Introduction to programming in C

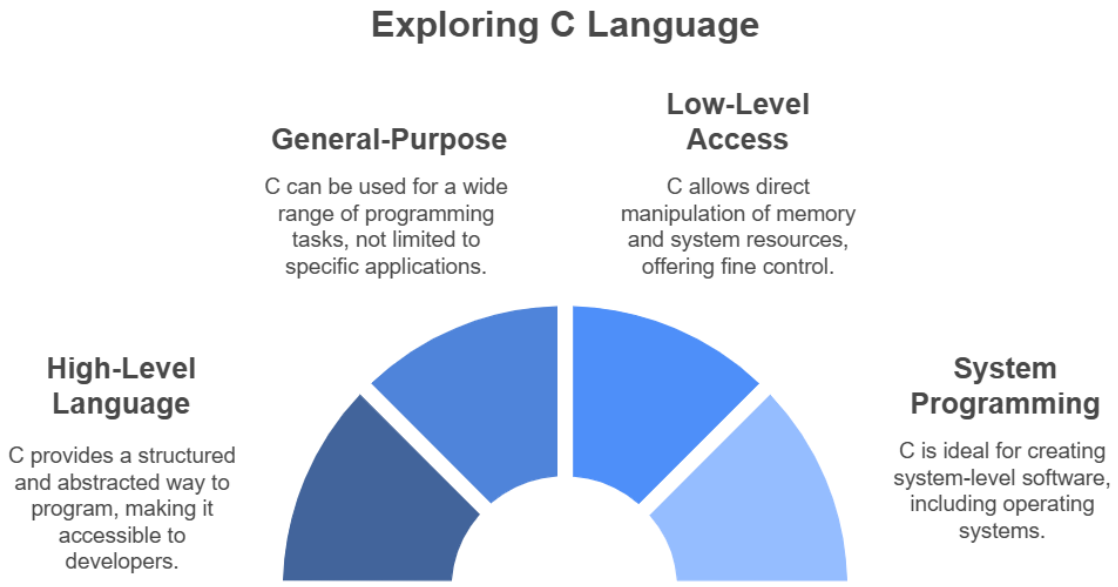
Dr. Sangeetha Annam

Department of Computer Science and Engineering,
Chitkara University, Punjab

Introduction to C

What is C programming?

- **Definition:** C is a high-level, general-purpose programming language that provides low-level access to memory and system processes.
- Designed for system programming, it is widely used for developing operating systems.

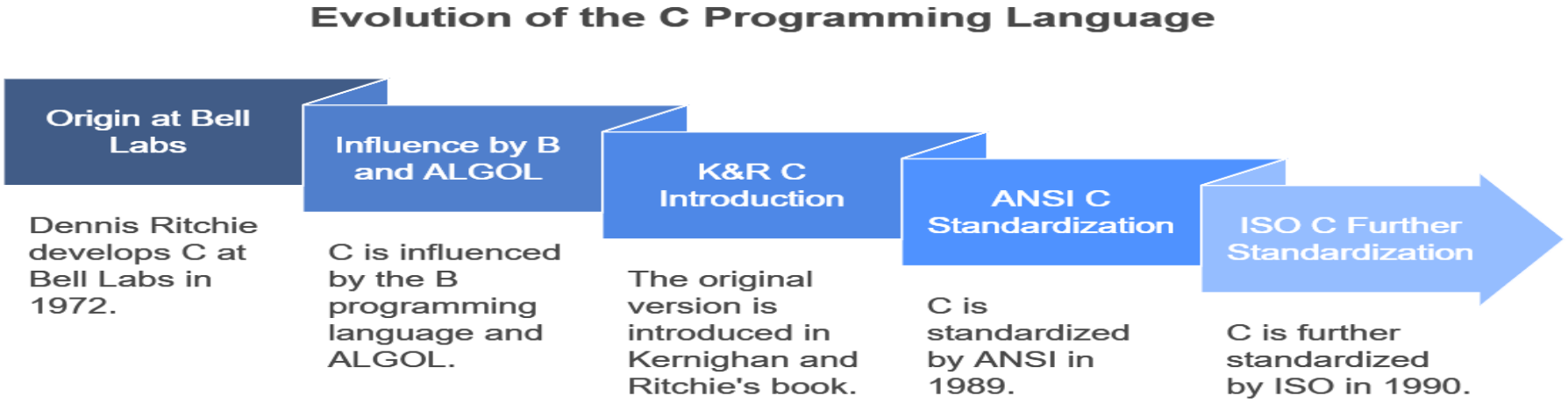


History of C

- **Origin:** Developed by Dennis Ritchie at Bell Labs in 1972.
- **Influences:** Based on the B programming language and influenced by ALGOL.
- **Evolution:**
  - K&R C: The original version described in "The C Programming Language" book by Kernighan and Ritchie.
  - ANSI C: Standardized version by ANSI in 1989.
  - ISO C: Further standardized by ISO in 1990.
  - operating systems.

