Generations of Different Computer Language

DR. RAHUL DAS GUPTA

1. The first **generation languages**, or **1GL**, are **low-level languages** that are **machine language**.

Machine language is a collection of **binary-digits** or bits that the computer reads and interprets. Machine language is the only language a hardware/digital circuit is capable of understanding.

2. The **second-generation languages**, or **2GL**, are also low-level **assembly languages**. They are sometimes used in kernels and hardware drives, but more commonly used for video editing and video games.

Assembly language uses a **mnemonic** to represent each low-level machine instruction or **opcode**.

An **assembly (or assembler) language**, often abbreviated asm, is any low-level programming language in which there is a very strong correspondence between the program's statements and the architecture's machine code instructions.

3. The **third-generation languages**, or **3GL**, are **high-level languages**, such as **C**, **C++**, **Java**, **JavaScript**, and **Visual Basic**.

4. The **fourth-generation languages**, or **4GL**, are languages that consist of statements similar to statements in a human language. Fourth generation languages are commonly used in database programming and scripts examples include **Perl**, **PHP**, **Python**, **Ruby**, and **SQL**.

5. The **fifth-generation languages**, or **5GL**, are programming languages that contain visual tools to help develop a program. Examples of fifth generation languages include **Mercury**, **OPS5**, and **Prolog**.

C – Language History

- The C programming language is a structure oriented programming language, developed at Bell Laboratories in 1972 by Dennis Ritchie.
- C programming language features were derived from an earlier language called "B" (Basic Combined Programming Language – BCPL).
- C language was invented for implementing UNIX operating system.
- In 1978, Dennis Ritchie and Brian Kernighan published the first edition "The C Programming Language" and commonly known as K&R C.
- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The

resulting definition, the ANSI standard, or “ANSI C”, was completed late 1988.

WHICH LEVEL IS C LANGUAGE BELONGING TO?

There are 3 levels of programming languages. They are,

1. **Middle Level Languages:**
   Middle level languages don’t provide all the built-in functions found in high level languages, but provides all building blocks that we need to produce the result we want.

