Why Did You Choose CSE?

Introduce With Variables, Constants, Literals And Language Processors

Course Title :- Structured Programming Language Sessional

Course Code :- CSE-122

Level Term: 1-II-A(G1) & 1-II-B(G3,G4)

Khandaker Jannatul Ritu, Lecturer(CSE), BAIUST

Resources will be followed:

Books:

1. Tech yourself c
2. C: the complete reference
3. Programming in ANSI-C
4. C programming by tamim shahriar subin

Websites:

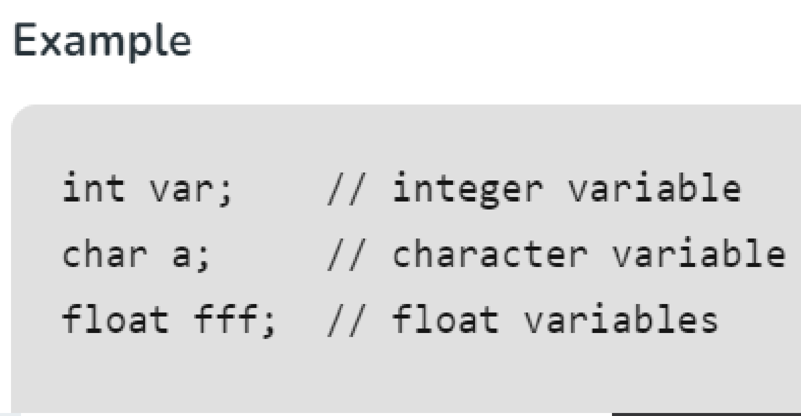
1. https://www.w3schools.com/c/index.php
2. https://www.programiz.com/c-programming
3. https://www.javatpoint.com/c-programming-language-tutorial
4. https://www.geeksforgeeks.org/c-programming-language/
5. https://www.tutorialspoint.com/cprogramming/index.htm

Outlines:-

- C Variables(Syntax, Components, Rules)
- C Variable Types
- Explain how Variable is a changeable entity?
- How to copy variables?
- Add Variables Together
- Real-Life Example of variable
- Calculate the Area of a Rectangle
- Constants in C
- Literals
- Language Processors: Assembler, Compiler and Interpreter

C Variables

- Variables are containers for storing data values, like numbers and characters.
- **In C, there are different types of variables (defined with different keywords), for example:**
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float/dobule - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Characters are surrounded by single quotes
- *A **variable in C** is a memory location with some name that helps store some form of data and retrieves it when required. We can store different types of data in the variable and reuse the same variable for storing some other data any number of times.*
- The syntax to declare a variable in C specifies the name and the type of the variable.
- Here,
- **data_type:** Type of data that a variable can store.
- **variable_name:** Name of the variable given by the user.
- **value:** value assigned to the variable by the user.

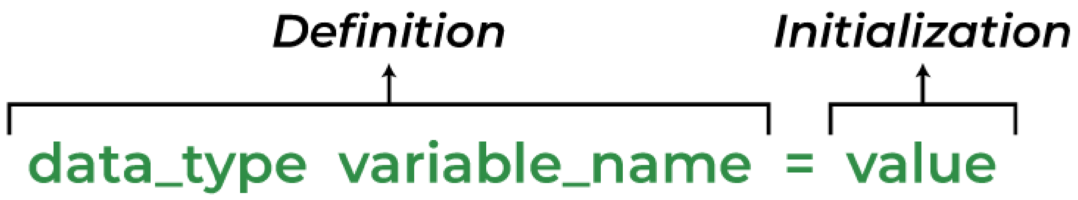

Example

```
int var;    // integer variable
char a;     // character variable
float fff;  // float variables
```

```
data_type variable_name = value;
or
data_type variable_name1, variable_name2;
```

The Three Components Of Declaring A Variable

- **There are 3 aspects of defining a variable:**
- **1. C Variable Declaration**
- Variable declaration in C tells the compiler about the existence of the variable with the given name and data type. When the variable is declared compiler automatically allocates the memory for it.
- **2. C Variable Definition**
- In the definition of a C variable, the compiler allocates some memory and some value to it. A defined variable will contain some random garbage value till it is not initialized.
- **3. C Variable Initialization**
- Initialization of a variable is the process where the user assigns some meaningful value to the variable.



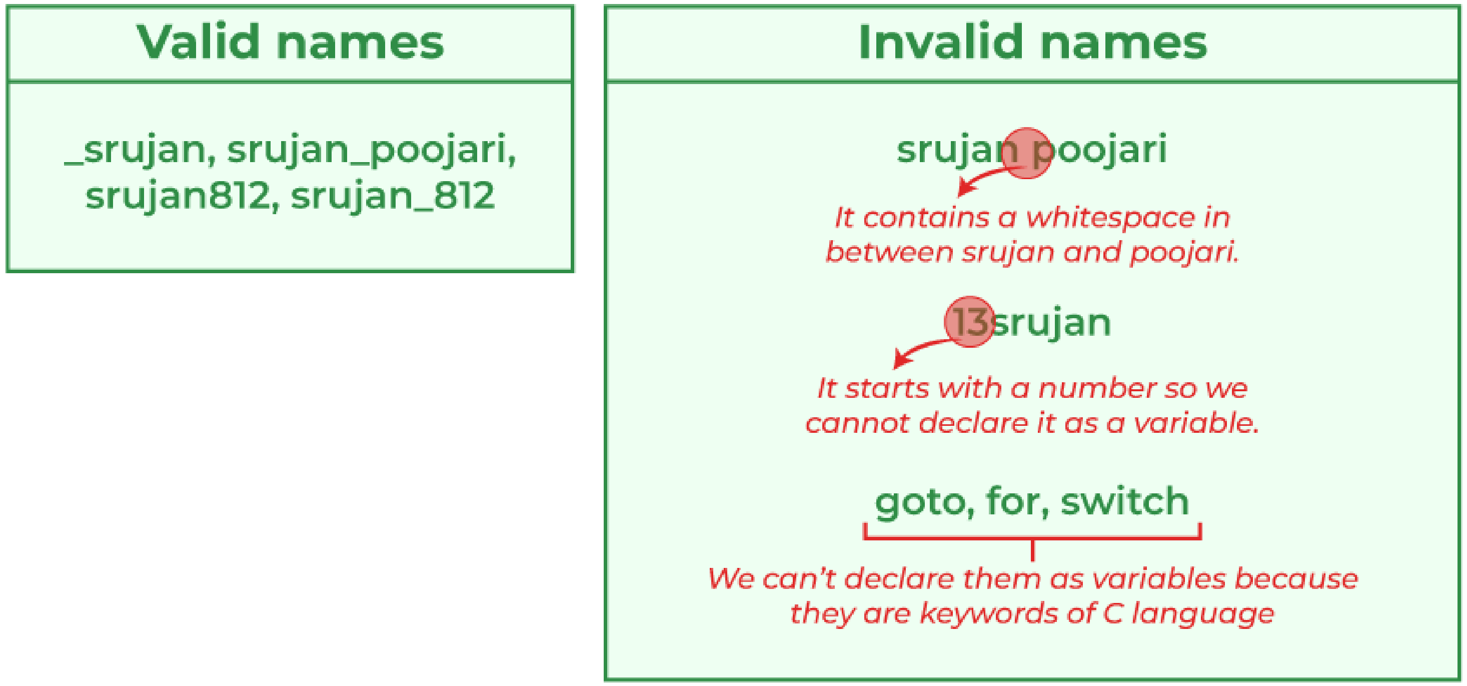
Example

```
int var;
char var2;
```

```
int var; // variable definition
var = 10; // initialization
      or
int var = 10; // variable declaration and definition
```