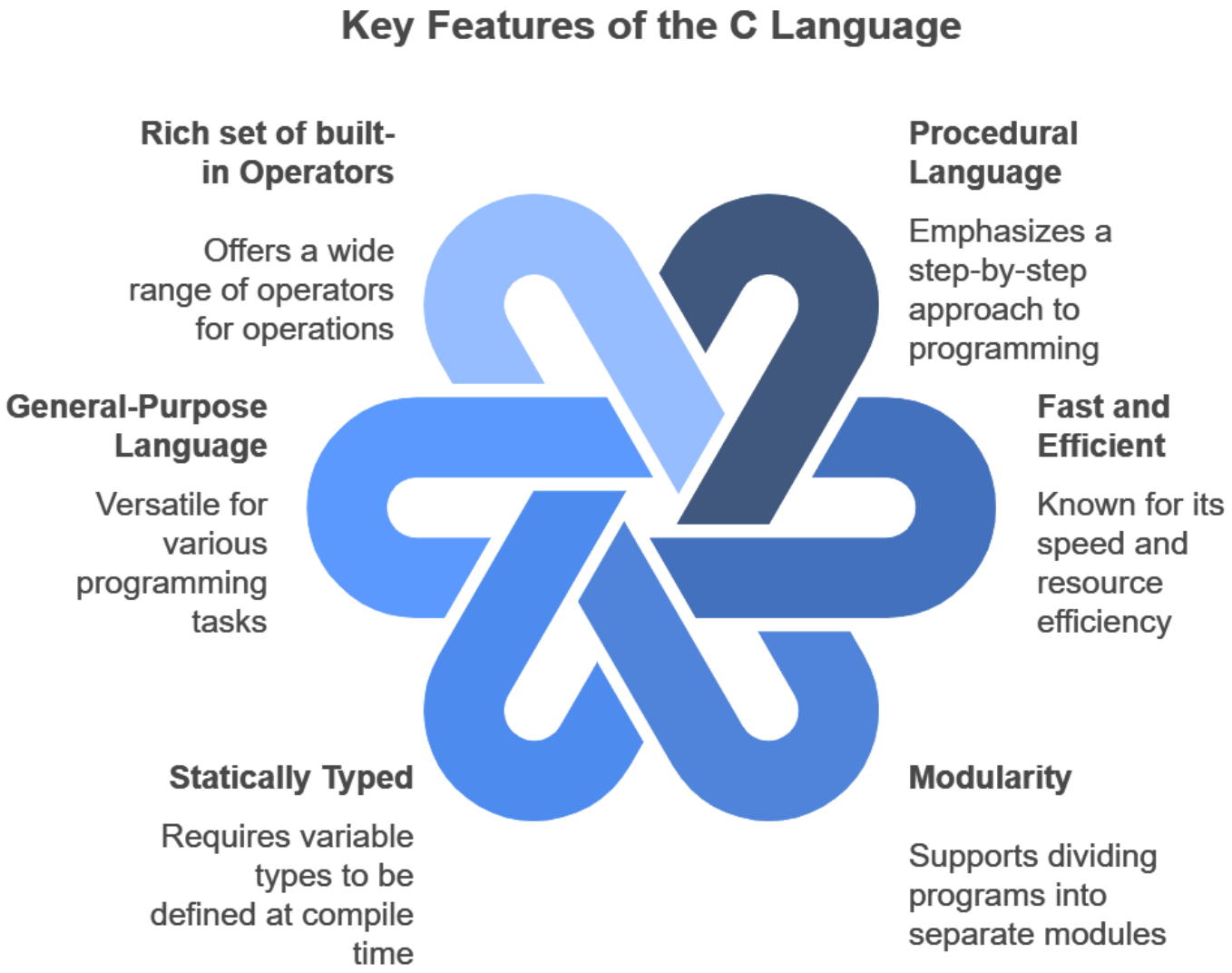
History of C

CHITKARA UNIVERSITY



Features of C

CHITKARA UNIVERSITY



Structure of C Program

Header	#include <stdio.h>
main()	int main() {
Variable declaration	int a = 10;
Body	Printf(@%d@,a);
Return	return 0; }

Structure of C

- **Header Files:** Include necessary libraries (e.g., #include <stdio.h>).
- **Main Function:** Entry point of the program (int main()).
- **Variable Declarations:** Declare variables to store data
- **Statements and Expressions:** Perform operations and control program flow.
- **Return Functions:** Define reusable code blocks.

Basic Syntax

CHITKARA UNIVERSITY

```
#include <stdio.h> //
Preprocessor directive

// Function declaration
int main() {
    // Variable declaration
    int a;

    // Initialization
    a = 10;

    // Function call
    printf("Value of a is %d\n",
a);

    // Return statement
    return 0;
```

- **Preprocessor Directive:**
- ✓ #include <stdio.h>: This line includes the Standard Input Output library, which allows the use of functions like printf.
- **Function Declaration:**
- ✓ int main() : This line declares the main function, which is the entry point of any C program.
- **Variable Declaration:**
- ✓ int a; : This line declares an integer variable a.

Basic Syntax

CHITKARA UNIVERSITY

```
#include <stdio.h> //
Preprocessor directive

// Function declaration
int main() {
    // Variable declaration
    int a;

    // Initialization
    a = 10;

    // Function call
    printf("Value of a is %d\n", a);

    // Return statement
    return 0;
}
```

- **Initialization::**
- ✓ a = 10;: This line initializes the variable a with the value 10.
- **Return Statement:**
- ✓ return 0;: This line returns 0, indicating that the program has been executed successfully.
- **Closing Bracket:**
- ✓ }: This line marks the end of the main function.

Keywords in C

CHITKARA UNIVERSITY

- Keywords are predefined or reserved words that have special meanings to the compiler. These are part of the syntax and cannot be used as identifiers in the program. A list of keywords in C or reserved words in the C programming language are mentioned below:

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

Variables in C

CHITKARA UNIVERSITY

- A variable in C language is the name associated with some memory location to store data of different types. There are many types of variables in C depending on the scope, storage class, lifetime, type of data they store, etc. A variable is the basic building block of a C program that can be used in expressions as a substitute in place of the value it stores.

C Variable Syntax

```
data_type variable_name = value;    // defining single variable
            or
data_type variable_name1, variable_name2;    // defining multiple variable
```

C Variable Syntax

CHITKARA UNIVERSITY

```
data_type variable_name = value;    // defining single variable
            or
data_type variable_name1, variable_name2;    // defining multiple 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.

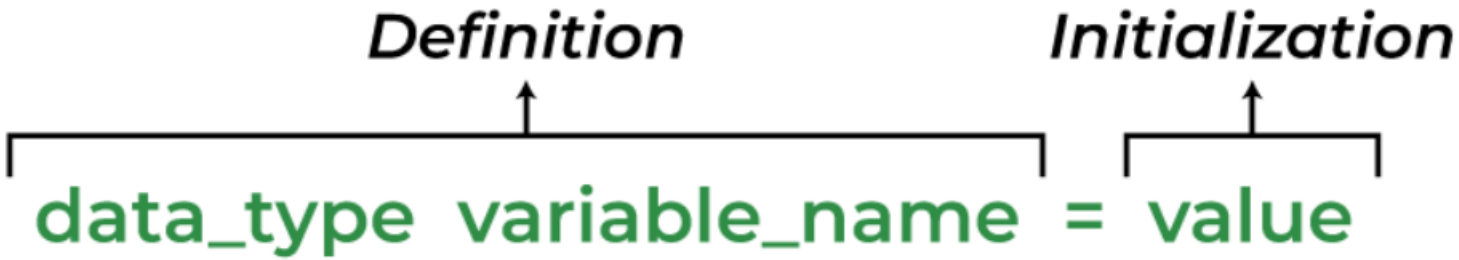
Example

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

C Variable Syntax

CHITKARA UNIVERSITY

Note: *C is a strongly typed language so all the variables types must be specified before using them.*



Variable Syntax Breakdown

There are 3 aspects of defining a variable:

1.Variable Declaration

2.Variable Definition

3.Variable Initialization

C Variable Syntax

CHITKARA UNIVERSITY

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, an entry in symbol table is created and memory will be allocated at the time of initialization of the variable.

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.

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

Variable Initialization: Initialization of a variable is the process where the user assigns some meaningful value to the variable when creating the variable.