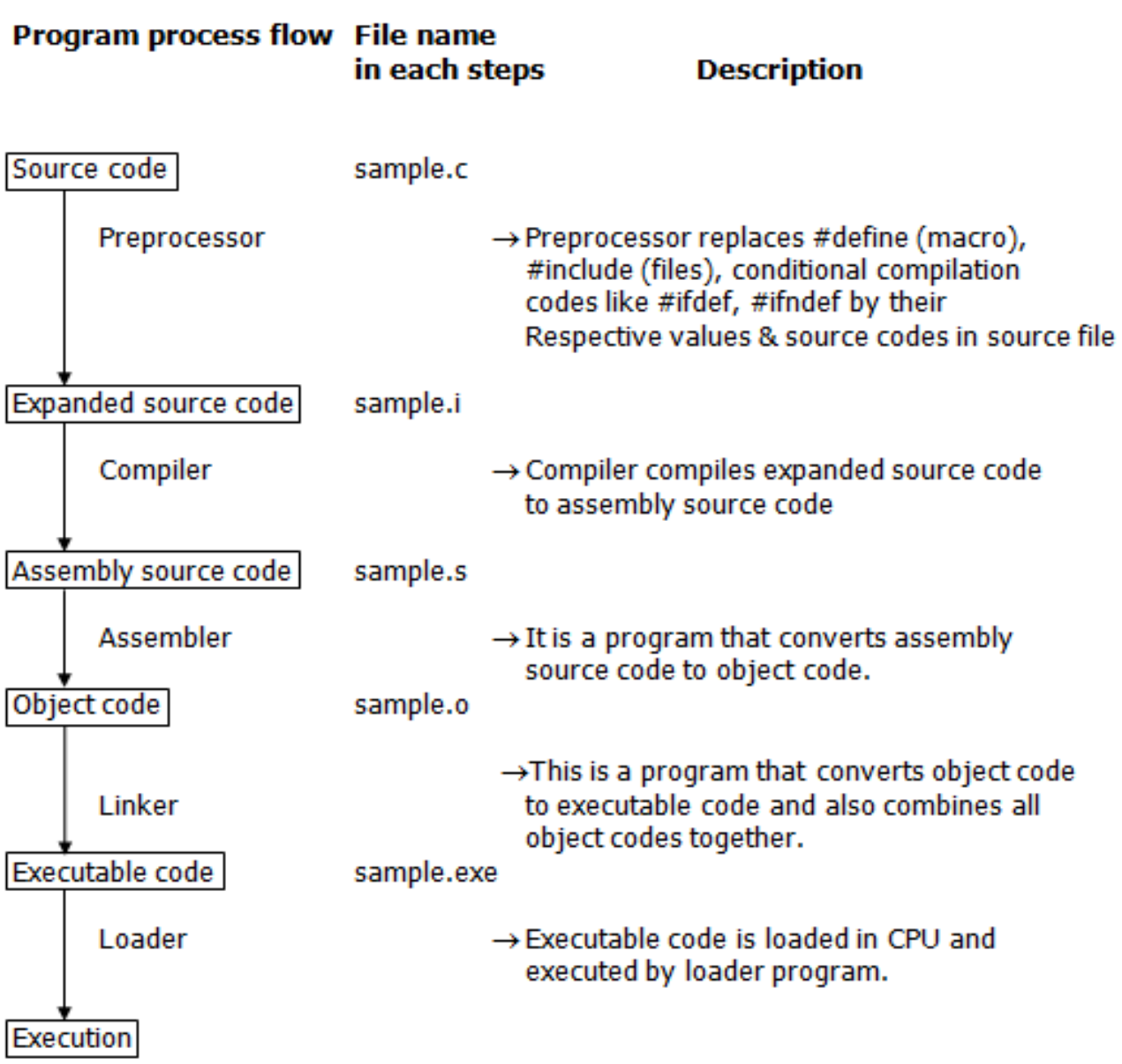
   Examples: C, C++

2. **High Level Languages:**
   High level languages provide almost everything that the programmer might need to do as already built into the language.

   Example: Java, Python

3. **Low Level Languages:**
   Low level languages provides nothing other than access to the machines basic instruction set.

   Example: Assembler



C PREPROCESSOR DIRECTIVES:

- Before a C program is compiled in a compiler, source code is processed by a program called preprocessor. This process is called preprocessing.
- Commands used in preprocessor are called preprocessor directives and they begin with "#" symbol.

Memory Region Involved in C Language:

There are 4 regions of memory which are created by a compiled C program. They are,

1. **First region** – This is the memory region which holds the executable code of the program.
2. **2nd region** – In this memory region, global variables are stored.
3. **3rd region** – stack
4. **4th region** – heap

Stack & Heap Memory in C Language

Stack

Stack is a memory region where “local variables”, “return addresses of function calls” and “arguments to functions” are hold while C program is executed.

CPU’s current state is saved in stack memory.

Heap

Heap is a memory region which is used by dynamic memory allocation functions at run time.

Linked list is an example which uses heap memory.

DESCRIPTION FOR EACH SECTION OF THE C PROGRAM:

Sections	Description
Documentation section	We can give comments about the program, creation or modified date, author name etc in this section. The characters or words or anything which are given between “/*” and “*/”, won’t be considered by C compiler for compilation process.These will be ignored by C compiler during compilation. Example : /* comment line1 comment line2 comment 3 */

Link Section	Header files that are required to execute a C program are included in this section
Definition Section	In this section, variables are defined and values are set to these variables.
Global declaration section	Global variables are defined in this section. When a variable is to be used throughout the program, can be defined in this section.
Function prototype declaration section	Function prototype gives many information about a function like return type, parameter names used inside the function.
Main Function	Every C program is started from main function and this function contains two major sections called declaration section and executable section.
User defined function section	User can define their own functions in this section which perform particular task as per the user requirement.

