# 0301102 LOGIC DEVELOPMENT & PROGRAMMING

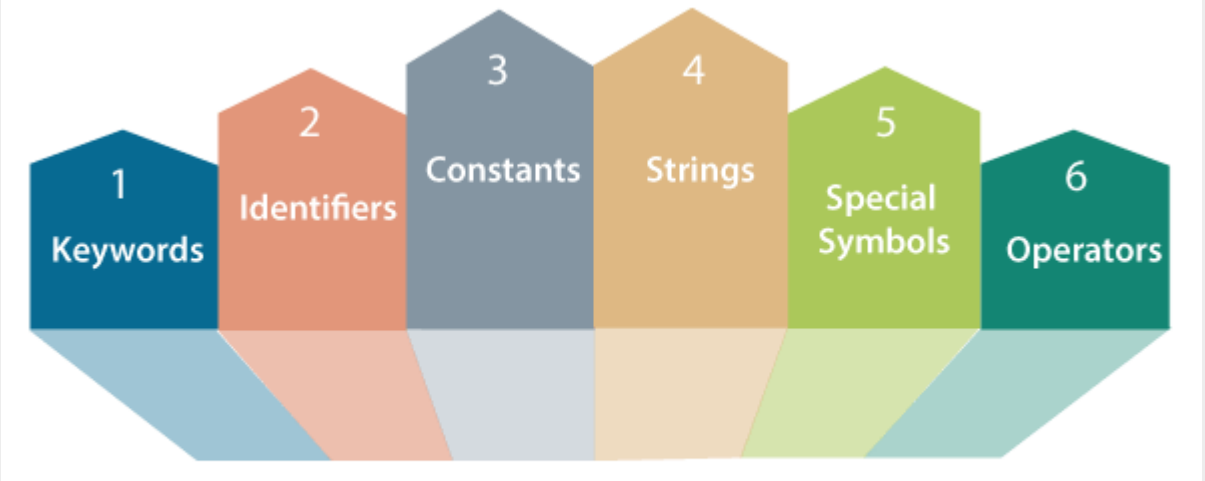
## UNIT – 2

## Principles of Programming Language

# Tokens

- Tokens in C is the most important element to be used in creating a program in C.
- A token is the smallest element of a program that is meaningful to the compiler.
- Tokens can be classified as follows:

1) Keywords
2) Identifiers/Variables
3) Constants
4) Strings
5) Special Symbols
6) Operators



# Tokens

1) **Keyword:** Keywords are pre-defined or reserved words in a programming language.
2) **Identifiers:** Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words.
3) **Strings:** Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

- Now, we describe the strings in different ways:
  - char a[10] = "GLS University"; // The compiler allocates the 14 bytes to the 'a' array.
    - char a[] = "GLS University"; // The compiler allocates the memory at the run time.
    - char a[10] = {'G','L','S',' ','U','n','i','v','e','r','s','i','t','y','\0'}; // String is represented in the form of characters.

# Tokens

4) **Operators:** Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

- **Unary Operator:** A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.
- **Binary Operator:** The binary operator is an operator applied between two operands. The following is the list of the binary operators:
- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators
- Bitwise Operators
- Conditional Operators
- Assignment Operator
- Misc Operator

# Tokens

**5) Constants:** A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

6) **Special Characters:** Some special characters are used in C, and they have a special meaning which cannot be used for another purpose

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call, separating the variable when printing the value of more than one variable using a single printf statement.

