and C++.

Declarations for String:

C Tokens XIV

```
1 char string[20] =
2 {
3     'g', 'e', 'e', 'k', 's', 'f', 'o', 'r', 'g', 'e', 'e', 'k', 's
    ', '\0'
4 }
5 ;
6 char string[20] = "BSCPMKSMK";
7 char string [] = "BSCPMKSMK";
```

C Tokens XV

✓ when we declare char as "string[20]", 20 bytes of memory space is allocated for holding the string value.

✓ When we declare char as "string[]", memory space will be allocated as per the requirement during the execution of the program.

Special Symbols:

The following special symbols are used in C having some special meaning and thus, cannot be used for some other purpose.[] () { }, ; * = #

- **Brackets[]:** Opening and closing brackets are used as array element reference. These indicate single and multidimensional subscripts.
- **Parentheses():** These special symbols are used to indicate function calls and function parameters.

C Tokens XVII

- **Braces:** These opening and ending curly braces mark the start and end of a block of code containing more than one executable statement.
- **Comma (,):** It is used to separate more than one statements like for separating parameters in function calls.
- **Colon(:):** It is an operator that essentially invokes something called an initialization list.

C Tokens XVIII

- **Semicolon(;):** It is known as a statement terminator. It indicates the end of one logical entity. That's why each individual statement must be ended with a semicolon.
- **Asterisk (*):** It is used to create a pointer variable.
- **Assignment operator(=):** It is used to assign values.
- **Pre-processor (#):** The preprocessor is a macro processor that is used automatically by the compiler to transform your program before actual compilation.

Operators:

✓ Operators are symbols that trigger an action when applied to C variables and other objects.

✓ The data items on which operators act upon are called operands.

✓ Depending on the number of operands that an operator can act upon, operators can be classified as follows:

C Tokens XX

- Unary Operators: Those operators that require only a single operand to act upon are known as unary operators.For Example increment and decrement operators.
- Binary Operators: Those operators that require two operands to act upon are called binary operators. Binary operators are classified into :
  1. Arithmetic Operators
  2. Assignment Operators
  3. Relational Operators
  4. Logical Operators

C Tokens XXI

5. Unary Operators
6. Bitwise Operators

C Tokens XXII

1. Arithmetic Operators: Arithmetic operators are used to perform arithmetic operations on variables and data.

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo Operation (Remainder after division)

Example:

```
1 a + b; a - b;  a * b;  a / b;  a % b;
```

C Tokens XXIII

❷ Assignment Operators: Assignment operators are used in C to assign values to variables.

Operator	Example	Equivalent to
=	a = b;	a = b;
+=	a += b;	a = a + b;
-=	a -= b;	a = a - b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

C Tokens XXIV

3. Relational Operators: Relational operators are used to check the relationship between two operands.

```
// check is a is less than b
a < b; // Here, < operator is the relational operator. It
    checks if a is less than b or not. It returns either true
     or false.
```

Operator	Description	Example
==	Is Equal To	3 == 5 returns **False**
!=	Not Equal To	3 != 5 returns **True**
>	Greater Than	3 > 5 returns **False**
<	Less Than	3 < 5 returns **True**
>=	Greater Than or Equal To	3 >= 5 returns **False**
<=	Less Than or Equal To	3 <= 5 returns **True**

C Tokens XXV

4. Logical Operators: Logical operators are used to check whether an expression is **True** or **False**. They are used in decision making.

Operator	Example	Meaning
&& (Logical AND)	expression1 && expression2	**True** only if both expression1 and expression2 are **True**
|| (Logical OR)	expression1 || expression2	**True** if either expression1 or expression2 is **True**
! (Logical NOT)	!expression	**True** if expression is **False** and vice versa

C Tokens XXVI

```
1  int num1=12;
2  int num2=15;
3  int num3=0;
4  int val;
5  val = num1 && num2;
6  val = num1 || num2;
7  val = !num1;
8  val = !num3;
9  ------------------------
10 Output: ????
```

C Tokens XXVII

5. Unary Operators: Unary operators are used with only one operand.

   For example, ++ is a unary operator that increases the value of a variable by 1. That is, ++5 will return 6.

Operator	Meaning
+	Unary plus: not necessary to use since numbers are positive without using it
-	Unary minus: inverts the sign of an expression
++	Increment operator: increments value by 1
- -	Decrement operator: decrements value by 1
!	Logical complement operator: inverts the value of a boolean

C Tokens XXVIII

```
1  int num = 5;
2  // increase num by 1
3  ++num;
4  // decrement num by 1
5  --num;
6  // + as unary operator
7  num = +5;
8  // - as unary operator
9  num = -55;
10 // ! as logical complement
11 num = !num;
12 ------------------------
13 Output: ????
```

C Tokens XXIX

C Tokens XXX

6. Bitwise Operators: Bitwise operators in C are used to perform operations on individual bits.

```
Bitwise complement Operation of 35
35 = 00100011 (In Binary)
~ 00100011
  ________
   11011100  = 220 (In decimal)
Here, ~ is a bitwise operator. It inverts the value of each
    bit (0 to 1 and 1 to 0).
```

C Tokens XXXI

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive XOR
~	Bitwise Complement
«	Left Shift
»	Right Shift

C Tokens XXXII

```c
unsigned int a = 60;    /* 60 = 0011 1100 */
unsigned int b = 13;  /* 13 = 0000 1101 */
int c = 0;
// & bitwise AND
c = a & b;  /* 12 = 0000 1100 */
// | bitwise OR
c = a | b;  /* 61 = 0011 1101 */
// ^ bitwise XOR
c = a ^ b;  /* 49 = 0011 0001 */
// ~ bitwise complement
c = ~a; /*-61 = 1100 0011 */
// << left shift
c = a << 2; /* 240 = 1111 0000 */
```