Key Points to Remember in Writing C Programs:

1. C programming is a case sensitive programming language. **Most of the keywords of C-Language are written in lower case (except NULL, FILE).**

   Name of the **identifier** (**not keyword**) may contain upper case.

   Examples:

   int i,N; /* Here N is an **identifier** (**not keyword**) */

   char Name[20]; /* Here Name is an **identifier** (**not keyword**) */

   FILE *fp;

   int *p=NULL;

2. **Each C programming statement is ended with semicolon (;)** which are referred as **statement terminator**.

   Examples:

   printf(“ Ban de mataram.”);

printf(“ Long live revolution.”);

3. **Do not put semicolon (;)**

(i) after any pre-processing statement

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

#include<stdio.h>; /*It is Wrong*/
#include<stdio.h> /*It is Correct*/

#include”math.h”; /*It is Wrong*/
#include”math.h” /* It is Correct */

#define MAX 10; /*It is Wrong*/
#define MAX 10 /* It is Correct */]

(ii) after any if conditional statement

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

if (n==0); /*It is Wrong*/
if (n==0)/*It is Correct*/

if (n>=2 && n<=5); /*It is Wrong*/
if (n>=2 && n<=5)/* It is Correct */]

(iii) after any else if conditional statement

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
else if (n==0); /*It is Wrong*/
else if (n==0)/*It is Correct*/

else if (n>=2 && n<=5); /*It is Wrong*/
else if (n>=2 && n<=5)/* It is Correct */]
```

(iv) after any else conditional statement

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
else; /*It is Wrong*/
else /*It is Correct*/]
```

(v) immediately after any for loop

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
for( i=0; i<100; i++ ); /*It is Wrong*/
for(i=0; i<100; i++ ) /*It is Correct*/]
```

(vi) immediately after any while loop (except do-while loop)

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
while (i<100); /*It is Wrong*/
while (i<100) /*It is Correct*/]
```

(vii) immediately after any do statement in do-while loop

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
do; /*It is Wrong*/
do /*It is Correct*/]
```

(viii) immediately after any function input argument type declaration in function definition

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
float area_circle( float r); /*It is Wrong*/
{
  return (22/7*r*r);
}
```

```
float area_circle( float r)/*It is Correct*/
{
  return (22/7*r*r);
}]
```

4. **It is important to remember that do-while loop must be terminated with a semicolon (;).**

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
int sum_one_to_hundred( )
{
   int S=0,n=1;
     do
     {
       S=S + n;
       n ++;
      }while(n<100) /*It is Wrong. ';' is missing*/
   return (S);
}

int sum_one_to_hundred( )
{

    int S=0,n=1;
    do
    {
```

```
        S=S + n;
        n ++;
     }while(n<100); /*It is Correct*/
  return (S);
}
]
```

Function prototype declaration must be terminated with a semicolon (;).

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
    float area_circle( float ) /*It is Wrong. */
    float area_circle( float ); /*It is Correct. */
]
```

5. **A block containing multiple programming statements** (at least two) under **if**, **else if**, **else**, **switch**, **while**, **for**, **do** must contained within an **opening and closing curly braces (that is between { and }).**

Example:

```
int  sum_of_digits (int n)
{
   int  i, d;S=0;
   for (i=n; i>0; i=i/10)
   { /*Beginning of the Block*/
```

```
        d=i%10;
        S=S+d;
    } /*Ending of the Block*/

    return (S);
}
```

6. There must be two semicolons (;) associated with **for loop.**

[Examples of some common mistakes:
Mistakes are highlighted by **red colour**

```
for(i=0; i<25, i++)/*It is Wrong. */
for(i=0, i<25; i++)/*It is Wrong. */
for(i=0, i<25, i++)/*It is Wrong. */
for(i=0; i<25; i++)/*It is Correct. */
```

]