# Tokens

- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

# Type Conversion

- Type Conversion (Typecasting) is converting one data type into another one. It is also called as data conversion or type conversion.
- C programming provides two types of type casting operations:
  - **Implicit type casting**
  - **Explicit type casting**
- **int a = 10 ->float - 10.0000 - implicit**
- **float f=2.345 -> int - 2 - explicit**

# Type Conversion

- **Implicit type casting:** Also known as '**automatic type conversion**'. Implicit type casting means conversion of data types **without losing its original meaning.** This type of typecasting is essential when you want to change data types without changing the significance of the values stored inside the variable.
- If the **operands** are of two different data types, then an operand having lower data type is automatically converted into a higher data type.
- Implicit casting can take place when converting from bigger data type to smaller data type and vice versa. We just need to keep in mind that if bigger data types implicitly converted to smaller data type then there may be loss of data. Here the conversion is taking place from float to int (int is smaller data type than float; .30 is getting lost)

# Type Conversion

**EXAMPLE 1:**

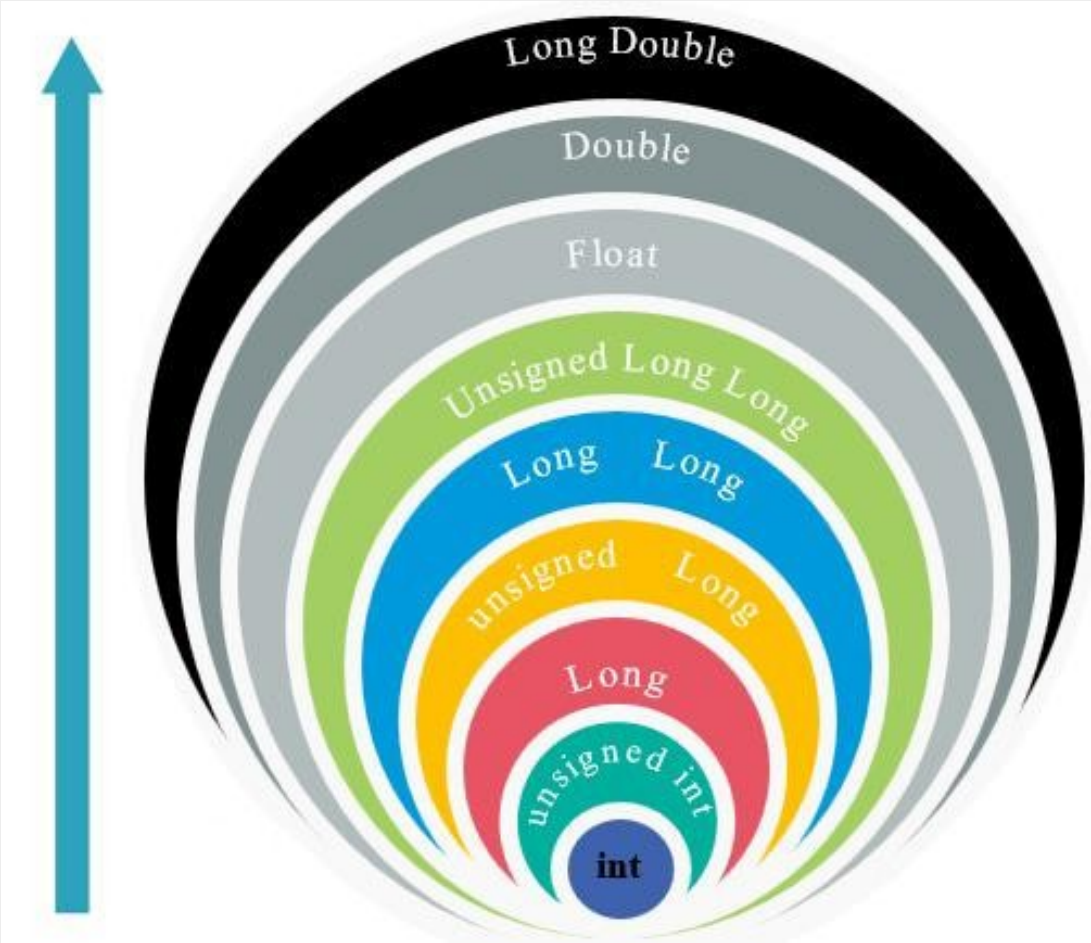
```
int main () {
int a;
float b= 5.30;
a = b;
printf("%d", a);}
```

**EXAMPLE 2:**

```
int main() {
   int  num = 13;
   char c = 'k'; /* ASCII value is 107 */
   float sum;
   sum = num + c;
   printf("sum = %f\n", sum );}
```

