

Introduction to Core C++ Concepts

Chapter One
Piyush Pant

Overview of Topics

1.4 C++ Program Structure

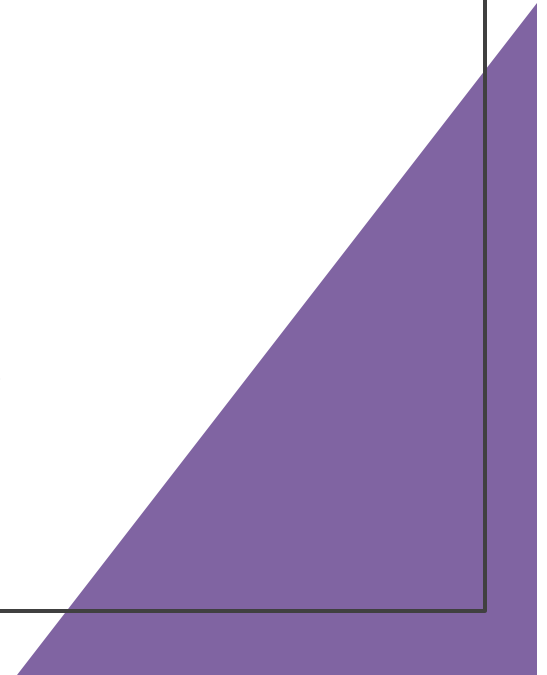
1.5 Data Types, Variables and Constants

1.6 Insertion and Extraction Operators

1.7 Type Conversion

1.8 Structure in C++

1.9 Dynamic memory allocation : new and delete operator



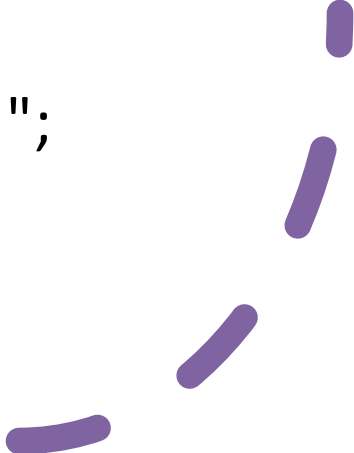
Basic C++ Program Structure

A typical C++ program includes:

- Preprocessor directives
- Main function
- Statements and expressions inside curly braces

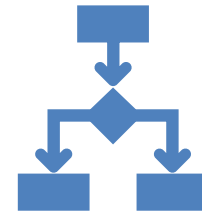
Example:

```
#include <iostream>
int main() {
    std::cout << "Hello, World!";
    return 0;
}
```



Preprocessor Directive

- A **preprocessor directive** in C++ is a command that starts with a # symbol and is processed **before the actual compilation** of the code begins.
- It tells the **C++ preprocessor** to perform specific actions like including files, defining constants, or conditionally compiling code.



Preprocessor Directives

- Begin with '#'
- Commonly used: #include, #define
- #include <iostream> includes standard input/output stream library
- #define is used for macros , compile time constant (#define PI 3.14159)

Common Preprocessor Directives

Directive	Purpose
#include	Includes the content of another file (usually header files).
#define	Defines macros or symbolic constants.
#undef	Undefines a macro defined with #define.
#ifdef / #ifndef	Checks if a macro is defined or not.
#if, #else, #elif, #endif	Conditional compilation.

Main Function and Execution Flow

- Entry point of every C++ program: `int main()`
- Code within `main()` executes sequentially
- Return value 0 signifies successful execution

Namespaces in C++

- A **namespace** is a way to group **related variables, functions, classes, etc.** under a single name to avoid **name conflicts** in large projects.
- Suppose two libraries have a function called `print()`. Without namespaces, the compiler won't know which one you mean. Namespaces **help avoid conflicts**.



Namespace

Defining

```
namespace MyNamespace {  
    int value = 100;  
  
    void show() {  
        std::cout << "Value: " << value <<  
std::endl;  
    }  
}
```

Using

1. Using Scope Resolution Operator ::

```
MyNamespace::show();  
std::cout << MyNamespace::value;
```

2. Using Directive (using namespace)

```
using namespace MyNamespace;
```

```
show();    // No need to prefix with namespace  
std::cout << value;
```

Namespace (Function example)

```
#include <iostream>

namespace English {
    void greet() {
        std::cout << "Hello!" << std::endl;
    }
}

namespace Spanish {
    void greet() {
        std::cout << "¡Hola!" << std::endl;
    }
}

int main() {
    English::greet(); // Outputs: Hello!
    Spanish::greet(); // Outputs: ¡Hola!
    return 0;
}
```



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

```
int main() {
    cout << "Welcome to C++";
    return 0;
}
```

- Include necessary library
- Uses namespace to avoid std::
- Outputs message using cout

Sample Program

Comments

Comments are used to explain code.
They do not affect the execution of the program.
Help improve code readability.