```
int var = 10;    // variable declaration and definition (i.e. Vairable Initialization)
```

C Variable Syntax

CHITKARA UNIVERSITY

CODE

```
#include <stdio.h>

int main()
{
    // declaration with definition
    int defined_var;

    printf("Defined_var: %d\n", defined_var);

    // assignment
    defined_var = 12;

    // declaration + definition + initialization
    int ini_var = 25;

    printf("Value of defined_var after assignment: %d\n",
defined_var);
    printf("Value of ini_var: %d", ini_var);

    return 0;
}
```

OUTPUT

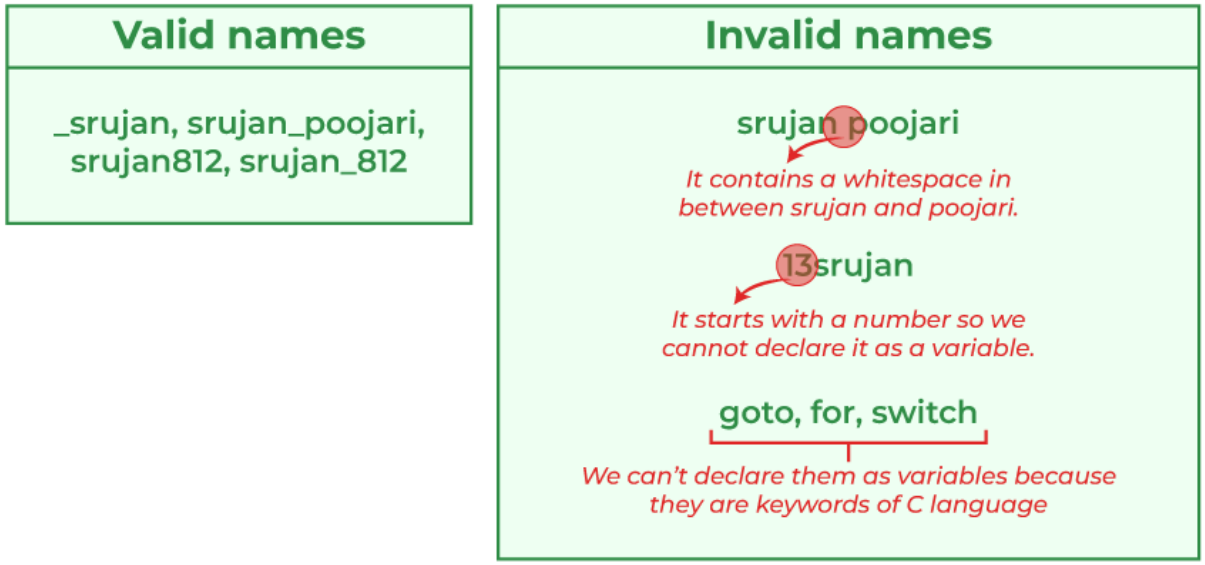
```
Defined_var: 0
Value of defined_var after assignment: 12
Value of ini_var: 25
```

Rules for Naming Variables in C

CHITKARA UNIVERSITY

You can assign any name to the variable as long as it follows the following rules:

1. A variable name must only contain alphabets, digits, and underscore.
2. A variable name must start with an alphabet or an underscore only. It cannot start with a digit.
3. No white space is allowed within the variable name.
4. A variable name must not be any reserved word or keyword.



C Variable Types

The C variables can be classified into the following types:

1. **Local Variables**
2. **Global Variables**
3. **Static Variables**
4. **Automatic Variables**
5. **Extern Variables**
6. **Register Variables**

Local Variables in C

A **Local variable in C** is a variable that is declared inside a function or a block of code. Its scope is limited to the block or function in which it is declared.

CODE

```
#include <stdio.h>

void function()
{
    int x = 10; // local variable
    printf("%d", x);
}

int main() { function(); }
```

OUTPUT

```
10
```

Global Variables in C

A Global variable in C is a variable that is declared outside the function or a block of code. Its scope is the whole program i.e. we can access the global variable anywhere in the C program after it is declared.

CODE

```
#include <stdio.h>

int x = 20; // global variable

void function1() { printf("Function 1: %d\n", x); }

void function2() { printf("Function 2: %d\n", x); }

int main()
{

    function1();
    function2();
    return 0;
}
```

OUTPUT

```
Function 1: 20
Function 2: 20
```

Static Variables in C

A **static variable in C** is a variable that is defined using the **static** keyword. It can be defined only once in a C program and its scope depends upon the region where it is declared (can be **global or local**).

The **default value** of static variables is **zero.**

CODE

```
#include <stdio.h>

void function()
{
    int x = 20; // local variable
    static int y = 30; // static variable
    x = x + 10;
    y = y + 10;
    printf("\tLocal: %d\n\tStatic: %d\n", x, y);
}

int main()
{
    printf("First Call\n");
    function();
    printf("Second Call\n");
    function();
    printf("Third Call\n");
    function();
    return 0;
}
```

OUTPUT

```
First Call
    Local: 30
    Static: 40
Second Call
    Local: 30
    Static: 50
Third Call
    Local: 30
    Static: 60
```

Automatic Variable in C

All the local variables are automatic variables by default. They are also known as auto variables. Their scope is local and their lifetime is till the end of the block. If we need, we can use the auto keyword to define the auto variables. The default value of the auto variables is a garbage value..

CODE

```
#include <stdio.h>

void function()
{
    int x = 10; // local variable (also automatic)
    auto int y = 20; // automatic variable
    printf("Auto Variable: %d", y);
}
int main()
{

    function();
    return 0;
}
```

OUTPUT

```
Auto Variable: 20
```

External Variables in C

External variables in C can be shared between multiple C files. We can declare an external variable using the extern keyword.

Their scope is global and they exist between multiple C files.

CODE

```
----------myfile.h------------
extern int x=10; //external variable (also global)



----------program1.c----------
#include "myfile.h"
#include <stdio.h>
void printValue(){
printf("Global variable: %d", x);
}
```

In the above example, x is an external variable that is used in multiple C files.

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.

CODE

