Module 1.4,1.5

Outline Module 1.4

- Header file
- Namespaces
- Data & Operators:
  - Identifier
  - Constants
  - Variables
  - Data Types

What is a Header File?

A header file contains:

- Function declarations
- Data Type definitions
- Constants & macros

In C++, all the header files may or may not end with the **".h"** extension unlike in C, Where all the header files must necessarily end with the **".h"** extension.

File extension: .h or standard library headers (<iostream>)

Syntax of Header Files in C++

```
#include <filename.h>  // for files already available in
system/default directory
or
#include "filename.h"  // for files in same directory as source file
```

"#include“- preprocessor directive are used to instruct the compiler that these files need to be processed before compilation

Example of inclusion of Header file

```cpp
// C++ program to demonstrate the inclusion of header file.

#include <cmath> // Including the standard header file for math operations
#include <iostream>
using namespace std;

int main()
{

    // Using functions from the included header file
    // (<cmath>)
    int sqrt_res = sqrt(25);
    int pow_res = pow(2, 3);

    // Displaying the results
    cout << "Square root of 25 is: " << sqrt_res << endl;
    cout << "2^3 (2 raised to the power of 3) is: "
         << pow_res << endl;

    return 0;
}
```

Output

```
Square root of 25 is: 5
2^3 (2 raised to the power of 3) is: 8
```

Types of Header Files in C++

- There are two types of header files in C++:
  - **Standard Header Files/Pre-existing header files**
  - **User-defined header files**

Standard Header Files/Pre-existing header files and their Uses

- Part of the C++ Standard Library and provide commonly used functionalities.
- Contains the declarations for standard functions, constants, and classes.

Header File	Description
`<iostream>`	It contains declarations for input and output operations using streams, such as `std::cout`, `std::cin`, and `std::endl`
`<cmath>`	It is used to perform mathematical operations like sqrt(), log2(), pow(), etc.
`<cstdlib>`	Declares functions involving memory allocation and system-related functions, such as `malloc()`, `exit()`, and `rand()`
`<cstring>`	It is used to perform various functionalities related to string manipulation like strlen(), strcmp(), strcpy(), size(), etc.

Example of inclusion of Standard header file

```
#include <iostream>

using namespace std;
int main()
 {
    int favorite_number;

    cout << "Enter your favorite number between 1 and 100: ";

    cin >> favorite_number;       cout << "Amazing!! That's my favorite number too!" << endl;
    cout << "No really!!, " << favorite_number << " is my favorite number!" << endl;

    return 0;
 }
```

User defined header file

- Writing a C++ source code with .h extension

```
// Function to find the sum of two
// numbers passed
int sumOfTwoNumbers(int a, int b) { return (a + b); }
```

- Include your header file with "#include" in your C++ program

```
// C++ program to find the sum of two
// numbers using function declared in
// header file
#include "iostream"

// Including header file
#include "sum.h"
using namespace std;

int main()
{

    // Given two numbers
    int a = 13, b = 22;

    // Function declared in header
    // file to find the sum
    cout << "Sum is: " << sumOfTwoNumbers(a, b) << endl;
}
```

Importance of header files

- Avoids code duplication
- Provides pre-written functions (e.g., cout, cin, sqrt())
- Increases modularity & readability
- Ensures faster development

Why Namespace?

- **Name Conflicts can occur in C++**
  - Same name used in different parts of a program
  - Occurs with variables, functions, or classes
  - Causes compiler confusion

Solution: C++ introduces **namespaces** to avoid such conflicts

Conflicts without using namespace

```
int value = 10;

int main() {
    int value = 20;   // Conflicts with global value
    cout << value;    // Ambiguity
}
```

What is Namespace?

- A namespace is a container for identifiers (variables, functions, classes).
- Used to group related code together.
- Prevents conflicts between identifiers with the same name.

Namespace Definition

- namespace definition begins with the keyword namespace followed by the namespace name

```
namespace name {
    // type1 mamber1
    // type2 mamber2
    // type3 mamber3
    .    .     .
    .    .     .
}
```

Example:

```
// Define a namespace called 'first_space'
namespace first_space {
    void func() {
        cout << "Inside first_space" << endl;
    }
}
```

Accessing Members in namespace

To access the members of namespace using scope resolution **operator(::)**.

Syntax:

namespace_name::member_name;

```cpp
#include <iostream>

// Define a namespace called 'first_space'
namespace first_space {
    void func() {
        std::cout << "Inside first_space" << std::endl;
    }
}

int main() {

    // Access member of namespace
    first_space::func();
    return 0;
}
```

Output

```
Inside first_space
```

Namespace with using Directive

- The **using namespace** directive

```
#include <iostream>

namespace first_space {
    void func() {
        std::cout << "Inside first_space"
            << std::endl;
    }
}

// Using first_space
using namespace first_space;

int main() {

    // Call the method of first_space
    func();
    return 0;
}
```

In-built Namespaces

std Namespace

- Part of the standard library contains most of the standard functions, objects, and classes like cin, cout, vector, etc.

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 3, b = 7;

    // 'cout' and 'endl' are part of the std namespace
    cout << "Sum: " << a + b ;
    return 0;
}
```

Use of namespaces

- Avoids Name Conflicts
  - Two libraries may use the same function/variable name.
- Organizes Code
  - Groups related classes & functions.
- Supports Modular Programming
  - Breaks large projects into logical parts.
- Readability & Maintainability
  - Clear scope of identifiers.

C++ First Program

```cpp
// Necessary header files for input output functions
#include <iostream>
using namespace std;

// main() function: where the execution of
// C++ program begins
int main() {

    // This statement prints "Hello World"
    cout << "Hello World";

    return 0;
}
```

Output

```
Hello World
```

C++ Comments

- Comments in C++ are meant to explain the code as well as to make it more readable.
- In C++ there are two types of comments:
  - Single Line Comment
  - Multi-Line Comment

Single Line Comment

- In C++, single line comments are represented as **// double forward slash**

Example:

```cpp
#include <iostream>
using namespace std;

int main() {

    // Single line comment
    cout << "GFG!";
    return 0;
}
```

Multi-Line Comment

- A multi-line comment can occupy many lines of code and starts with **/*** and ends with ***/**.

Example:

```
#include <iostream>
using namespace std;

int main() {

    /* Multi line comment which
       will be ignored by the compiler
    */
    cout << "GFG!";
    return 0;
}
```

C++ Tokens

- In C++, **tokens** can be defined as the smallest building block of C++ programs that the compiler understands

Identifiers in C++

In C++, names given to program entities like variables, functions, classes, or structs to uniquely identify are known as **identifiers.**

Valid identifier names:

totalCount, _privateVariable, data_1, calculateArea.

Invalid Identifier names:

1stNumber (starts with a digit), my Variable (contains a space), int (is a keyword), and item# (contains a special character).

Rules for Identifier names

- An identifier can only begin with a **letter or an underscore(_).**
- An identifier can consist of **letters** (A-Z or a-z), **digits** (0-9), and **underscores (_)**. White spaces and Special characters can not be used as the name of an identifier.
- Keywords cannot be used as an identifier because they are reserved words to do specific tasks. For example, **string**, **int, class**, **struct**, etc.
- Identifier must be **unique** in its namespace.
- As C++ is a case-sensitive language so identifiers such as 'first_name' and 'First_name' are different entities.

C++ Keywords

- **Keywords** are the **reserved words** that have special meanings in the C++ language.
- The total number of keywords in C++ are 93 up to C++ 23 specification

C++ Key words

alignas	alignof	and	and_eq	asm
auto	bitand	bitor	bool	break
case	catch	char	char8_t	char16_t
char32_t	class	compl	concept	const
consteval	constexpr	constinit	const_cast	continue
co_await	co_return	co_yield	decltype	default
delete	do	double	dynamic_cast	else
enum	explicit	export	extern	false
final	float	for	friend	goto
if	inline	int	long	mutable
namespace	new	noexcept	not	not_eq
nullptr	operator	or	or_eq	private
protected	public	register	reinterpret_cast	requires
return	short	signed	sizeof	static
static_assert	static_cast	struct	switch	template
this	thread_local	throw	true	try
typedef	typeid	typename	union	unsigned
using	virtual	void	volatile	wchar_t
while	xor	xor_eq		

Punctuators

Symbols which have specific meaning

- **Brackets** `[]` → Array element references (single/multi-dimensional)
- **Parentheses** `()` → Function calls & parameters
- **Braces** `{}` → Define a block of code
- **Comma** `,` → Separator (e.g., function parameters)
- **Colon** `:` → Initialization lists, inheritance
- **Semicolon** `;` → Statement terminator
- **Asterisk** `*` → Pointer declaration, multiplication
- **Assignment** `=` → Assign values
- **Pre-processor** `#` → Preprocessor directives (e.g., `#include`)
- **Dot** `.` → Access structure/union members
- **Tilde** `~` → Bitwise One's Complement Operator

C++ Variables

- **variable** is a name given to a memory location
- value stored in a variable can be accessed or changed during program execution.

Creating a Variable

Syntax:

type name;

Type name1, name2..;

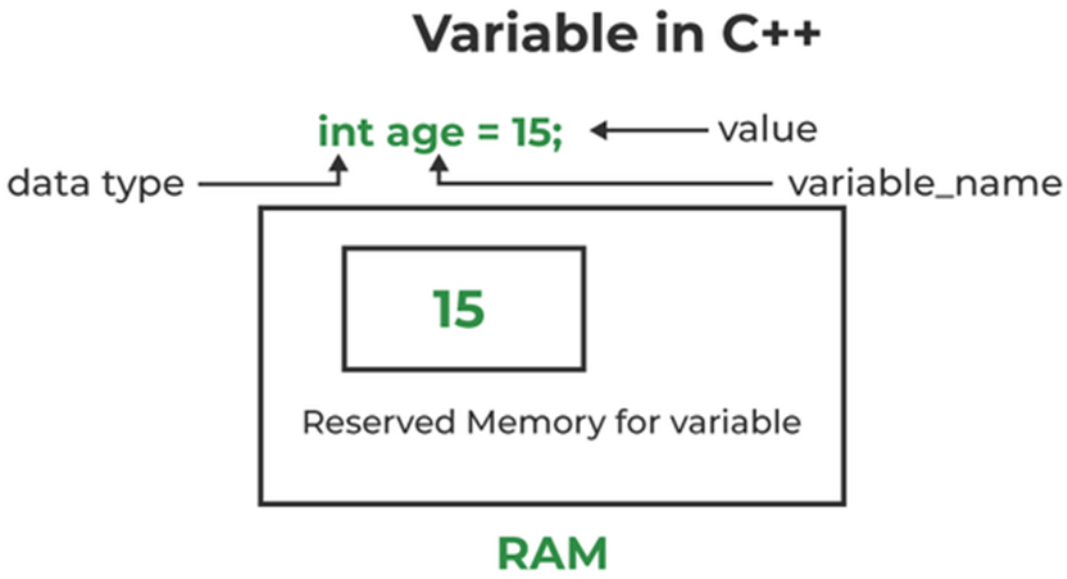
Example:

Int num;

Initializing variable

int num;

num = 3;



Accessing and Updating

```cpp
#include <iostream>
using namespace std;

int main() {

    // Creating a single character variable
    int num = 3;

    // Accessing and printing above variable
    cout << num << endl;

    // Update the value
    num = 7;

    cout << num;

    return 0;
}
```

Output

```
3
7
```

Constants in C++

- In C++, const is a keyword used to declare a variable as constant, meaning its value cannot be changed after it is initialized.
- Constants in C++ can be of various types such as int, float, char, or string.