Single-line Comment:

Syntax: `// comment`

Example: `// This is a comment`

Multi-line Comment:

Syntax:

`/* This is multi line comment */`

Comments Best Practice

1

Use comments to explain *why* the code exists.

2

Avoid obvious comments.

3

Keep comments updated with code changes.

Data Types

A **data type** in programming defines **what kind of data** a variable can hold and **how much memory** it occupies.

Why are Data Types Important?

- They tell the computer **how to interpret the bits** stored in memory.
- They help the compiler **check for errors** and **optimize memory use**.
- Different data types support **different operations** (e.g., math on numbers, or characters in text).

Built-in Data Types

C++ provides the following fundamental data types:

- int: Integer values (e.g., 10)
- float: Floating point numbers (e.g., 3.14)
- double: Double precision floating point
- char: Single character (e.g., 'A')
- bool: Boolean values (true or false)



Integer Data Types

Type	Size (Bytes)	Range
short	2	-32,768 to 32,767
unsigned short	2	0 to 65,535
int	4	-2,147,483,648 to 2,147,483,647
unsigned int	4	0 to 4,294,967,295
long	4 or 8	Platform-dependent
unsigned long	4 or 8	Platform-dependent
long long	8	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
unsigned long long	8	0 to 18,446,744,073,709,551,615

Floating Point Dat Types

Type	Size (Bytes)	Range (Approx.)
float	4	$\pm 1.2 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ (6 digits precision)
double	8	$\pm 2.3 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ (15 digits)
long double	8 or 16	Platform-dependent (higher precision)

Character and Boolean Types

Type	Size (Bytes)	Range
char	1	-128 to 127 (signed)
unsigned char	1	0 to 255
wchar_t	2 or 4	Wide character (Unicode)
char16_t	2	UTF-16 character
char32_t	4	UTF-32 character
bool	1	true (1) or false (0)

Variable

A **variable** is a **named storage** that holds data which can **change** during program execution.

You must **declare** the variable's **data type** before using it.

Variable names must follow identifier rules (e.g., no spaces, no starting with numbers)

Syntax

```
dataType variableName = value;
```

```
int age = 25;
```

```
float temperature = 36.6;
```

```
char grade = 'A';
```

You can **change** the value of a variable later:

```
age = 30; // age now holds 30
```

Constant

A **constant** is like a variable, but its value **cannot change** after initialization.

Declared using the `const` keyword.

Helps prevent accidental changes to important values.

Syntax

```
const dataType constantName = value;
```

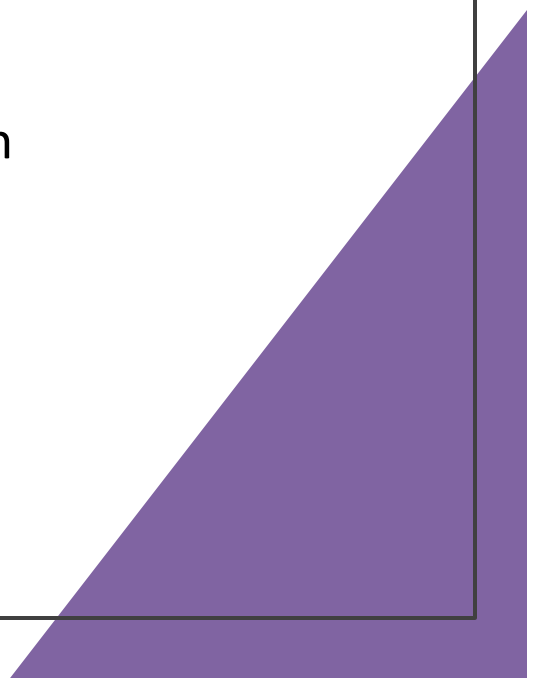
```
const double PI = 3.14159;
```

```
const int DAYS_IN_WEEK = 7;
```

Trying to change a constant will cause a **compile-time error**:

```
PI = 3.14; // Error! Cannot assign to a constant
```

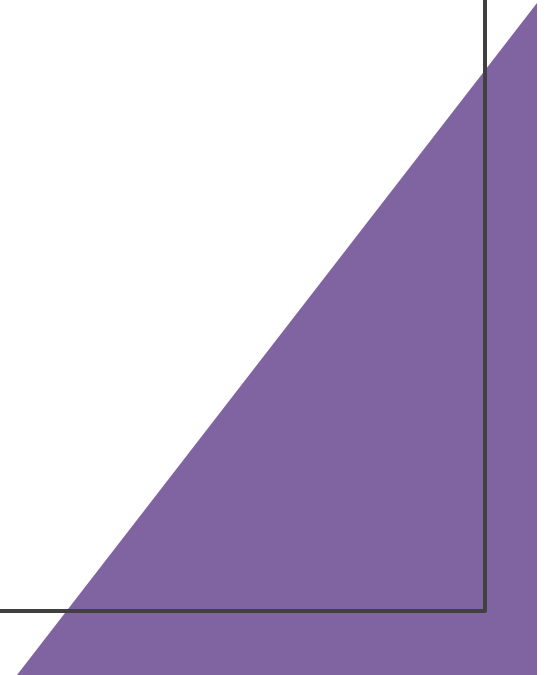
Constants in C++

- `const` keyword: `const int size = 10;`
 - `#define` macro: `#define SIZE 10`
 - Constants cannot be changed during execution
 - Use constants to improve code readability and maintenance
- 