```
#include <stdio.h>

int main()
{
    //    register variable
    register int var = 22;

    printf("Value of Register Variable: %d\n", var);
    return 0;
}
```

OUTPUT

```
Value of Register Variable: 22
```

Constant Variable in C

A constant variable in C is a read-only variable whose value cannot be modified once it is defined. We can declare a constant variable using the const keyword.

CODE

```
#include <stdio.h>

int main()
{
    // variable
    int not_constant;

    // constant variable;
    const int constant = 20;

    // changing values
    not_constant = 40;
    constant = 22;

    return 0;
}
```

OUTPUT

```
variables.c: In function 'main':
variables.c:13:13: error: assignment of read-only variable 'constant'
   13 |     constant = 22;
      |             ^
```

Constants in C

As the name suggests, a constant in C is a variable that cannot be modified once it is declared in the program. We can not make any change in the value of the constant variables after they are defined.

CODE

```
#include <stdio.h>

int main()
{

    // defining integer constant using const keyword
    const int int_const = 25;

    // defining character constant using const keyword
    const char char_const = 'A';

    // defining float constant using const keyword
    const float float_const = 15.66;

    printf("Printing value of Integer Constant: %d\n",
           int_const);
    printf("Printing value of Character Constant: %c\n",
           char_const);
    printf("Printing value of Float Constant: %f",
           float_const);

    return 0;
}
```

OUTPUT

```
Printing value of Integer Constant: 25
Printing value of Character Constant: A
Printing value of Float Constant: 15.660000
```

CHITKARA UNIVERSITY

How to Declare Constants

```
const int var;
```

```
const int var;
var=5
```

```
Const int var = 5;
```

The type of the constant is the same as the data type of the variables. Following is the list of the types of constants

1. Integer Constant
2. Character Constant
3. Floating Point Constant
4. Double Precision Floating Point Constant
5. Array Constant
6. Structure Constant

Difference Between Constants and Literals

CHITKARA UNIVERSITY

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.

Data Types in C

CHITKARA UNIVERSITY

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it.

The data types in C can be classified as follows:

Types	Description	Data Types
Primitive Data Types	Primitive data types are the most basic data types that are used for representing simple values such as integers, float, characters, etc.	int, char, float, double, void
Derived Types	The data types that are derived from the primitive or built-in datatypes are referred to as Derived Data Types.	array, pointers, function
User Defined Data Types	The user-defined data types are defined by the user himself.	structure, union, enum

Integer Data Type

The integer datatype in C is used to store the integer numbers (any number including positive, negative and zero without decimal part). Octal values, hexadecimal values, and decimal values can be stored in int data type in C.

- **Range:** -2,147,483,648 to 2,147,483,647
- **Size:** 4 bytes
- **Format Specifier:** %d

The integer data type can also be used as

1. **unsigned int:** Unsigned int data type in C is used to store the data values from zero to positive numbers but it can't store negative values like signed int.
2. **short int:** It is lesser in size than the int by 2 bytes so can only store values from -32,768 to 32,767.
3. **long int:** Larger version of the int datatype so can store values greater than int.
4. **unsigned short int:** Similar in relationship with short int as unsigned int with int.

Integer Data Type

CHITKARA UNIVERSITY

CODE

```
#include <stdio.h>

int main()
{
    // Integer value with positive data.
    int a = 9;

    // integer value with negative data.
    int b = -9;

    // U or u is Used for Unsigned int in C.
    int c = 89U;

    // L or l is used for long int in C.
    long int d = 99998L;

    printf("Integer value with positive data: %d\n", a);
    printf("Integer value with negative data: %d\n", b);
    printf("Integer value with an unsigned int data: %u\n",
           c);
    printf("Integer value with an long int data: %ld", d);

    return 0;
}
```

OUTPUT

```
Integer value with positive data: 9
Integer value with negative data: -9
Integer value with an unsigned int data: 89
Integer value with an long int data: 99998
```

Character Data Type

Character data type allows its variable to store only a single character. The size of the character is 1 byte. It is the most basic data type in C. It stores a single character and requires a single byte of memory in almost all compilers.

- **Range:** (-128 to 127) or (0 to 255)
- **Size:** 1 byte
- **Format Specifier:** %c

OUTPUT

```
Value of a: a
Value of a after increment is: b
Value of c: c
```

CODE

```
#include <stdio.h>

int main()
{
    char a = 'a';
    char c;

    printf("Value of a: %c\n", a);

    a++;
    printf("Value of a after increment is: %c\n", a);

    // c is assigned ASCII values
    // which corresponds to the
    // character 'c'
    // a-->97 b-->98 c-->99
    // here c will be printed
    c = 99;

    printf("Value of c: %c", c);

    return 0;
}
```

Float Data Type

In C programming float data type is used to store floating-point values. Float in C is used to store decimal and exponential values. It is used to store decimal numbers (numbers with floating point values) with single precision.

Range: 1.2E-38 to 3.4E+38

Size: 4 bytes

Format Specifier: %f

OUTPUT

```
9.000000
2.500000
0.000200
```

CODE

```
#include <stdio.h>

int main()
{
    float a = 9.0f;
    float b = 2.5f;

    // 2x10^-4
    float c = 2E-4f;
    printf("%f\n", a);
    printf("%f\n", b);
    printf("%f", c);

    return 0;
}
```

Double Data Type

A Double data type in C is used to store decimal numbers (numbers with floating point values) with double precision. It is used to define numeric values which hold numbers with decimal values in C.

The double data type is basically a precision sort of data type that is capable of holding 64 bits of decimal numbers or floating points. Since double has more precision as compared to that float then it is much more obvious that it occupies twice the memory occupied by the floating-point type. It can easily accommodate about 16 to 17 digits after or before a decimal point.

Range: 1.7E-308 to 1.7E+308

Size: 8 bytes

Format Specifier: %lf

OUTPUT

```
123123123.000000
12.293123
2312312312.123123
```

CODE

```
#include <stdio.h>

int main()
{
    double a = 123123123.00;
    double b = 12.293123;
    double c = 2312312312.123123;

    printf("%lf\n", a);

    printf("%lf\n", b);

    printf("%lf", c);

    return 0;
}
```

Void Data Type

The void data type in C is used to specify that no value is present. It does not provide a result value to its caller. It has no values and no operations. It is used to represent nothing. Void is used in multiple ways as function return type, function arguments as void, and pointers to void.

CODE

```
#include <stdio.h>

int main()
{
    int val = 30;
    void* ptr = &val;
    printf("%d", *(int*)ptr);
    return 0;
}
```

OUTPUT