Rules for Naming Variables in C

- **You can assign any name to the variable as long as it follows the following rules:**
- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case-sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (such as int) cannot be used as names



Valid names

_srujan, srujan_poojari, srujan812, srujan_812

Invalid names

srujan poojari

It contains a whitespace in between srujan and poojari.

13srujan

It starts with a number so we cannot declare it as a variable.

goto, for, switch

We can't declare them as variables because they are keywords of C language

C Variable Types

1. Local Variable

A variable that is declared inside the function or block is called a local variable.
It must be declared at the start of the block.

```
void function1()
{
	int x=10;//local variable
}
```

2. Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.
It must be declared at the start of the block.

```
int value=20;//global variable
void function1()
{
	int x=10;//local variable
}
```

3. Static Variable

A variable that is declared with the static keyword is called static variable. It retains its value between multiple function calls.

```
void function1()
{
	int x=10;//local variable
	static int y=10;//static variable
	x=x+1;
	y=y+1;
	printf("%d,%d",x,y);
}
```

4. Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```
void main()
{
	int x=10;//local variable (also automatic)
	auto int y=20;//automatic variable
}
```

5. External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

```
extern int x=10;//external variable (also global)
```

6. Register Variables in C

Register variables in C are those variables that are stored in the **CPU register** instead of the conventional storage place like RAM. Their scope is **local** and exists till the **end** of the **block** or a function.

These variables are declared using the register keyword.

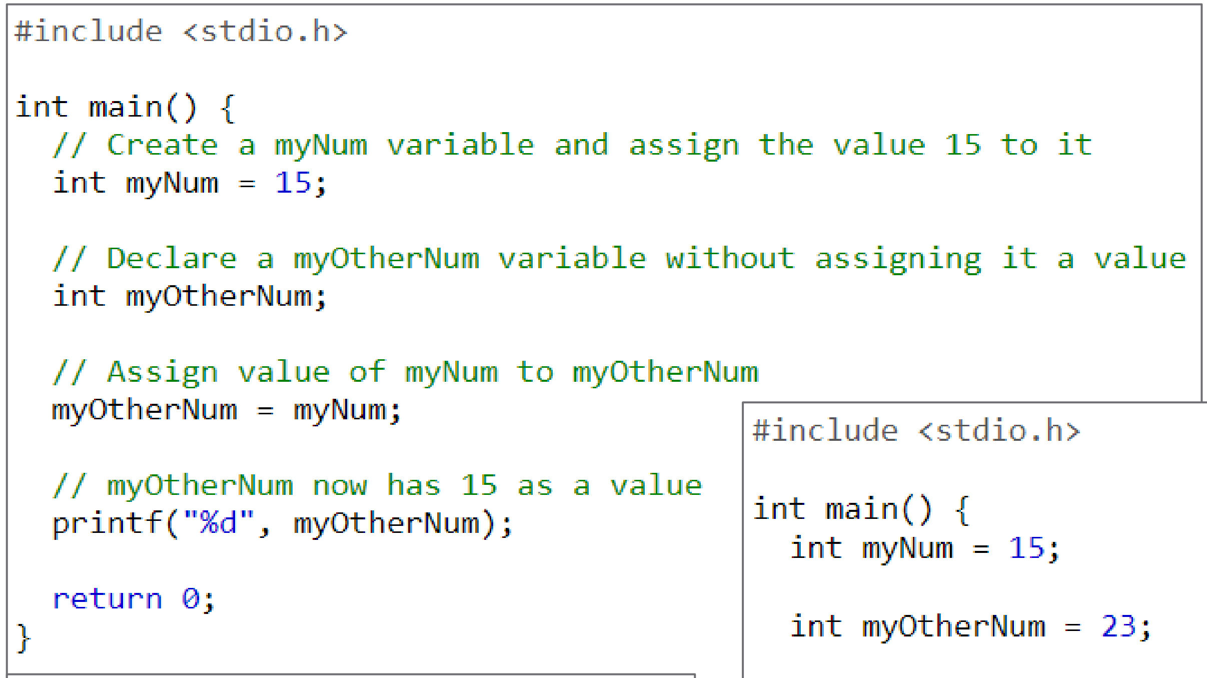
The default value of register variables is a **garbage value**.

Syntax of Register Variables in C

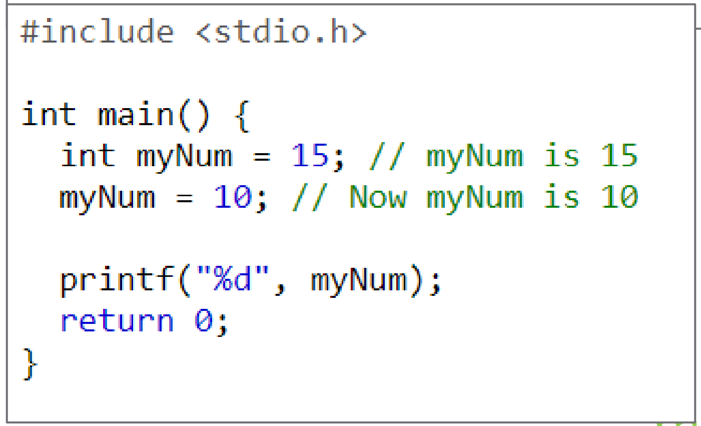
```
register data_type variable_name = initial_value;
Register int number = 122;
```

C Variable Values: Copying Variables

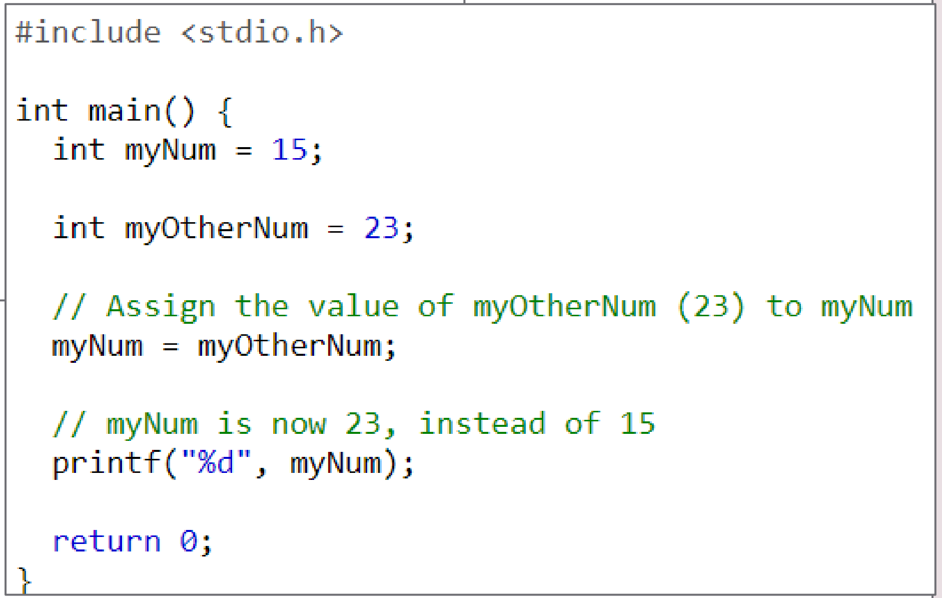
- Change Variable Values. If you assign a new value to an existing variable, it will **overwrite** the previous value:



```c
#include <stdio.h>

int main() {
  // Create a myNum variable and assign the value 15 to it
  int myNum = 15;

  // Declare a myOtherNum variable without assigning it a value
  int myOtherNum;

  // Assign value of myNum to myOtherNum
  myOtherNum = myNum;

  // myOtherNum now has 15 as a value
  printf("%d", myOtherNum);

  return 0;
}
```



```c
#include <stdio.h>

int main() {
  int myNum = 15; // myNum is 15
  myNum = 10; // Now myNum is 10

  printf("%d", myNum);
  return 0;
}
```



```c
#include <stdio.h>

int main() {
  int myNum = 15;

  int myOtherNum = 23;

  // Assign the value of myOtherNum (23) to myNum
  myNum = myOtherNum;

  // myNum is now 23, instead of 15
  printf("%d", myNum);

  return 0;
}
```

```c
int main(){
   int id1, id2, id3;
   int a,b,c,d;
   ///type=1
   id1 = 5;
   id2 = 102;
   id3 = id2;

   id2 = id1;
   id1 = id2;

   printf("id1 = %d\n", id1);
   printf("id2 = %d\n", id2);
   printf("id3 = %d\n\n", id3);

   ///type=2
   a = 61;
   a = 23;
   printf("a = %d\n\n", a);
   a = 78;
   a = 32;
   printf("a = %d\n\n", a);
}
```

Add Variables Together

```
#include <stdio.h>
int main() {
  int x = 5;
  int y = 6;
  int sum = x + y;
  printf("%d", sum);
  return 0;
}
```

Declare Multiple Variables

To declare more than one variable of the same type, use a **comma-separated** list:

```
#include <stdio.h>

int main() {
  int x = 5, y = 6, z = 50;
  printf("%d", x + y + z);
  return 0;
}
```

```
int main(){
    int num1, num2;
    //printf("Enter two number : ");
    //scanf("%d %d", &num1, &num2);
    printf("Enter 1st number : ");
    scanf("%d", &num1);

    printf("Enter 2nd number : ");
    scanf("%d", &num2);

    int sum;
    sum = num1 + num2;\

    //printf("\nsum = %d\n", sum);
    printf("\n\naddition of ( %d + %d ) = %d\n\n", num1, num2, sum);
}
```

You can also assign the **same value** to multiple variables of the same type:

```
#include <stdio.h>

int main() {
  int x, y, z;
  x = y = z = 50;
  printf("%d", x + y + z);
  return 0;
}
```

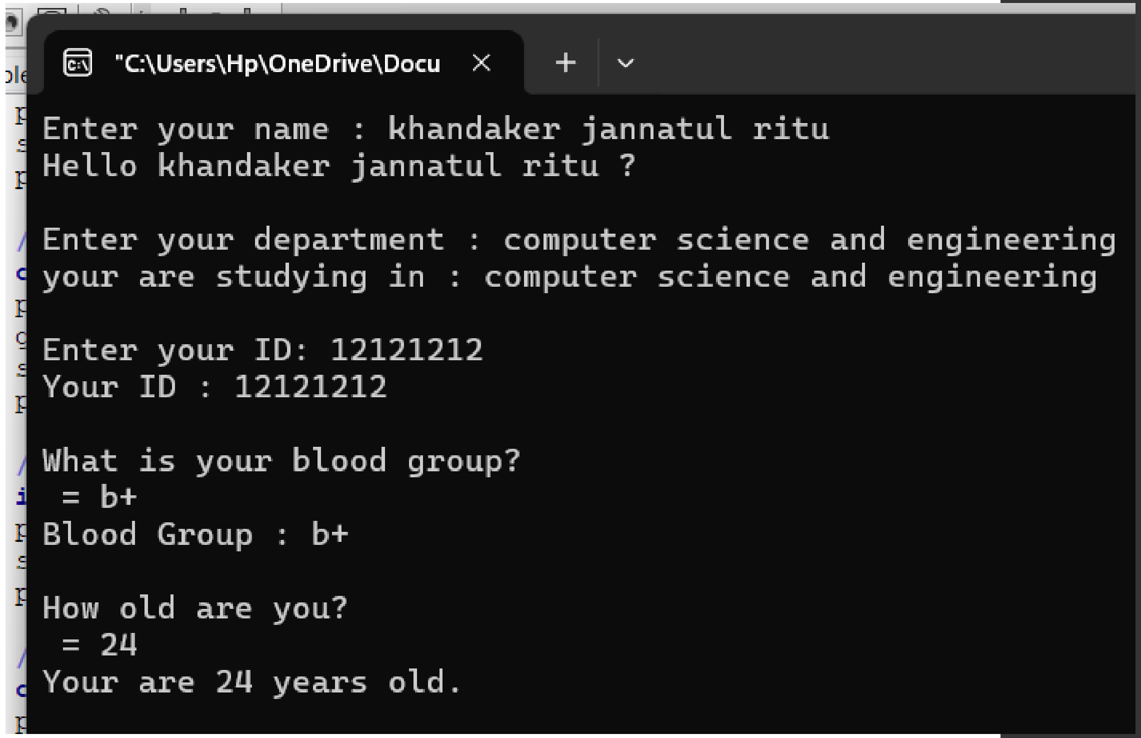
Real-life Example

Often in our examples, we simplify variable names to match their data type (myInt or myNum for int types, myChar for char types, and so on). This is done to avoid confusion.

However, if you want a real-life example of how variables can be used, take a look at the following, where we have made a program that stores different data of a college student:

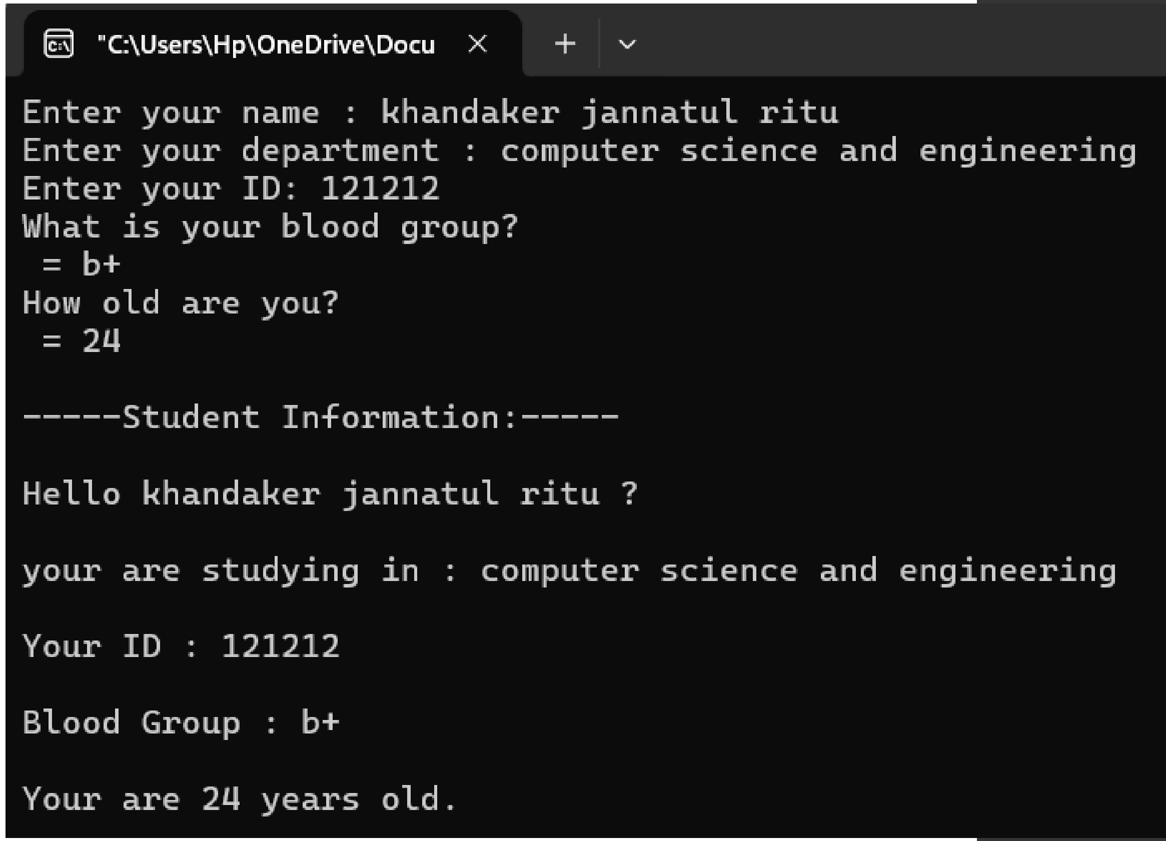
```
#include <stdio.h>

int main() {
  // Student data
  int studentID = 15;
  int studentAge = 23;
  float studentFee = 75.25;
  char studentGrade = 'B';

  // Print variables
  printf("Student id: %d\n", studentID);
  printf("Student age: %d\n", studentAge);
  printf("Student fee: %f\n", studentFee);
  printf("Student grade: %c", studentGrade);
}
```

```
int main()
{
   char name[100];   ///-------YOUR NAME--------///
   printf("Enter your name : ");
   scanf("%[^\n]s", name);
   printf("Hello %s \?\n\n", name);
   char department[50]; ///--------YOUR DEPARTMENT-------//
   printf("Enter your department : ");
   getchar();
   scanf("%[^\n]s", department);
   printf("your are studying in : %s\n\n", department);
   int id; ///---- YOUR ID--------///
   printf("Enter your ID: ");
   scanf("%d", &id);
   printf("Your ID : %d\n\n", id);
   char blood_group[5]; ///----BLOOD GROUP-----///
   printf("What is your blood group?\n = ");
   getchar();
   scanf("%[^\n]s", &blood_group);
   printf("Blood Group : %s\n\n", blood_group);
   int age; ///------AGE-----///
   printf("How old are you?\n = ");
   scanf("%d", &age);
   printf("Your are %d years old.\n\n", age);
}
```



```
Enter your name : khandaker jannatul ritu
Hello khandaker jannatul ritu ?