```cpp
#include <iostream>
using namespace std;

int main() {

    // Declaring and defining a constant variable
    const int c = 24;

    cout << c;
    return 0;
}
```

Output

```
24
```

2 ways to declare a constant in c++

Using const Keyword

Syntax:

const DATATYPE variable_name = value;

```cpp
#include <iostream>
using namespace std;

int main() {
    int var = 10;

    // Declaring a constant variable
    const int c = 24;

    // Trying to change the value constant c
    c = 0;

    cout << c;
    return 0;
}
```

Output

```
./Solution.cpp: In function 'int main()':
./Solution.cpp:19:10: error: assignment of read-only variable 'cons'
     cons = 0;
```

2 ways to declare a constant in c++

- **Using #define Preprocessor**

Syntax

#define MACRO_NAME replacement_value

- Constants created using the #define preprocessor are called "macro constants"

Example

```
#include <iostream>
using namespace std;

// Using #define to create a macro
#define Side 5

int main()
{
    // Using constant
    double area = Side * Side;

    cout << area;
    return 0;
}
```

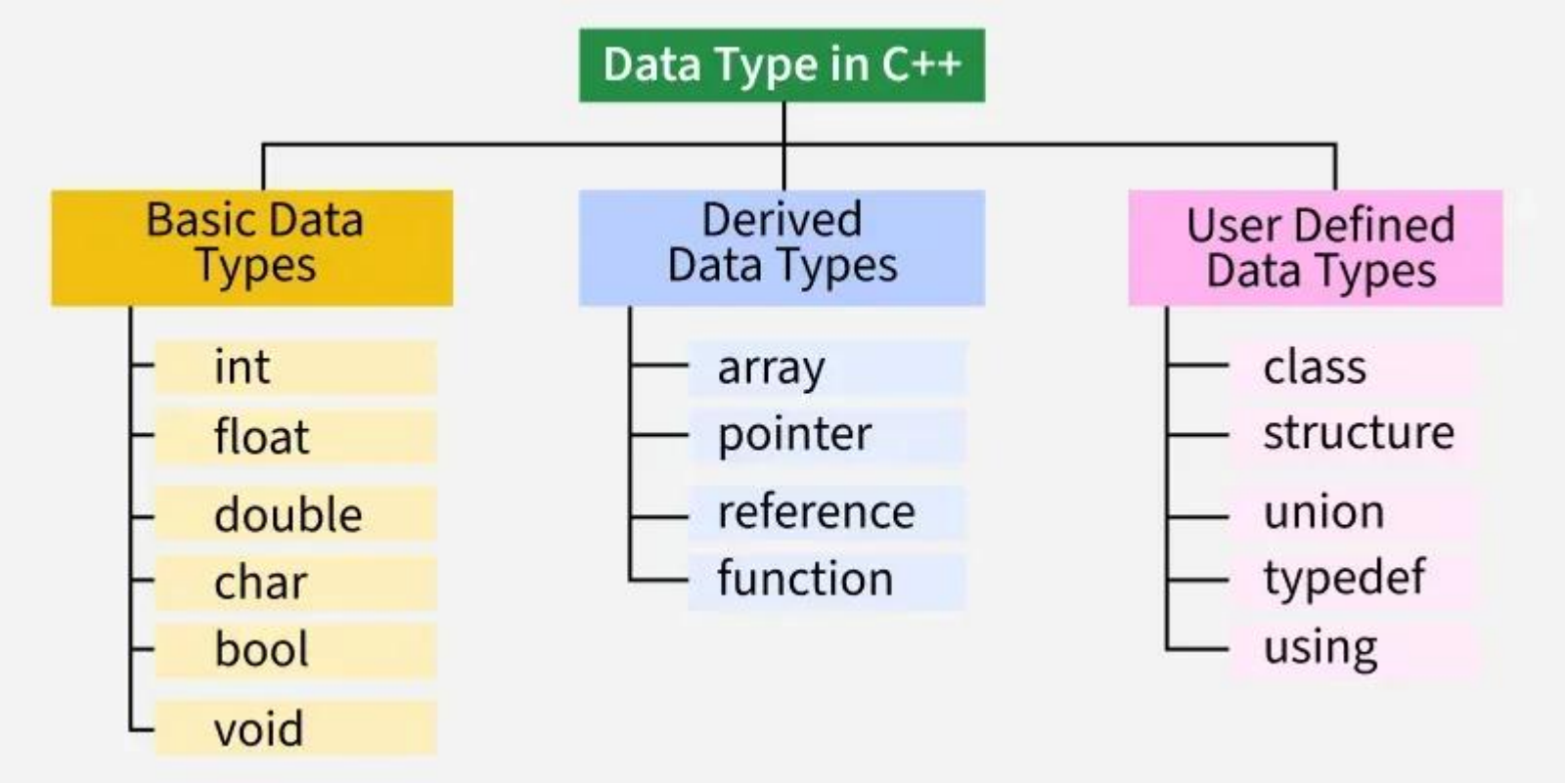
Output