```
30
```

Size of Data Types in C

The size of the data types in C is dependent on the size of the architecture, so we cannot define the universal size of the data types. For that, the C language provides the sizeof() operator to check the size of the data types.

CODE

```
#include <stdio.h>

int main()
{
    int size_of_int = sizeof(int);
    int size_of_char = sizeof(char);
    int size_of_float = sizeof(float);
    int size_of_double = sizeof(double);

    printf("The size of int data type : %d\n", size_of_int);
    printf("The size of char data type : %d\n",
           size_of_char);
    printf("The size of float data type : %d\n",
           size_of_float);
    printf("The size of double data type : %d",
           size_of_double);

    return 0;
}
```

OUTPUT

```
The size of int data type : 4
The size of char data type : 1
The size of float data type : 4
The size of double data type : 8
```

Data Types in C

CHITKARA UNIVERSITY

Data Type	Size (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	1.2E-38 to 3.4E+38	%f
double	8	1.7E-308 to 1.7E+308	%lf
long double	16	3.4E-4932 to 1.1E+4932	%Lf

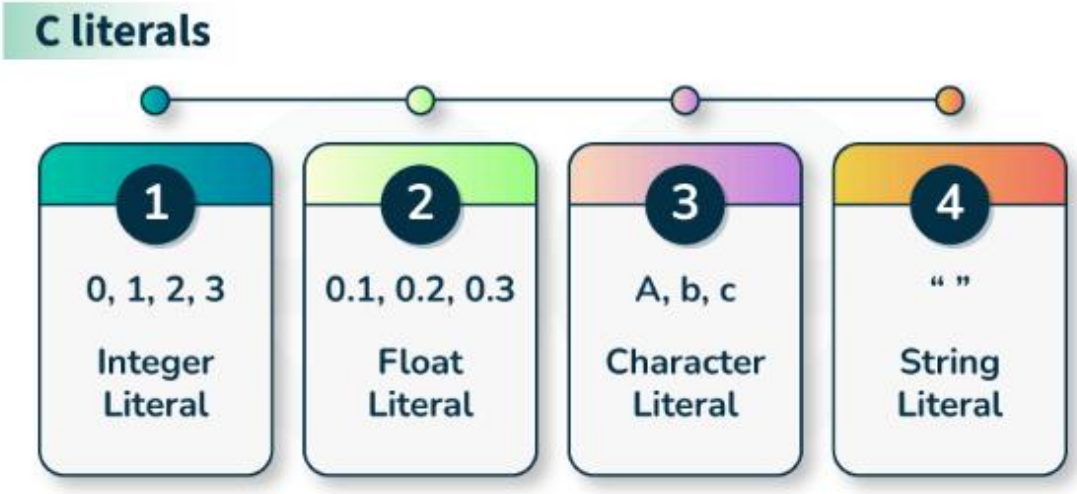
Literals in C

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. 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

There are 4 types of literal in C:

- **Integer Literal**
- **Float Literal**
- **Character Literal**
- **String Literal**



Integer Literals

Integer literals are used to represent and store the integer values only

CODE

```
#include <stdio.h>

int main()
{

    // constant integer literal
    const int intVal = 10;

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

OUTPUT

```
Integer Literal:10
```

Floating-Point Literals

CHITKARA UNIVERSITY

These are used to represent and store real numbers. The real number has an integer part, real part, fractional part, and exponential part. The floating-point literals can be stored either in decimal form or exponential form. While representing the floating-point decimals one must keep two things in mind to produce valid literal:

- In the decimal form, one must include the integer part, or fractional part, or both, otherwise, it will lead to an error.
- In the exponential form, one must include both the significand and exponent part, otherwise, it will lead to an error.

CODE

```
#include <stdio.h>

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

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

OUTPUT

```
Floating point literal: 4.14
```

Character Literals

This refers to the literal that is used to store a single character within a single quote. To store multiple characters, one needs to use a character array. Storing more than one character within a single quote will throw a warning and display just the last character of the literal. It gives rise to the following two representations:

- **char type:** This is used to store normal character literal or narrow-character literals.

CODE

```
#include <stdio.h>

int main()
{
    // constant char literal
    const char charVal = 'A';

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

OUTPUT

```
Character Literal: A
```

String Literals

String literals are similar to that character literals, except that they can store multiple characters and uses a double quote to store the same. It can also accommodate the special characters and escape sequences mentioned in the table above. We can break a long line into multiple lines using string literal and can separate them with the help of white spaces.

CODE

```
#include <stdio.h>

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

OUTPUT

```
Welcome
To
Chitkara University
```

Escape Sequence in C

CHITKARA UNIVERSITY

- The escape sequence in C is the characters or the sequence of characters that can be used inside the string literal. The purpose of the escape sequence is to represent the characters that cannot be used normally using the keyboard. Some escape sequence characters are the part of ASCII charset but some are not.

- Different escape sequences represent different characters but the output is dependent on the compiler you are using.

Escape Sequence in C

CHITKARA UNIVERSITY

Escape Sequence	Name	Description
\a	Alarm or Beep	It is used to generate a bell sound in the C program.
\b	Backspace	It is used to move the cursor one place backward.
\f	Form Feed	It is used to move the cursor to the start of the next logical page.
\n	New Line	It moves the cursor to the start of the next line.
\r	Carriage Return	It moves the cursor to the start of the current line.
\t	Horizontal Tab	It inserts some whitespace to the left of the cursor and moves the cursor accordingly.
\v	Vertical Tab	It is used to insert vertical space.
\\	Backlash	Use to insert backslash character.
\'	Single Quote	It is used to display a single quotation mark.
\"	Double Quote	It is used to display double quotation marks.
\?	Question Mark	It is used to display a question mark.
\ooo	Octal Number	It is used to represent an octal number.
\xhh	Hexadecimal Number	It represents the hexadecimal number.
\0	NULL	It represents the NULL character.

Escape Sequence in C

CODE

```
#include <stdio.h>

int main(void)
{
    // output may depend upon the compiler
    printf("My mobile number "
           "is 7\a8\a7\a3\a9\a2\a3\a4\a0\a8\a");
    return (0);
}
```

OUTPUT

```
My mobile number is 7873923408
```

Boolean in C

The bool in C is a fundamental data type in most that can hold one of two values: true or false. It is used to represent logical values and is commonly used in programming to control the flow of execution in decision-making statements such as if-else statements, while loops, and for loops.

In C, the bool data type is not a built-in data type. However, the C99 standard for C language supports bool variables. Boolean can store values as true-false, 0-1, or can be yes-no. It can be implemented in C using different methods as mentioned below:

1. Using header file “stdbool.h”
2. Using Enumeration type
3. Using define to declare boolean values

Using Header File “stdbool.h”

CHITKARA UNIVERSITY

CODE