Enter your department : computer science and engineering
your are studying in : computer science and engineering

Enter your ID: 12121212
Your ID : 12121212

What is your blood group?
 = b+
Blood Group : b+

How old are you?
 = 24
Your are 24 years old.
```

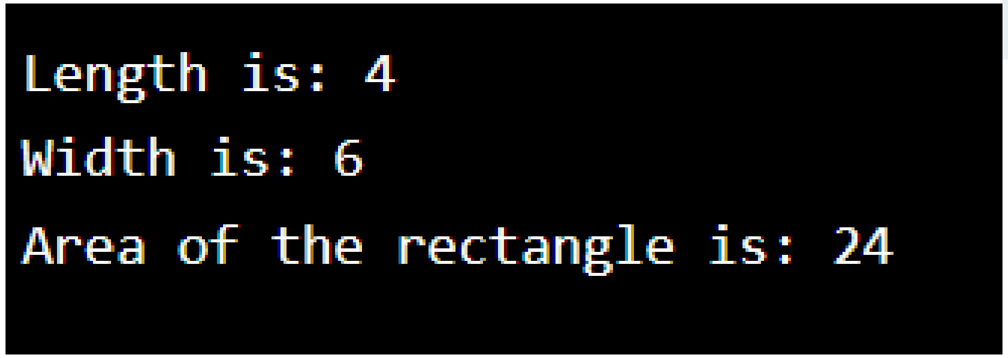
```
char name[100]; ///-------YOUR NAME--------///
printf("Enter your name : ");
scanf("%[^\n]s", name);
char department[50]; ///--------YOUR DEPARTMENT-------//
printf("Enter your department : ");
getchar();
scanf("%[^\n]s", department);
int id; ///---- YOUR ID--------///
printf("Enter your ID: ");
scanf("%d", &id);
char blood_group[5]; ///----BLOOD GROUP-----///
printf("What is your blood group?\n = ");
getchar();
scanf("%[^\n]s", &blood_group);
int age; ///------AGE-----///
printf("How old are you?\n = ");
scanf("%d", &age);

printf("\n-----Student Information:-----\n\n");
printf("Hello %s \?\n\n", name);//1
printf("your are studying in : %s\n\n", department);//2
printf("Your ID : %d\n\n", id);//3
printf("Blood Group : %s\n\n", blood_group);//4
printf("Your are %d years old.\n\n", age);//5
```



```
Enter your name : khandaker jannatul ritu
Enter your department : computer science and engineering
Enter your ID: 121212
What is your blood group?
 = b+
How old are you?
 = 24

-----Student Information:-----

Hello khandaker jannatul ritu ?

your are studying in : computer science and engineering

Your ID : 121212

Blood Group : b+

Your are 24 years old.
```

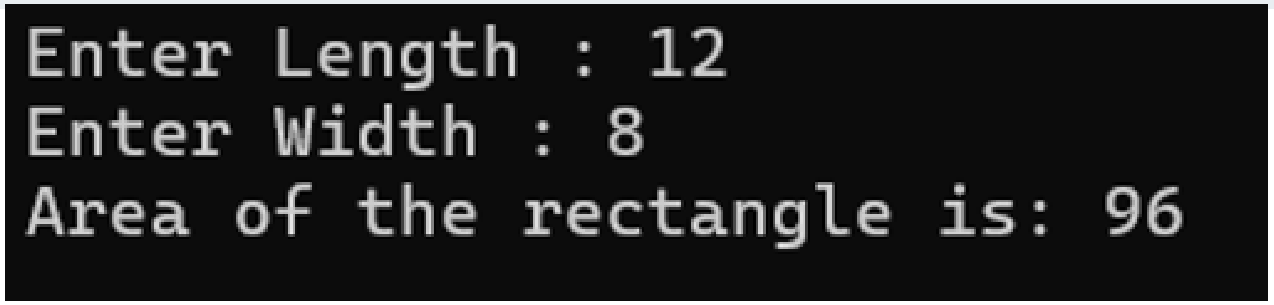
Calculate the Area of a Rectangle (user input)

```
int main() {
  // Create integer variables
  int length = 4;
  int width = 6;
  int area;

  // Calculate the area of a rectangle
  area = length * width;

  // Print the variables
  printf("Length is: %d\n", length);
  printf("Width is: %d\n", width);
  printf("Area of the rectangle is: %d", area);
}
```



```
Length is: 4
Width is: 6
Area of the rectangle is: 24
```

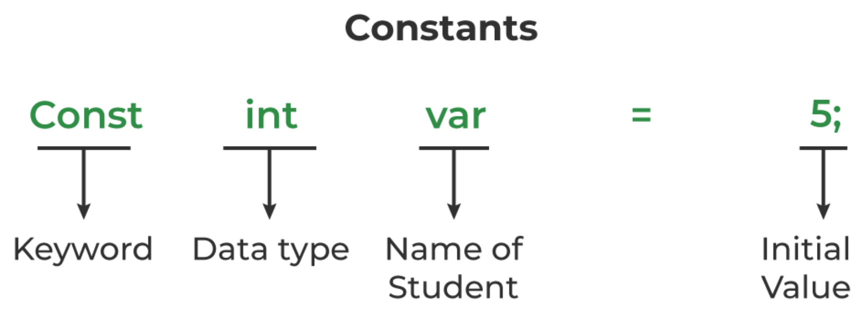
```
int main() {
  // Create integer variables
  int length;
  int width;
  int area;
  printf("Enter Length : ");
  scanf("%d", & length);
  printf("Enter Width : ");
  scanf("%d", & width);
  // Calculate the area of a rectangle
  area = length * width;
  // Print the variables
  printf("Area of the rectangle is: %d\n\n\n", area);
}
```

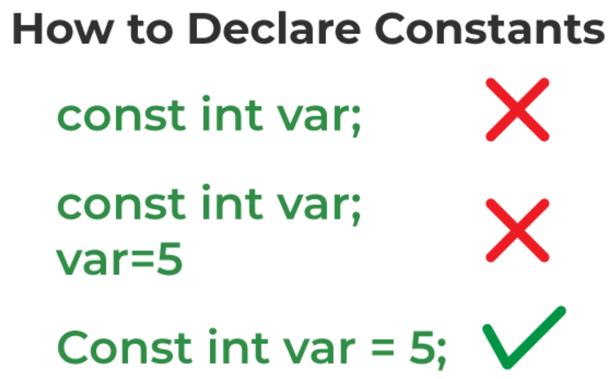