```
25
```

C++ Data Types

- **Data types** specify the type of data that a variable can store.
- The compiler allocates some memory for that variable based on the data type
- C++ supports a wide variety of data types

Classification of Datatypes



Character Data Type (char)

- The **character data type** is used to store a single character.
- The keyword used to define a character is **char**.
- Its size is 1 byte, and it stores characters enclosed in single quotes (' ').
- It can generally store upto 256 characters according to their ASCII codes.

- **Syntax:**

Char variablename;

Example

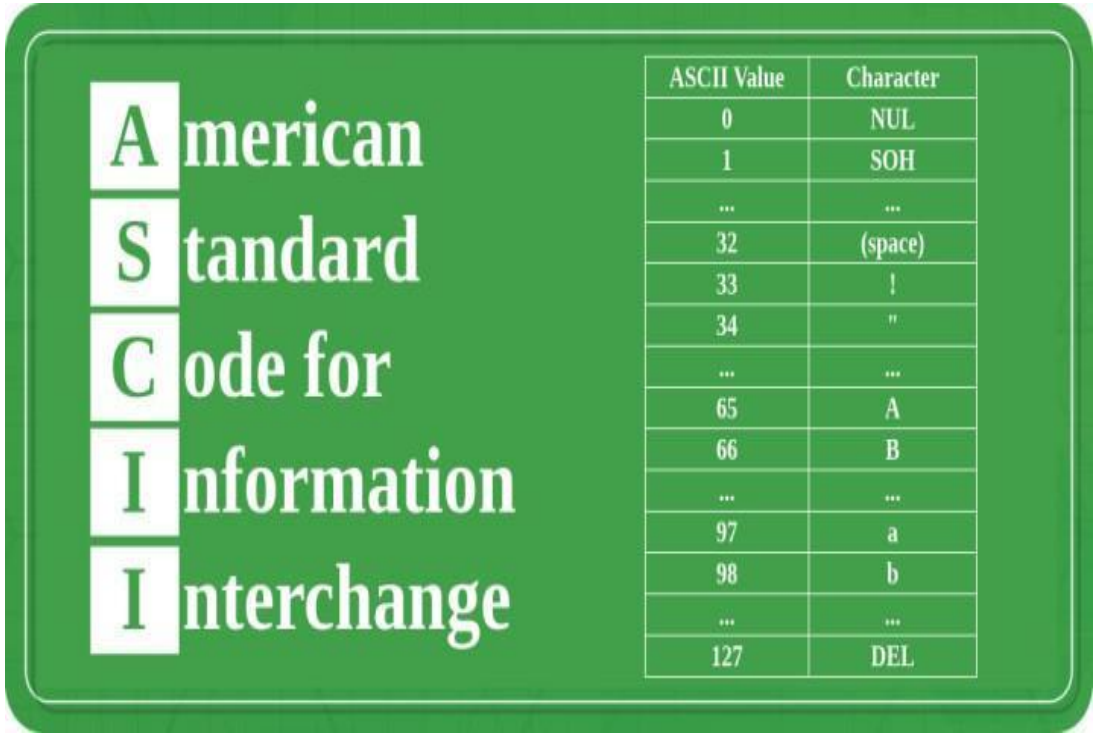
Example:

```cpp
// C++ Program demonstrate
// Use of char
#include <iostream>
using namespace std;

int main()
{
    char c = 'g';
    cout << c;
    return 0;
}
```

Output

```
g
```

ASCII codes



- Uses 7 bits to encode 128 characters (0–127); modern usage often stores them in 8-bit bytes with the high bit set to 0.
- 95 codes (32–126) for printable characters including space, digits, uppercase/lowercase English letters, punctuation, and symbols
- An 33 control characters that are non-printing (0–31, 127) for formatting and control (e.g., NUL, LF, CR)

ASCII code for characters

- *ASCII Range of 'a' to 'z' = 97-122*
- *ASCII Range of 'A' to 'Z' = 65-90*
- *ASCII Range of '0' to '9' = 48-57*

ASCII to Character and Vice-Versa

Example:

```cpp
// C++ Program to convert
// Char to ASCII value
#include <iostream>
using namespace std;

int main()
{
    char c = 'g';
    cout << "The Corresponding ASCII value of 'g'
    cout << int(c) << endl;

    c = 'A';
    cout << "The Corresponding ASCII value of 'A'
    cout << int(c) << endl;
    return 0;
}
```

Output

```
The Corresponding ASCII value of 'g' : 103
The Corresponding ASCII value of 'A' : 65
```

Example:

```cpp
// C++ Program to convert
// ASCII value to character
#include <iostream>
using namespace std;

int main()
{
    int x = 53;
    cout << "The Corresponding character value of x is : ";
    cout << char(x) << endl;

    x = 65;
    cout << "The Corresponding character value of x is : ";
    cout << char(x) << endl;

    x = 97;
    cout << "The Corresponding character value of x is : ";
    cout << char(x) << endl;
    return 0;
}
```

Output

```
The Corresponding character value of x is : 5
The Corresponding character value of x is : A
The Corresponding character value of x is : a
```

Integer Data Type (int)

- **Integer data type** variable can store the integer numbers.
- keyword **int** used to define integers **.**
- size is **4-bytes** (for 64-bit machine) systems
- range from **-2,147,483,648** to **2,147,483,647.**
- can store numbers for binary, octal, decimal and hexadecimal base systems

Example

```cpp
#include <iostream>
using namespace std;

int main() {

    // Creating an integer variable
    int x = 25;
    cout << x << endl;

    // Using hexadecimal base value
    x = 0x15;
    cout << x;

    return 0;
}
```

```
25
21
Process returned 0 (0x0)   execution time : 0.224 s
Press any key to continue.
```

Boolean Data Type (bool)

- **boolean data type** stores logical values: **true(1)** or **false(0)**.
- keyword **bool** define a boolean variable.
- size is 1 byte.

```cpp
#include <iostream>
using namespace std;

int main() {

    // Creating a boolean variable
    bool isTrue = true;
    cout << isTrue;
    return 0;
}
```

```
1
Process returned 0 (0x0)   e
Press any key to continue.
```

Floating Point Data Type (float)

- **Floating-point data type** store numbers with decimal points
- keyword **float** used to define floating-point numbers.
- size is 4 bytes (on 64-bit systems)
- store values in the range from **1.2e-38** to **3.4e+38.**

```cpp
#include <iostream>
using namespace std;

int main() {

    // Floating point variable with a decimal value
    float f = 36.5;
    cout << f;

    return 0;
}
```

```
36.5
Process returned 0 (0x0)
Press any key to continue.
```

Double Data Type (double)

- **double data type** is used to store decimal numbers with higher precision.
- keyword **double is** used to define double-precision floating-point numbers.
- size is 8 bytes (on 64-bit systems)
- range from **1.7e-308** to **1.7e+308**