```
#include <stdio.h>  // For printf()
#include <stdbool.h>  // For boolean data type (bool,
true, false)

// Main Function
int main() {
    // Boolean data types declared
    bool a = true;  // 'a' initialized to true
    bool b = false; // 'b' initialized to false

    // Printing Boolean values as integers (true = 1, false
= 0)
    printf("True : %d\n", a);
    printf("False : %d\n", b);

    return 0;
}
```

OUTPUT

```
True : 1
False : 0
```

NOTE: If we save the above program as a .c file, it will not compile. But if we save it as a .cpp file, it will work fine.

Using the Enumeration Type

Alternatively, you can implement bool in C using an enumeration type. Here rather than importing the library, we declare an enumeration type so as to use bool as the data type.

CODE

```
#include <stdio.h>

typedef enum { false, true } bool;

int main()
{
    bool a = true;
    bool b = false;

    printf("True : %d\n", a);
    printf("False : %d", b);

    return 0;
}
```

OUTPUT

```
True : 1
False : 0
```

Using Define to Declare Boolean Values

In this case, the false value is assigned the integer value of 0, and the true value is assigned the integer value of 1. You can also use an int or a char with a value of either 0 (false) or 1 (true) to represent the bool data type in C.

CODE

```
#define bool int
#define false 0
#define true 1

int main()
{
    bool a = true;
    bool b = false;

    printf("True : %d\n", a);
    printf("False : %d", b);

    return 0;
}
```

OUTPUT

```
True : 1
False : 0
```

Type conversion in C is the process of converting one data type to another. The type conversion is only performed to those data types where conversion is possible. Type conversion is performed by a compiler. In type conversion, the destination data type can't be smaller than the source data type. Type conversion is done at compile time and it is also called widening conversion because the destination data type can't be smaller than the source data type.

There are two types of Conversion:

1. *Implicit Type Conversion*
2. *Explicit Type Conversion*

Implicit Type Conversion

Also known as ‘automatic type conversion’.

1. Done by the compiler on its own, without any external trigger from the user.
2. Generally, takes place when in an expression more than one data type is present. In such conditions type conversion (type promotion) takes place to avoid loss of data.
3. All the data types of the variables are upgraded to the data type of the variable with the largest data type.
4. It is possible for implicit conversions to lose information, signs can be lost (when signed is implicitly converted to unsigned), and overflow can occur (when long is implicitly converted to float).

Implicit Type Conversion

CHITKARA UNIVERSITY

CODE

```
#include <stdio.h>
int main()
{
    int x = 10; // integer x
    char y = 'a'; // character c

    // y implicitly converted to int. ASCII
    // value of 'a' is 97
    x = x + y;

    // x is implicitly converted to float
    float z = x + 1.0;

    printf("x = %d, z = %f", x, z);
    return 0;
}
```

OUTPUT

```
x = 107, z = 108.000000
```

Explicit Type Conversion

This process is also called type casting and it is user-defined. Here the user can typecast the result to make it of a particular data type.

CODE

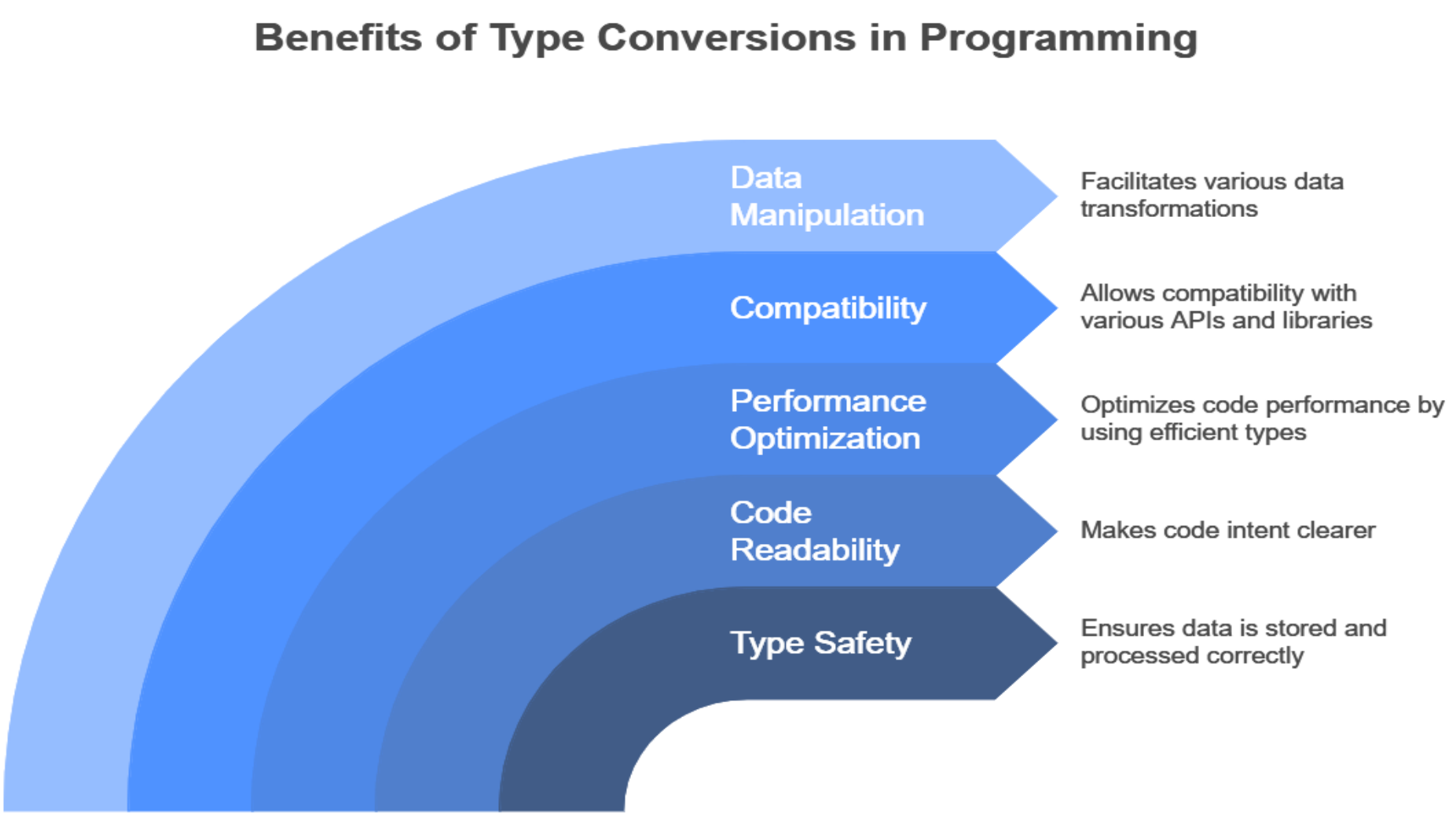
```
#include<stdio.h>

int main()
{
    double x = 1.2;