```
Enter Length : 12
Enter Width : 8
Area of the rectangle is: 96
```

Constants in C : Unchangeable Variable

- If you don't want others (or yourself) to change existing variable values, you can use the const keyword.
- This will declare the variable as "constant", which means unchangeable and read-only:
- **How to Define Constant in C?**





Properties of Constant in C
The important properties of constant variables in C defined using the const keyword are as follows:
1. Initialization with Declaration
We can only initialize the constant variable in C at the time of its declaration. Otherwise, it will store the garbage value.
2. Immutability
The constant variables in c are immutable after its definition, i.e., they can be initialized only once in the whole program. After that, we cannot modify the value stored inside that variable.

Constants in C

```
#include <stdio.h>
int main() {
  const int myNum = 15;  // myNum will always be 15
  myNum = 10;  // error: assignment of read-only
variable 'myNum'
  printf("%d", myNum);
  return 0;
}
```

```
prog.c: In function 'main':
prog.c:5:18: error: assignment of read-only
variable 'myNum'
    5 |   myNum = 10;
      |         ^
```

You should always declare the variable as constant when you have values that are unlikely to change:

```
#include <stdio.h>

int main() {
  const int minutesPerHour = 60;
  const float PI = 3.14;

  printf("%d\n", minutesPerHour);
  printf("%f\n", PI);
  return 0;
}
```

```
60
3.140000
```

- Defining Constant using

  #define Preprocessor

- We can also define a constant in C using #define preprocessor. The constants defined using #define are macros that behave like a constant. These constants are not handled by the compiler, they are handled by the preprocessor and are replaced by their value before compilation.

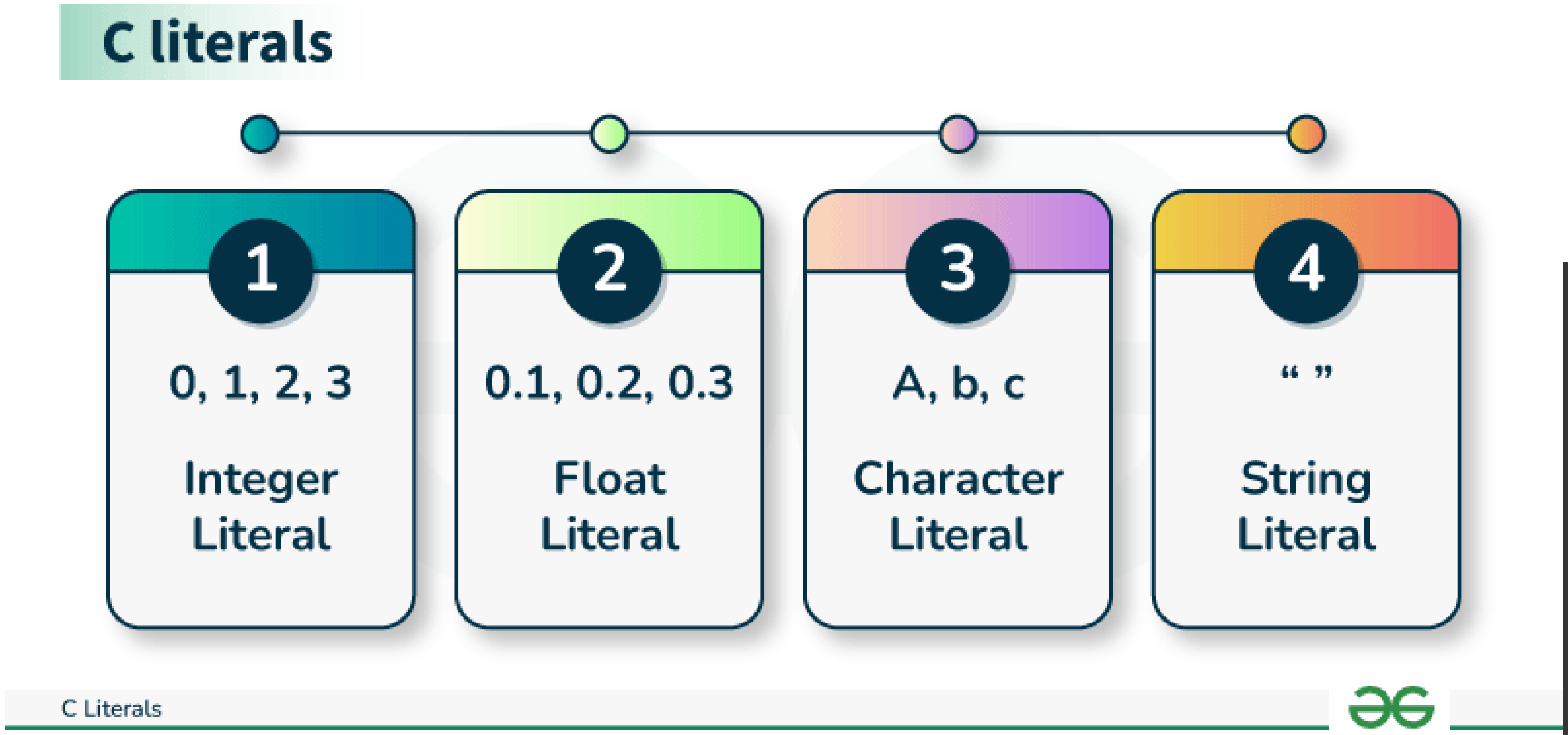
- #define const_name value