```cpp
#include <iostream>
using namespace std;

int main() {

    // double precision floating point variable
    double pi = 3.1415926535;
    cout << pi;

    return 0;
}
```

```
3.14159
Process returned 0 (0x0)
Press any key to continue.
```

Void Data Type (void)

- **void data type** represents the absence of value.
- cannot create a variable of void type.
- used for pointer and functions that do not return any value defined using the keyword **void**.

Data Types in C

Type	Size	Range	Precision for real numbers
char	1 byte	-128 to 127	
unsigned char	1 byte	0 to 255	
signed char	1 byte	-128 to 127	
short int or short	2 bytes	-32,768 to 32,767	
unsigned short or unsigned short int	2 bytes	0 to 65535	
int	2 bytes	-32,768 to 32,767	
unsigned int	2 bytes	0 to 65535	
Long or long int	4 bytes	-2147483648 to 2147483647 (2.1 billion)	
unsigned long or unsigned long int	4 bytes	0 to 4294967295	
float	4 bytes	3.4 E–38 to 3.4 E+38	6 digits of precision
double	8 bytes	1.7 E-308 to 1.7 E+308	15 digits of precision
long double	10 bytes	+3.4 E-4932 to 1.1 E+4932	provides between 16 and 30 decimal places

Type Safety in C++

- C++ is a **strongly typed language (**all variables' data type specified at the declaration and does not change throughout the program)

```cpp
#include <iostream>
using namespace std;

int main() {

    // Assigning float value to boolean variable
    bool a = 10.248f;
    cout << a;
    return 0;
}
```

```
1
Process returned 0 (0x0)
Press any key to continue.
```

Floating-point value is not stored in the bool variable a

Data Type Conversion

- Type conversion refers to the process of changing one data type into another compatible one without losing its original meaning
- **C Style Typecasting**

(type) expression;

where type indicates the data type to which the final result is converted.

```cpp
#include <iostream>
using namespace std;

int main() {
    double x = 1.2;

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

    cout << sum;

    return 0;
}
```

```
2
Process returned 0 (0x0)
Press any key to continue
```

Implicit type casting

```cpp
#include <iostream>
using namespace std;

int main() {

    int i = 10;
    char c = 'a';
    // c implicitly converted to int. ASCII
    // value of 'a' is 97
    i = i + c;

    // x is implicitly converted to float
    float f = i + 1.0;

    cout << "i = " << i << endl
         << "c = " << c << endl
         << "f = " << f;

    return 0;
}
```

```
i = 107
c = a
f = 108
Process returned 0 (0x0)
Press any key to continue.
```

Module 1.5

- Types of Operators
- Expressions and Evaluation of Expressions,
- Operator Precedence and Associativity
- Type Conversions

Operators in C++

- C++ operators are the symbols that operate on values to perform specific mathematical or logical computations on given values
- C++ Operator Types
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Ternary or Conditional Operators

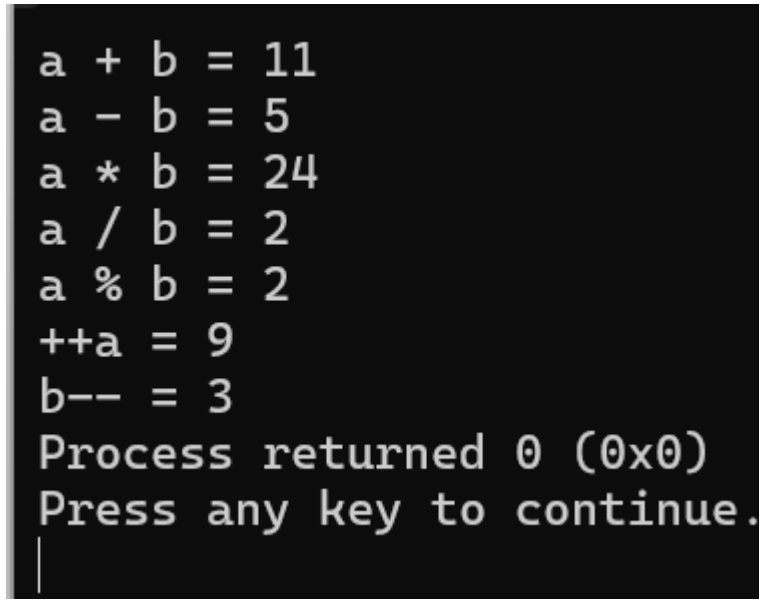
Arithmetic Operators

- **Arithmetic operators** are used to perform arithmetic or mathematical operations on the operands

Name	Symbol	Description
Addition	+	Adds two operands.
Subtraction	-	Subtracts second operand from the first.
Multiplication	*	Multiplies two operands.
Division	/	Divides first operand by the second operand.
Modulo Operation	%	Returns the remainder an integer division.
Increment	++	Increase the value of operand by 1.
Decrement	--	Decrease the value of operand by 1.

Example

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 8, b = 3;

    // Addition
    cout << "a + b = " << (a + b) << endl;

    // Subtraction
    cout << "a - b = " << (a - b) << endl;

    // Multiplication
    cout << "a * b = " << (a * b) << endl;

    // Division
    cout << "a / b = " << (a / b) << endl;

    // Modulo
    cout << "a % b = " << (a % b) << endl;

    // Increament
    cout << "++a = " << ++a << endl;

    // Decrement
    cout << "b-- = " << b--;

    return 0;
}
```



% and post-increment and pre-increment operator

- Modulo operator (%) operator should only be used with integers..
- ++a and a++, both are increment operators
- ++a, the value of the variable is incremented first and then it is used in the program.
- a++, the value of the variable is assigned first and then it is incremented.

Example

```cpp
#include <iostream>

using namespace std;

int main() {

    // Two integer variables
    int a = 10;
    int b, c;

    // Prefix decrement: first, decrement a, then
    // assign to b
    b = --a;

    // Postfix decrement: first, assign a to c,
    // then decrement a
    c = a--;

    cout << "a: " << a << ", b: " << b << ", c: " << c;

    return 0;
}
```

```
a: 8, b: 9, c: 9
Process returned 0 (0x0)   execution
Press any key to continue.
```

Relational Operators

- **Relational operators** are used for the comparison of the values of two operands.