C Tokens XXXIII

```
14 // >> right shift
15 c = a >> 2; /* 15 = 0000 1111 */
```

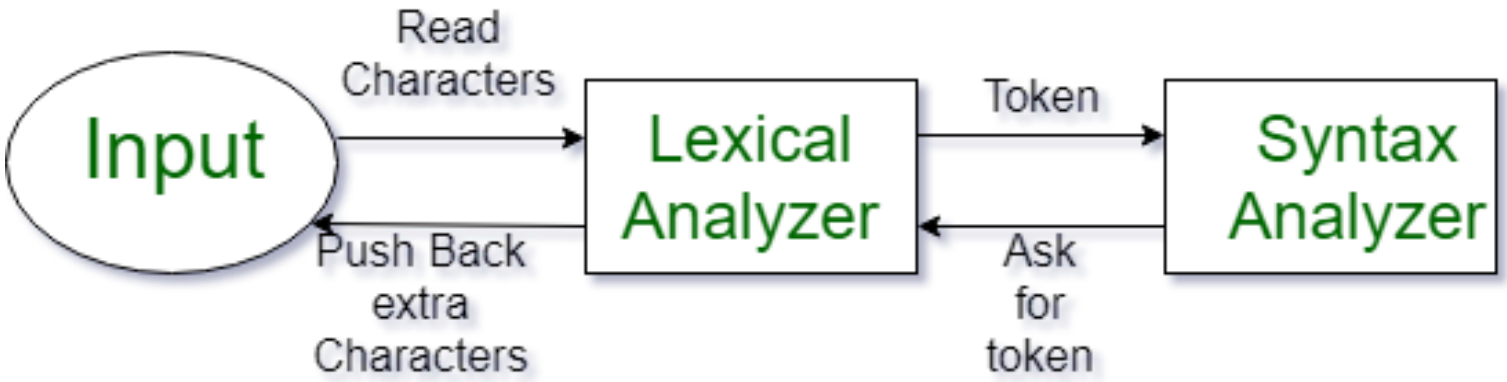
Introduction of Lexical Analyzer I

✓ Lexical Analysis is the first phase of the compiler also known as a scanner. It converts the High level input program into a sequence of Tokens.

✓ The lexical analyzer is the part of the compiler that detects the token of the program and sends it to the syntax analyzer.

✓ Token is the smallest entity of the code, it is either a keyword, identifier, constant, string literal, symbol. Examples of different types of tokens in C.

Introduction of Lexical Analyzer II



What is a token?

A lexical token is a sequence of characters that can be treated as a unit in the grammar of the programming languages.

Example of tokens:

Introduction of Lexical Analyzer III

- Type token (id, number, real, . . .)
- Punctuation tokens (IF, void, return, . . .)
- Alphabetic tokens (keywords)
- Keywords; Examples-for, while, if etc.
- Identifier; Examples-Variable name, function name, etc.
- Operators; Examples '+', '++', '-' etc.
- Separators; Examples ',' ';' etc

Example of Non-Tokens:

Introduction of Lexical Analyzer IV

```
Comments, preprocessor directive, macros, blanks, tabs, newline,
    etc.
```

Lexeme: The sequence of characters matched by a pattern to form the corresponding token or a sequence of input characters that comprises a single token is called a lexeme.

eg- “float”, “abs_zero_Kelvin”, “=”, “-”, “273”, “;” .

How Lexical Analyzer functions

1. It always matches the longest character sequence.

Introduction of Lexical Analyzer V

2. Tokenization i.e. Dividing the program into valid tokens.
3. Remove white space characters.
4. Remove comments.
5. It also provides help in generating error messages by providing row numbers and column numbers.

Introduction of Lexical Analyzer VI

✓ The lexical analyzer(scanner) identifies the error with the help of the automation machine and the grammar of the given language on which it is based like C, C++, and gives row number and column number of the error. Suppose we pass a statement through lexical analyzer–

```
1 a = b + c ;  //It will generate token sequence like this:
2 id= id + id;  //Where each id refers to it's variable in the
    symbol table referencing all details
```

For example, consider the program:

Introduction of Lexical Analyzer VII

```
int a=5; // int a = 5 ; just for understanding.
Tokens:
|int| |a| |=| |5| |; |
```

Another example:

```
int main()
{
      // 2 variables
      int a, b;
      a = 10;
     return 0;
}
-------------------------
```

Introduction of Lexical Analyzer VIII

```
Tokens:
'int'
'main'
'('
')'
'
{
    '
    'int'
    'a'
    ','
    'b'
    '; '
```

Introduction of Lexical Analyzer IX

```
22      'a'
23      '='
24      '10'
25      '; '
26      'return'
27      '0'
28      '; '
29      '
30  }
31  '
```

```
1  //How many tokens?
2
3  printf("BSCPMKSMK");
```

Introduction of Lexical Analyzer X

```
int main()
{
      int a = 10, b = 20;
      printf("sum is :%d",a+b);
      return 0;
}
----------
//How many tokens?
```

```
int max(int i);
--------------------
//Count number of tokens :
```

Introduction of Lexical Analyzer XI

- Lexical analyzer first read int and finds it to be valid and accepts as token
- max is read by it and found to be a valid function name after reading (
- int is also a token , then again i as another token and finally ;

Answer: Total number of tokens 7:

```
1 |int| |max| |(| |int| |i| |)| |; |
```

Introduction of Lexical Analyzer XII

```
1  //Count number of tokens :
2
3 printf("i = %d, &i = %x", i, &i);
```