```
#include<stdio.h>
#define month 12
#define PI 3.1416

const int ar=15;
const float age=25.5;
const double d;

int main(){
    const int ar=5;
    //ar = 6; // cannot to change

    printf("constant variable ar : %d\n\n", ar);
    printf("constant variable age : %f\n\n", age);

    scanf("%lf", &d);
    printf("constant variable d : %lf\n\n", d);
    //d =45;// cannot to change

    printf("month : %d\n", month);
    printf("value of pi : %f\n", PI);
}
```

Literals

In C, Literals are the constant values that are assigned to the variables. Literals represent fixed values that cannot be modified. Literals contain memory but they do not have references as variables. Generally, both terms, constants, and literals are used interchangeably. Literals are the constant values assigned to the constant variables. For example, “const int = 5;“, is a constant expression and the value 5 is referred to as a constant integer literal.

Types of C Literals
- •Integer Literal
- •Float Literal
- •Character Literal
- •String Literal



```
#include <stdio.h>

int main()
{

	// constant integer literal
	const int intVal = 10;

	printf("Integer Literal:%d \n", intVal);
	return 0;
}
```

```
#include <stdio.h>

int main()
{
	// constant float literal
	const float floatVal = 4.14;

	printf("Floating point literal: %.2f\n",floatVal);
	return 0;
}
```

```
#include <stdio.h>
int main()
{
	// constant char literal
	const char charVal = 'A';