Name	Symbol	Description
Is Equal To	==	Checks both operands are equal
Greater Than	>	Checks first operand is greater than the second operand
Greater Than or Equal To	>=	Checks first operand is greater than equal to the second operand
Less Than	<	Checks first operand is lesser than the second operand
Less Than or Equal To	<=	Checks first operand is lesser than equal to the second operand
Not Equal To	!=	Checks both operands are not equal

Example

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 6, b = 4;

    // Equal operator
    cout << "a == b is " << (a == b) << endl;

    // Greater than operator
    cout << "a > b is " << (a > b) << endl;

    // Greater than Equal to operator
    cout << "a >= b is " << (a >= b) << endl;

    //  Lesser than operator
    cout << "a < b is " << (a < b) << endl;

    // Lesser than Equal to operator
    cout << "a <= b is " << (a <= b) << endl;

    // Not equal to operator
    cout << "a != b is " << (a != b);

    return 0;
}
```

```
a == b is 0
a > b is 1
a >= b is 1
a < b is 0
a <= b is 0
a != b is 1
Process returned 0 (0x0)
Press any key to continue.
```

0 denotes false and 1 denotes true.

Logical Operators

- Logical operators are used to combine two or more conditions or constraints or to complement the evaluation of the original condition in consideration.
- result returns a Boolean value, i.e., true or false.

Name	Symbol	Description
Logical AND	&&	Returns true only if all the operands are true or non-zero.
Logical OR	||	Returns true if either of the operands is true or non-zero.
Logical NOT	!	Returns true if the operand is false or zero.

Example:

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 6, b = 4;

    // Logical AND operator
    cout << "a && b is " << (a && b) << endl;

    // Logical OR operator
    cout << "a || b is " << (a || b) << endl;

    // Logical NOT operator
    cout << "!b is " << (!b);

    return 0;
}
```

```
a && b is 1
a || b is 1
!b is 0
Process returned 0 (0x0)
Press any key to continue.
```

Example:

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 10, b = 20;

    if (a > 0 && b > 0) {
        cout << "Both values are greater than 0" << endl;
    } else {
        cout << "Both values are less than 0" << endl;
    }

    return 0;
}
```

```
Both values are greater   than 0

Process returned 0 (0x0)   execu
Press any key to continue.
```

Bitwise Operators

- **Bitwise operators** are works on bit-level.
- compiler first converts to bit-level and then the calculation is performed on the operands.

Name	Symbol	Description
Binary AND	&	Copies a bit to the evaluated result if it exists in both operands
Binary OR	|	Copies a bit to the evaluated result if it exists in any of the operand
Binary XOR	^	Copies the bit to the evaluated result if it is present in either of the operands but not both
Left Shift	<<	Shifts the value to left by the number of bits specified by the right operand.
Right Shift	>>	Shifts the value to right by the number of bits specified by the right operand.
One's Complement	~	Changes binary digits 1 to 0 and 0 to 1

Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows −

		AND	OR	Exclusive OR EXOR
p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Example Bitwise Operators

Assume A = 60 and B = 13 in binary format, they will be as follows −

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

Bitwise Operators

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) = 12, i.e., 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) = 61, i.e., 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) = 49, i.e., 0011 0001
~	Binary One's Complement Operator is unary and has the effect of 'flipping' bits.	(~A) = ~(60), i.e,. -0111101
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 = 240 i.e., 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 = 15 i.e., 0000 1111

```cpp
int main() {
    int a = 60;    // 60 = 0011 1100
    int b = 13;    // 13 = 0000 1101
    int c = 0;

    // Bitwise AND
    c = a & b;     // 12 = 0000 1100
    cout << "Line 1 - Value of c (a & b) is " << c << endl;

    // Bitwise OR
    c = a | b;     // 61 = 0011 1101
    cout << "Line 2 - Value of c (a | b) is " << c << endl;

    // Bitwise XOR
    c = a ^ b;     // 49 = 0011 0001
    cout << "Line 3 - Value of c (a ^ b) is " << c << endl;

    // Bitwise NOT
    c = ~a;        // -61 (two's complement of 60)
    cout << "Line 4 - Value of c (~a) is " << c << endl;

    // Left Shift
    c = a << 2;    // 240 = 1111 0000
    cout << "Line 5 - Value of c (a << 2) is " << c << endl;

    // Right Shift
    c = a >> 2;    // 15 = 0000 1111
    cout << "Line 6 - Value of c (a >> 2) is " << c << endl;

    return 0;
```

```
Line 1 - Value of c (a & b) is 12
Line 2 - Value of c (a | b) is 61
Line 3 - Value of c (a ^ b) is 49
Line 4 - Value of c (~a) is -61
Line 5 - Value of c (a << 2) is 240
Line 6 - Value of c (a >> 2) is 15
```

Only char and int data types can be used with Bitwise Operators.

```cpp
#include <iostream>
using namespace std;

#include <iostream>
using namespace std;

int main() {
    int a = 6, b = 4;

    // Binary AND operator
    cout << "a & b is " << (a & b) << endl;

    // Binary OR operator
    cout << "a | b is " << (a | b) << endl;

    // Binary XOR operator
    cout << "a ^ b is " << (a ^ b) << endl;

    // Left Shift operator
    cout << "a << 1 is " << (a << 1) << endl;

    // Right Shift operator
    cout << "a >> 1 is " << (a >> 1) << endl;

    // One's Complement operator
    cout << "~(a) is " << ~(a);

    return 0;
}
```

```
a & b is 4
a | b is 6
a ^ b is 2
a << 1 is 12
a >> 1 is 3
~(a) is -7
Process returned 0 (0x0)
Press any key to continue.
```

Binary 1’s compilement operator in c++

- **Step 1**: Decimal → Binary
  60 = 0011 1100 (8 bit)
- **Step 2**: Apply Bitwise Complement (~)
  ~0011 1100 = 1100 0011

  This in decimal is 195

  But in C++ when using ~ (bitwise NOT), it gives result in 2’s complement signed integer representation.

- **Step 3**: Interpret Result (Two’s Complement)
  1100 0011 → Negative number
  Invert: 0011 1100 → Add 1 → 0011 1101 = 61
  Final value = -61

Demo