    // Explicit conversion from double to int
    int sum = (int)x + 1;

    printf("sum = %d", sum);

    return 0;
}
```

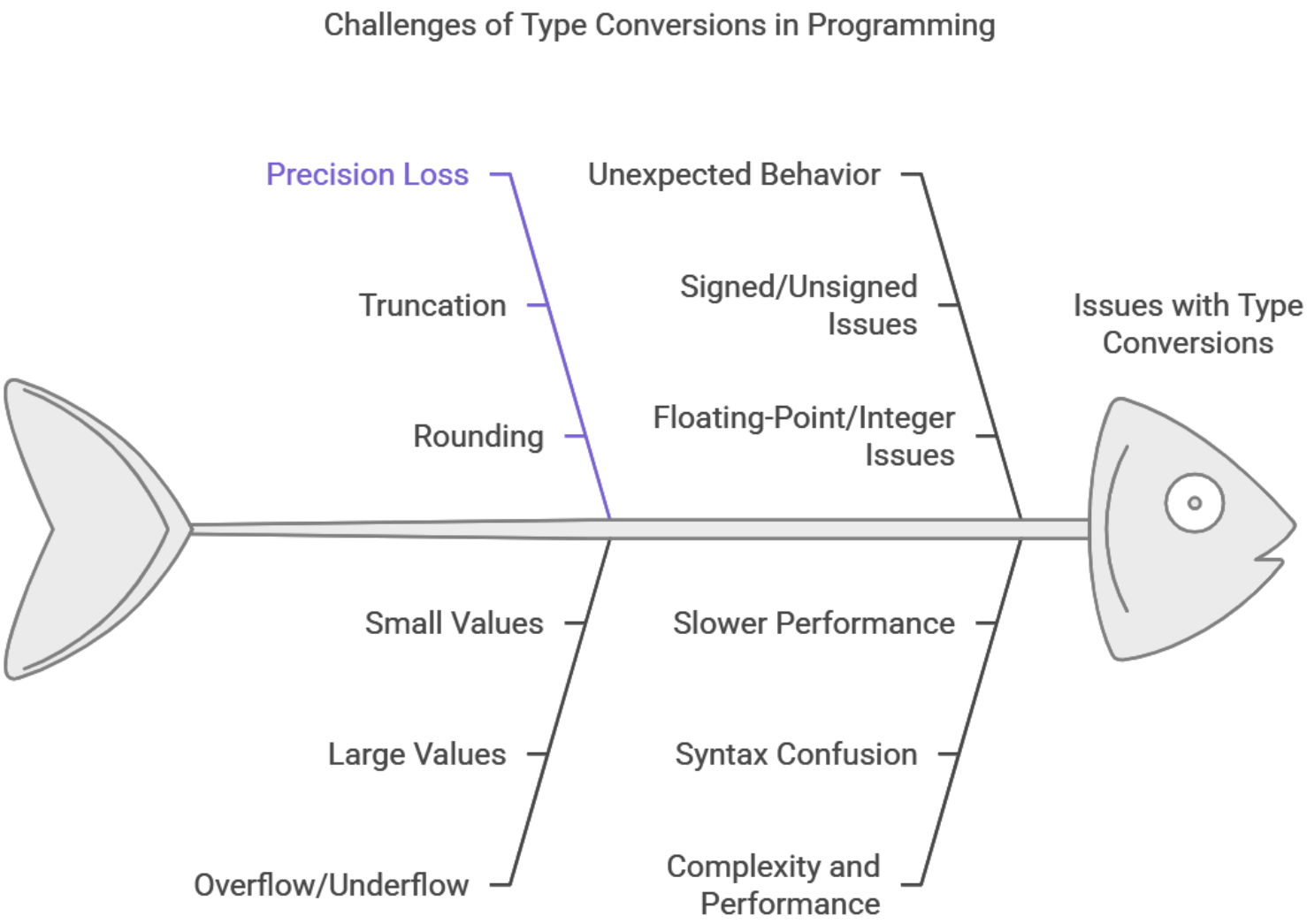
OUTPUT

```
sum = 2
```

Advantages of Type Conversion

CHITKARA UNIVERSITY




Challenges of Type Conversions in Programming
Precision Loss
Truncation
Rounding
Small Values
Large Values
Overflow/Underflow
Unexpected Behavior
Signed/Unsigned Issues
Floating-Point/Integer Issues
Slower Performance
Syntax Confusion
Complexity and Performance
Issues with Type Conversions

C language has standard libraries that allow input and output in a program. The **stdio.h** or **standard input output library** in C that has methods for input and output.

1. **scanf(): T**he scanf() method, in C, reads the value from the console as per the type specified and store it in the given address.

2. **printf():** The printf() method, in C, prints the value passed as the parameter to it, on the console screen.

Basic Input and Output in C

CHITKARA UNIVERSITY

```c
#include <stdio.h>

int main()
{

    // Declare the variables
    int num;
    char ch;
    float f;

    // --- Integer ---

    // Input the integer
    printf("Enter the integer: ");
    scanf("%d", &num);

    // Output the integer
    printf("\nEntered integer is: %d", num);

    // --- Float ---

    //For input Clearing buffer
      while((getchar()) != '\n');

    // Input the float
    printf("\n\nEnter the float: ");
    scanf("%f", &f);

    // Output the float
    printf("\nEntered float is: %f", f);

    // --- Character ---

    // Input the Character
    printf("\n\nEnter the Character: ");
    scanf("%c", &ch);