	printf("Character Literal: %c\n",charVal);
	return 0;
}
```

```
#include <stdio.h>

int main()
{
	const char str[]= "Welcome\nTo\nGeeks\tFor\tGeeks";
	printf("%s", str);
	return 0;
}
```

Difference Between Constants and Literals

Constant	Literals
Constants are variables that cannot be modified once declared.	Literals are the fixed values that define themselves.
Constants are defined by using the const keyword in C. They store literal values in themselves.	They themselves are the values that are assigned to the variables or constants.
We can determine the address of constants.	We cannot determine the address of a literal except string literal.
They are lvalues.	They are rvalues.
Example: const int c = 20.	Example: 24,15.5, ‘a’, “Geeks”, etc.

Language Processors: Assembler, Compiler and Interpreter

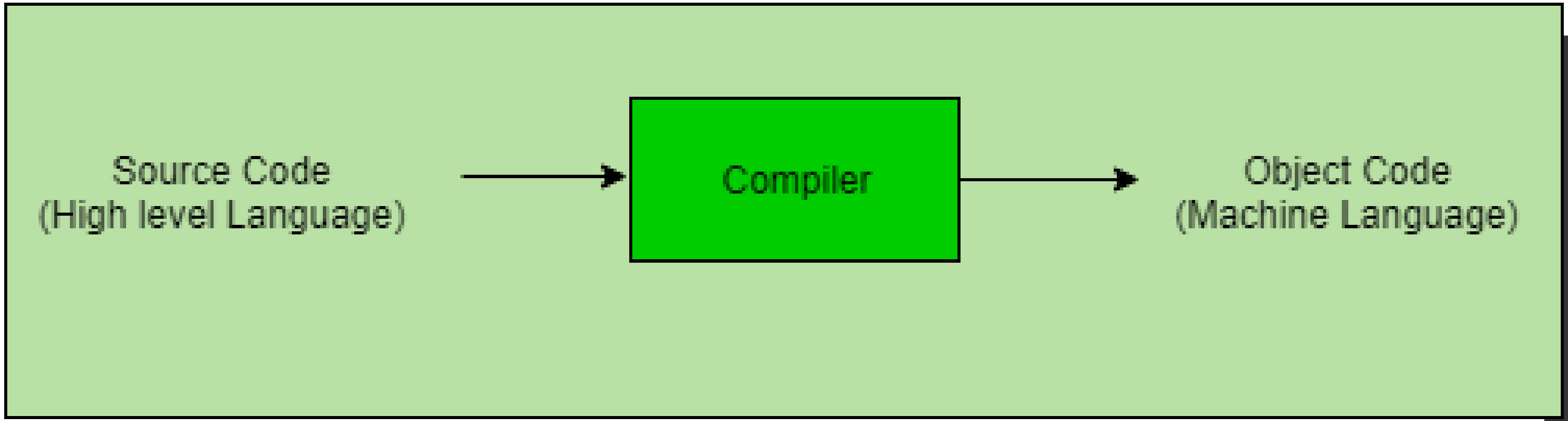
- **What is Language Processors?**

- Compilers, interpreters, translate programs written in high-level languages into machine code that a computer understands and assemblers translate programs written in low-level or assembly language into machine code. In the compilation process, there are several stages. To help programmers write error-free code, tools are available.

- **1. Compiler**

- The language processor that reads the complete source program written in high-level language as a whole in one go and translates it into an equivalent program in machine language is called a Compiler. Example: C, C++, C#.

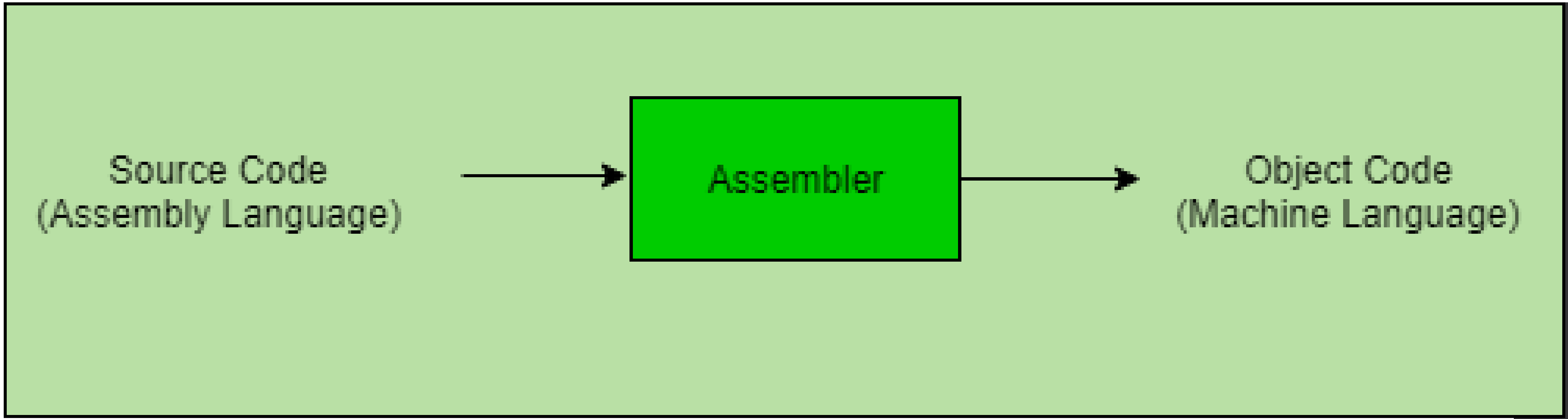
- In a compiler, the source code is translated to object code successfully if it is free of errors. The compiler specifies the errors at the end of the compilation with line numbers when there are any errors in the source code. The errors must be removed before the compiler can successfully recompile the source code again the object program can be executed number of times without translating it again.



- ## 2. Assembler

- The Assembler is used to translate the program written in Assembly language into machine code. The source program is an input of an assembler that contains assembly language instructions. The output generated by the assembler is the object code or machine code understandable by the computer. Assembler is basically the 1st interface that is able to communicate humans with the machine. We need an assembler to fill the gap between human and machine so that they can communicate with each other. code written in assembly language is some sort of mnemonics(instructions) like ADD, MUL, MUX, SUB, DIV, MOV and so on. and the assembler is basically able to convert these mnemonics in binary code. Here, these mnemonics also depend upon the architecture of the machine.

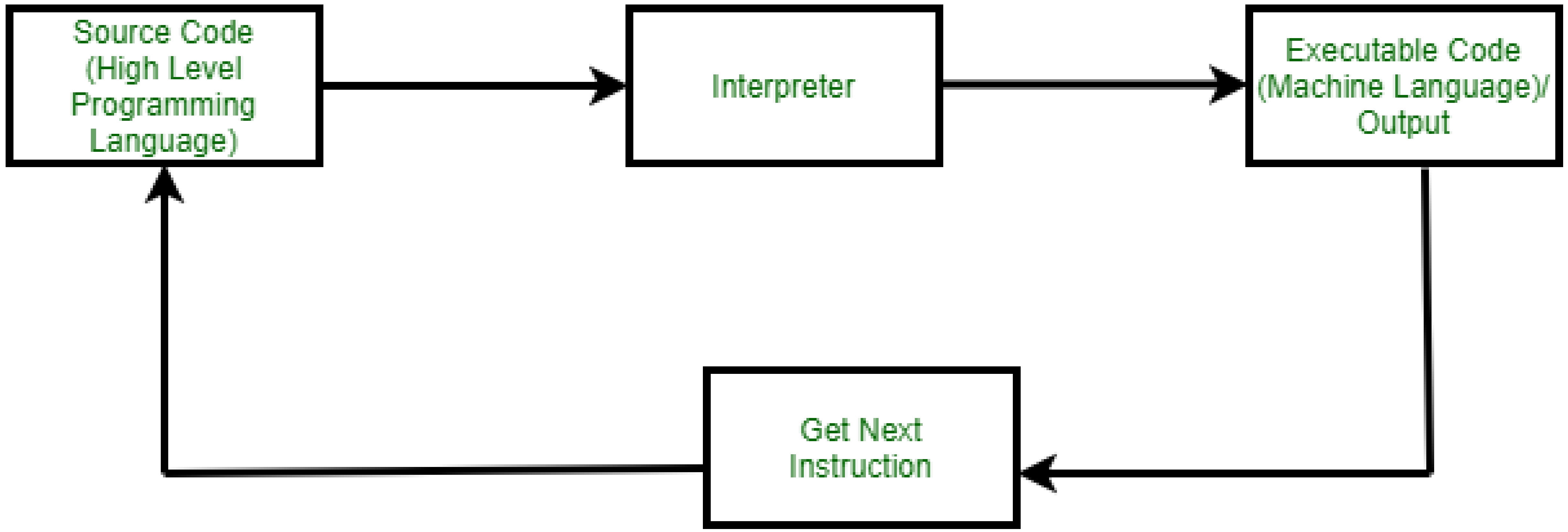
- For example, the architecture of intel 8085 and intel 8086 are different.



- **3. Interpreter**

- The translation of a single statement of the source program into machine code is done by a language processor and executes immediately before moving on to the next line is called an interpreter. If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message. The interpreter moves on to the next line for execution only after the removal of the error. An Interpreter directly executes instructions written in a programming or scripting language without previously converting them to an object code or machine code. An interpreter translates one line at a time and then executes it.

- Example: Perl, Python and Matlab.


Source Code (High Level Programming Language)
Interpreter
Executable Code (Machine Language)/ Output
Get Next Instruction

Difference Between Compiler and Interpreter

Compiler	Interpreter
A compiler is a program that converts the entire source code of a programming language into executable machine code for a CPU.	An interpreter takes a source program and runs it line by line, translating each line as it comes to it.
The compiler takes a large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	An interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
The compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present anywhere in the program.	Its Debugging is easier as it continues translating the program until the error is met.
The compiler requires a lot of memory for generating object codes.	It requires less memory than a compiler because no object code is generated.
Generates intermediate object code.	No intermediate object code is generated.
For Security purpose compiler is more useful.	The interpreter is a little vulnerable in case of security.
Examples: C, C++, C#	Examples: Python, Perl, JavaScript, Ruby.

Video Resources

- C programming Bangla Tutorial 5.4 : Translator program - Compiler,Interpreter
- C programming Bangla Tutorial 5.15 : Keyword, Variable, data type (part-1)
- C programming Bangla Tutorial 5.16 : Keyword, Variable, data type (part-2)
- C programming Bangla Tutorial 5.17 : More on data types
- C programming Bangla Tutorial 5.18 : how to get user input using scanf
- C programming Bangla Tutorial 5.19 : Get inputs using gets, fgets, getchar