```cpp
#include <iostream>
#include <bitset>   // for binary representation
using namespace std;

int main() {
    unsigned int x = 60;
    cout << "Decimal: " << x << endl;
    cout << "Binary : " << bitset<8>(x) << endl;  // 8-bit binary
    cout << "Complement (~x): " << bitset<8>(~x) << endl;
     cout << "Complement (~x): " << bitset<16>(~x) << endl;


    return 0;
}
```

```
Decimal: 60
Binary : 00111100
Complement (~x): 11000011
Complement (~x): 1111111111000011
```

```
Decimal: 60
Binary : 00111100
1's Complement (~num): 4294967235
```

Assignment Operators

- **Assignment operators** are used to assign value to a variable

Name	Symbol	Description
Assignment	=	Assigns the value on the right to the variable on the left.
Add and Assignment	+=	First add right operand value into left operand then assign that value into left operand.
Subtract and Assignment	-=	First subtract right operand value into left operand then assign that value into left operand.
Multiply and Assignment	*=	First multiply right operand value into left operand then assign that value into left operand.
Divide and Assignment	/=	First divide right operand value into left operand then assign that value into left operand.

Example

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 6, b = 4;

    // Assignment Operator.
    cout << "a = " << a << endl;

    //  Add and Assignment Operator.
    cout << "a += b is " << (a += b) << endl;

    // Subtract and Assignment Operator.
    cout << "a -= b is " << (a -= b) << endl;

    //  Multiply and Assignment Operator.
    cout << "a *= b is " << (a *= b) << endl;

    //  Divide and Assignment Operator.
    cout << "a /= b is " << (a /= b);

    return 0;
}
```

```
a = 6
a += b is 10
a -= b is 6
a *= b is 24
a /= b is 6
```

Ternary or Conditional Operators

- Ternary operator returns the value, based on the condition.
- Ternary operator takes three operands

Synatax:

Expression1 ? Expression2 : Expression3

- The ternary operator **?** determines the answer on the basis of the evaluation of **Expression1**.
- If **Expression1** is true, then **Expression2** gets evaluated.
- If **Expression1** is false, then **Expression3** gets evaluated.

Example Ternary operator

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 3, b = 4;

    // Conditional Operator
    int result = (a < b) ? b : a;
    cout << "The greatest number "
         "is " << result;

    return 0;
}
```

```
The greatest number is 4
Process returned 0 (0x0)
Press any key to continue.
```

sizeof Operator

- sizeof operator is a unary operator used to compute the size of its operand or variable in bytes

sizeof (char);

sizeof (var_name);

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 4;

    double c;

    // Example of sizeof operator
    cout << "Line 1 - Size of variable a = " << sizeof(a) << " bytes" << endl;

    cout << "Line 2 - Size of variable c = " << sizeof(c) << " bytes" << endl;

    return 0;
}
```

```
Line 1 - Size of variable a = 4 bytes
Line 2 - Size of variable c = 8 bytes
```

Addressof Operator (&)

- Addressof operator is used to find the memory address in which a particular variable is stored

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 4;
    int* ptr;

    // Example of & and * operators
    ptr = &a;   // 'ptr' now contains the address of 'a

    cout << "Value of address of a is " << ptr << endl;
    cout << "Value of a is " << a << endl;
    cout << "*ptr is " << *ptr << endl;

    return 0;
}
```

```
Value of address of a is 0x61fe44
Value of a is 4
*ptr is 4
```

Type Conversions: Precedence of operators

Priority	Operators	Description
1st	* / %	multiplication. division. modular division
2nd	+ -	addition, subtraction
3rd	=	assignment

Example 1.1: Determine the hierarchy of operations and evaluate the following expression:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

Stepwise evaluation of this expression is shown below:

i = 2 * 3 / 4 + 4 / 4 + 8 - 2 + 5 / 8

i = 6 / 4 + 4 / 4 + 8 - 2 + 5 / 8 operation: *
i = 1 + 4 / 4 + 8 - 2 + 5 / 8 operation: /
i = 1 + 1+ 8 - 2 + 5 / 8 operation: /
i = 1 + 1 + 8 - 2 + 0 operation: /
i = 2 + 8 - 2 + 0 operation: +
i = 10 - 2 + 0 operation: +
i = 8 + 0 operation : -
i = 8 operation: +

Operator Precedency and Associativity

- When there are multiple operators in a single expression, operator precedency and associativity decide in which order and which part of expression are calculate.
- Precedency tells which part of expression should be calculate first
- Associativity tells which direction to solve when same precedency operators are in

Example:

3 * 2 + 8

Will be evaluated as:

(3 * 2) + 8 = 14.

Example

```cpp
#include <iostream>
using namespace std;

int main() {
    int result = 50 / 25 * 2;


    cout << "Result = " << result << endl;

    return 0;
}
```

```
Result = 4
```

Basic Input / Output in C++

- In C++, input and output are performed as sequences of bytes, called **streams**.
- **Input Stream**: Flow of bytes from device (e.g., keyboard) → main memory.
- **Output Stream**: Flow of bytes from main memory → device (e.g., display screen).
- Streams are defined in the **<iostream>** header file, which provides standard input/output tools.
- **cin** (instance of iostream) → used for input.
- **cout** (instance of iostream) → used for output.

Standard Output Stream - cout

- **cout** in C++ is an instance of the ostream class.
- It is used to produce output on the standard output device (usually the display screen).
- Data to be displayed is sent to cout using the insertion operator (<<).
- **Syntax:**

  cout << value/variable;

```
#include <iostream>
using namespace std;

int main() {

    // Printing the given text using cout
    cout << "Hello World!!";
    cout << 6;
    return 0;
}
```