    // Output the Character
    printf("\nEntered character is: %c", ch);

    return 0;
}
```

OUTPUT

```
Enter the integer: 10
Entered integer is: 10

Enter the float: 2.5
Entered float is: 2.500000

Enter the Character: A
Entered Character is: A
```

The format specifier in C is used to tell the compiler about the type of data to be printed or scanned in input and output operations. They always start with a **%** symbol and are used in the formatted string in functions like printf(), scanf, sprintf(), etc.

The C language provides a number of format specifiers that are associated with the different data types such as %d for int, %c for char, etc.

Format Specifiers in C

CHITKARA UNIVERSITY

Format Specifier	Description
%c	For character type.
%d	For signed integer type.
%e or %E	For scientific notation of floats.
%f	For float type.
%g or %G	For float type with the current precision.
%i	signed integer
%ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

Character Format Specifier – %c in C

The %c is the format specifier for the **char** data type in C language. It can be used for both formatted input and formatted output in C language.

CODE

```
// C Program to illustrate the %c
format specifier.
#include <stdio.h>

int main()
{

    char c;
    // using %c for character input
    scanf("Enter some character: %c",
&c);

    // using %c for character output
    printf("The entered character: %c",
&c);
    return 0;
}
```

OUTPUT

```
Enter some character: A
```

```
The entered character: A
```

Integer Format Specifier (signed) – %d in C

We can use the signed integer format specifier %d in the scanf() and print() functions or other functions that use formatted string for input and output of int data type.

CODE

```
#include <stdio.h>

// Driver code
int main()
{
    int x;
    // taking integer input
    scanf("Enter the two integers: %d",
&x);

    // printing integer output
    printf("Printed using %%d: %d\n",
x);
    printf("Printed using %%i: %3i\n",
x);
    return 0;
}
```

OUTPUT

```
Enter the integer: 45
```

```
Printed using %d: 45
Printed using %i:    45
```

Unsigned Integer Format Specifier – %u in C

The %u is the format specifier for the unsigned integer data type. If we specify a negative integer value to the %u, it converts the integer to its 2's complement.

CODE

```
#include <stdio.h>

// driver code
int main()
{
    unsigned int var;

    scanf("Enter an integer: %u", &var);

    printf("Entered Unsigned Integer:
%u", var);

    // trying to print negative value
using %u
    printf("Printing -10 using %%u:
%u\n", -10);
    return 0;
}
```

OUTPUT

```
Enter an integer: 25
```

```
Entered unsigned integer: 25
Printing -10 using %u: 4294967286
```

Floating-point format specifier – %f in C

The **%f** is the floating point format specifier in C language that can be used inside the formatted string for input and output of **float** data type. Apart from %f, we can use **%e** or **%E** format specifiers to print the **floating point value in the exponential form.**

CODE

```
#include <stdio.h>

// driver code
int main()
{
    float a = 12.67;
    printf("Using %%f: %f\n", a);
    printf("Using %%e: %e\n", a);
    printf("Using %%E, %E", a);
    return 0;
}
```

OUTPUT

```
Using %f: 12.670000
Using %e: 1.267000e+01
Using %E, 1.267000E+01
```

Unsigned Octal number for integer – %o in C

CHITKARA UNIVERSITY

We can use the %o format specifier in the C program to print or take input for the unsigned octal integer number.

CODE

```
#include <stdio.h>
int main()
{
    int a = 67;
    printf("%o\n", a);
    return 0;
}
```

OUTPUT

```
103
```

Unsigned Hexadecimal for integer – %x in C

The %x format specifier is used in the formatted string for hexadecimal integers. In this case, the alphabets in the hexadecimal numbers will be in lowercase. For uppercase alphabet digits, we use %X instead.

CODE

```
#include <stdio.h>
int main()
{
    int a = 15454;
    printf("%x\n", a);
    printf("%X", a);
    return 0;
}
```

OUTPUT

```
3c5e
3C5E
```

String Format Specifier – %s in C

The %s in C is used to print strings or take strings as input.

CODE

```
#include <stdio.h>

int main()
{
    char a[] = "Hi ";
    printf("%s\n", a);
    return 0;
}
```

OUTPUT

```
Hi
```

In C language, printf() function is used to print formatted output to the standard output **stdout** (which is generally the console screen). The printf function is a part of the C standard library <**stdio.h**> and it can allow formatting the output in numerous ways.

Parameters

- **formatted_string:** It is a string that specifies the data to be printed. It may also contain a format specifier to print the value of any variable such as a character and an integer.
- **arguments_list:** These are the variable names corresponding to the format specifier.

Return Value

- printf() returns the number of characters printed after successful execution.
- If an error occurs, a negative value is returned.

CODE

```
#include <stdio.h>

int main()
{
    // using printf to print "Hello Geek!"
    printf("Hello");

    return 0;
}
```

OUTPUT

```
Hello
```

scanf in C

CHITKARA UNIVERSITY

In C programming language, scanf is a function that stands for Scan Formatted String. It is used to read data from stdin (standard input stream i.e. usually keyboard) and then writes the result into the given arguments.

- It accepts character, string, and numeric data from the user using standard input.

- scanf also uses format specifiers like printf.

CODE

```
#include <stdio.h>

// Driver code
int main()
{
    int a, b;

      printf("Enter first number: ");
      scanf("%d", &a);

      printf("Enter second number: ");
      scanf("%d", &b);

      printf("A : %d \t B : %d" ,
            a , b);

    return 0;
}
```

OUTPUT

```
Enter first number: 5
Enter second number: 6
A : 5       B : 6
```

CHITKARA UNIVERSITY

Q1

```
#include <stdio.h>
// Assume base address of “ABCDEQuiz" to be 1000
int main()
{
   printf(5 + “ABCDEQuiz");
   return 0;
}
```

1. ABCDEQuiz
2. **Quiz**
3. 1005
4. Compile-time error

Explanation

printf is a library function defined under *stdio.h* header file. The compiler adds 5 to the base address of the string through the expression ***5 + “ABCDEQuiz"*** . Then the string "Quiz" gets passed to the standard library function as an argument.

CHITKARA UNIVERSITY

Q2. What does the following C statement mean?

```
scanf("%4s", str);
```

1. Read exactly 4 characters from console.
2. **Read maximum 4 characters from console.**
3. Read a string str in multiples of 4
4. None

Q3. Which statement is used to indicate the end of a statement in C?

1. .
2. **;**
3. :
4. ,

Q4. Which datatype is used to represent a single character in C?

1. **char**
2. int
3. float
4. double

Q5. What is the correct syntax for declaring a variable in C?

1. **int variable_name;**
2. variable_name = 5;
3. variable_name int;
4. int = variable_name

CHITKARA UNIVERSITY

Thank You!