# Type Conversion

- **Arithmetic Conversion Hierarchy**
- The compiler first proceeds with promoting a character to an integer. If the operands still have different data types, then they are converted to the highest data type that appears in the following hierarchy chart:

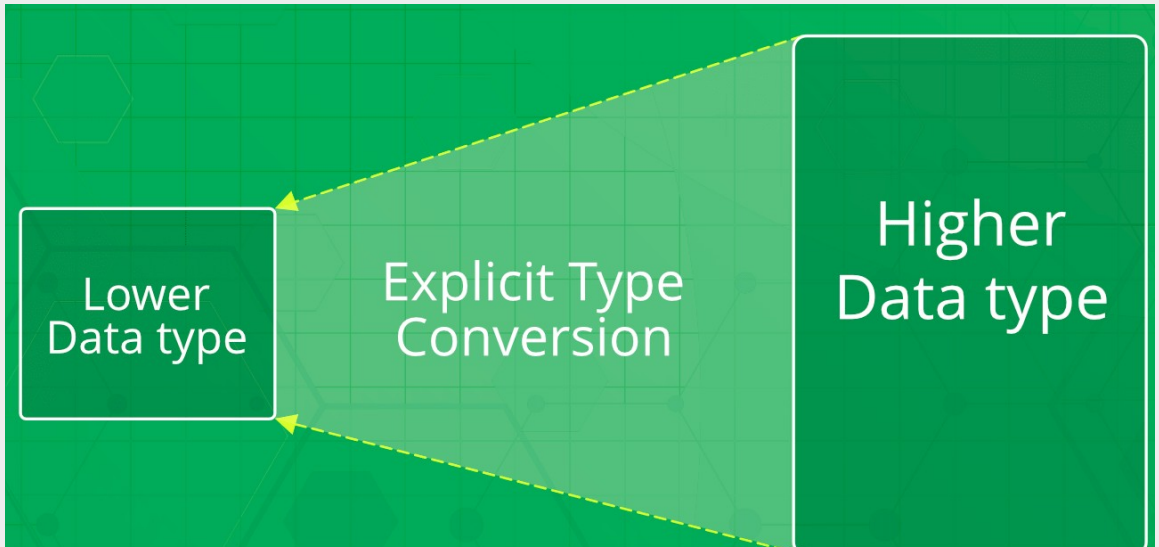


# Type Conversion

- **Important Points about Implicit Conversions:**
  - Implicit type of type conversion is also called as standard type conversion. We do not require any keyword or special statements in implicit type casting.
  - The implicit type conversion always happens with the compatible data types.
- **We cannot perform implicit type casting on the data types which are not compatible with each other such as:**
  - Converting **float to an int** will truncate the fraction part hence losing the meaning of the value.
- 'C' programming provides another way of typecasting which is **explicit type casting.**

# Type Conversion

- **Explicit type casting:**
- There are some scenarios in which we may have to **force type conversi**on. *Suppose we have a variable **result** that stores the division of two **variables** which are declared as an int data type.*
    - **int result, var1=10, var2=3;**
    - **result=var1/var2;**
- After the division performed on variables var1 and var2 the result stored in the variable "result" will be in an integer format. Whenever this happens, the value stored in the variable "result" loses its meaning because it does not consider the fraction part which is normally obtained in the division of two numbers.
- ***To force the type conversion in such situations, we use explicit type casting.***



# Type Conversion

- **Explicit type casting:**
- It requires a type casting operator. The general **syntax** for type casting operations is as follows:
  - ***data-type-name expression/variablename***
- **Example:**

```
int main()
{
float a = 1.2;
//int b  = a; // don’t go
int b = (int)a + 1;
printf("Value of a is %f\n", a);
printf("Value of b is %d\n",b);
return 0;
}
```

# Type Conversion

- **Explicit type casting:**

```
#include <stdio.h>

int main () {

int a,b;

float c;

a = 5; b = 3;

c = (float)a/b;

printf("%f", c);

}
```