```
Hello World!!6
```

In the program, cout is used to output the text “Hello World!!" and value 6 to the standard output stream. The insertion operator (<<) to send the specified data to the output stream.

Example:

```cpp
#include <iostream>
using namespace std;

int main() {
    int a = 22;

    // Printing variable 'a' using cout
    cout << a;
    return 0;
}
```

```
22
Process returned 0 (0x0)
Press any key to continue.
```

Standard Input Stream - cin

- **cin** in C++ is an instance of the **istream** class.
- It is used to take input from the standard input device (usually the keyboard).
- Data entered is extracted from cin using the extraction operator (**>>**).

Syntax

cin >> variable;

Example

```cpp
#include <iostream>
using namespace std;

int main() {
    int age;
    // Taking input from user and store  in variable
      cin >> age;
    // Output the entered age
    cout << "Age entered: " << age;
    return 0;
}
```

```
34
Age entered: 34
```

The above program asks the user to input the **age**. The object **cin** is connected to the input device (keyboard). The age entered by the user is extracted from **cin** using the **extraction operator(>>)** and the extracted data is then stored in the variable **age** present on the right side of the extraction operator.

Example:

```cpp
#include <iostream>
using namespace std;

int main() {
    string name;
    cin >> name;
    cout << "Name entered: " << name;
    return 0;
}
```

```
Vishal Kumar
Name entered: Vishal
```

cin stops reading input as soon as it encounters a whitespace (space, tab, or newline). It only captures the first word or characters until the first whitespace

std::endl in C++

- Belongs to the <iostream> library.
- Used with output streams (cout).
- Purpose:
  - Inserts a newline character (\n) into the output
  - Flushes the output buffer (ensures data is written immediately).

```
cout << "Hello" << std::endl;
```

Or

```
cout << "Hello" << endl;
```

Difference from \n:

- \n → only moves to a new line.
- std::endl → moves to a new line and flushes the buffer.

Program to accept Name and Age and display

```cpp
#include <iostream>
using namespace std;

int main()
{
    string name;
    int age;

    cin >> name >> age;
    cout << "Name : " << name << endl;
    cout << "Age : " << age << endl;
    return 0;
}
```

```
ABC
65
Name : ABC
Age : 65
```

Program to add 2 numbers

```cpp
#include <iostream>
using namespace std;

int main() {
    int a , b ;
    cout<<"Enter the two Numbers: ";
    cin>> a>>b;
    cout << "Sum of Two numbers = "<<a + b;
    return 0;
}
```

```
Enter the two Numbers: 34 56
Sum of Two numbers = 90
```

WAP to swap 2 numbers using temporary variable

Program to swap 2 numbers using temporary variable

```cpp
#include <iostream>
using namespace std;

int main() {
    int a, b, temp;

    cout << "Enter first number: ";
    cin >> a;
    cout << "Enter second number: ";
    cin >> b;
    // Swapping using temporary variable
    temp = a;
    a = b;
    b = temp;

    cout << "After Swapping: a = " << a << ", b = " << b << endl;
    return 0;
}
```

```
Enter first number: 22
Enter second number: 99
After Swapping: a = 99, b = 22
```

WAP to swap 2 numbers without using temporary variable

```cpp
#include<iostream>
using namespace std;

int main()
{
    int a, b;
     cout << "Enter first number: ";
     cin >> a;
     cout << "Enter second number: ";
     cin >> b;

     b = a + b;
     a = b - a; //original b restored in a
     b = b - a; //original a restored in b
     cout << "After swapping a = " << a << " , b = " << b
          << endl;
     return 0;
}
```

```
Enter first number: 89
Enter second number: 45
After swapping a = 45 , b = 89
```

WAP to display area and perimeter of Rectangle given the sides

Program to display area and perimeter of Rectangle given the sides

```cpp
#include <iostream>
using namespace std;

int main() {
    float length, breadth, area, perimeter;

    cout << "Enter length of rectangle: ";
    cin >> length;
    cout << "Enter breadth of rectangle: ";
    cin >> breadth;

    area = length * breadth;
    perimeter = 2 * (length + breadth);


    cout << "\nArea of Rectangle = " << area << endl;
    cout << "Perimeter of Rectangle = " << perimeter << endl;

    return 0;
}
```

```
Enter length of rectangle: 5
Enter breadth of rectangle: 13

Area of Rectangle = 65
Perimeter of Rectangle = 36
```

WAP to Check the number is odd or even

- Use ternatory operator

Program to Check the number is odd or even

```cpp
#include <iostream>
using namespace std;

int main()
 {
    int number;
    cout << "Enter a number: ";
    cin >> number;

    // One-line condition using ternary operator
    cout << number << " is " << ((number % 2 == 0) ? "Even." : "Odd.");
    return 0;
}
```

```
Enter a number: 177
177 is Odd.
```

Try using bitwise operator

Solution using bitwise operator

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    cout << (n & 1 ? "Odd" : "Even");
    return 0;
}
```

WAP to display maximum of 2 numbers

- Use ternary operator

Solution

```cpp
#include <iostream>
using namespace std;

int main() {
    int a, b;
    cout << "Enter two numbers: ";
    cin >> a >> b;

    cout << "Maximum = " << (a > b ? a : b);
    return 0;
}
```

```
Enter two numbers: 67 -99
Maximum = 67
```

Program to check whether a number is positive, negative or zero

Program to check whether a numbe is positive, negative or zero

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;

    cout << (n > 0 ? "Positive" : (n < 0 ? "Negative" : "Zero"));
    return 0;
}
```

```
Enter a number: -90
Negative
```

WAP to check maximum of 3 numbers

Program to check maximum of 3 numbers

```cpp
#include <iostream>
using namespace std;

int main() {
    int a, b, c;
    cout << "Enter three numbers: ";
    cin >> a >> b >> c;

    int max = (a > b ? (a > c ? a : c) : (b > c ? b : c));
    cout << "Maximum = " << max;
    return 0;
}
```

```
Enter three numbers: -23 -45 -87
Maximum = -23
```