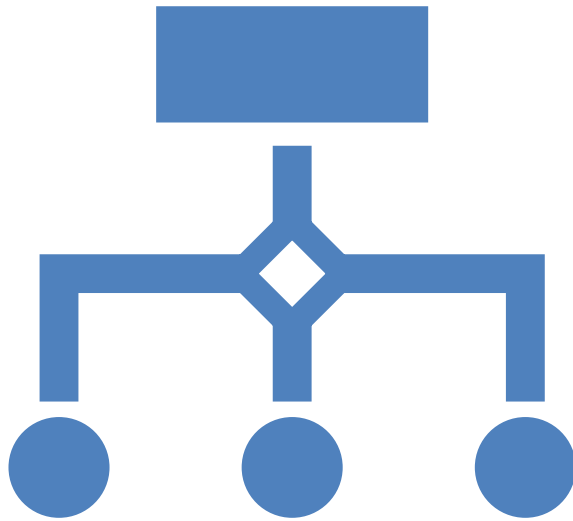
Example: Variables and Constants

```
const float PI = 3.14;  
int radius = 5;  
float area = PI * radius * radius;  
cout << "Area: " << area;
```

- PI is a constant
- Variables used to compute area



Operators in C++



Operators are symbols that tell the compiler to perform specific mathematical, logical, or relational operations.

Types of Operators

- Arithmetic
- Relational
- Logical
- Assignment
- Incremental

Arithmetic Operator

Used to perform basic math calculations.

Operator	Meaning	Example	Result
+	Addition	$5 + 3$	8
-	Subtraction	$5 - 3$	2
*	Multiplication	$5 * 3$	15
/	Division	$6 / 3$	2
%	Modulus (remainder)	$5 \% 3$	2

Relational Operator

- Used to compare two values, results in true or false.

OPERATOR	MEANING	EXAMPLE	RESULT
==	Equal to	5 == 3	false
!=	Not equal to	5 != 3	true
>	Greater than	5 > 3	true
<	Less than	5 < 3	false
>=	Greater or equal	5 >= 5	true
<=	Less or equal	3 <= 5	true

Logical Operators

- Used to combine multiple conditions.

Operator	Meaning	Example	Result
&&	Logical AND	(5 > 3) && (2 < 4)	true
	Logical OR	(5 > 3) (2 > 4)	true
!	Logical NOT	!(5 == 3)	true

Assignment Operators



Used to assign values to variables.

Operator	Meaning	Example	Result
=	Assign	x = 5	x = 5
+=	Add and assign	x += 3	x = x + 3
-=	Subtract and assign	x -= 2	x = x - 2
*=	Multiply and assign	x *= 4	x = x * 4
/=	Divide and assign	x /= 2	x = x / 2
%=	Modulus and assign	x %= 3	x = x % 3

Increment and Decrement Operators



Operator	Meaning	Example	Result
++	Increment by 1	x++ or ++x	x = x + 1
--	Decrement by 1	x-- or --x	x = x - 1


Escape Sequence

- Sometimes, you want your program to display special characters or format the text output in a certain way. Since you can't type these special commands directly in strings, C++ uses **escape sequences** to handle this.
- They are **special codes** inside strings that start with a backslash (\).
- They tell the program to perform an action or insert a special character.
- For example, moving to a new line, adding tabs, or printing quotes.

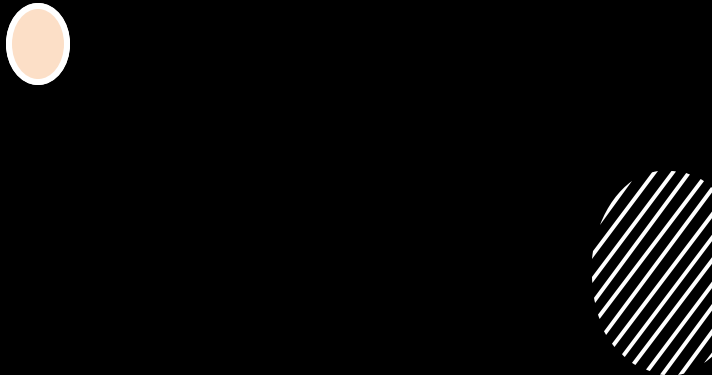


Common Escape Sequence

ESCAPE CODE	DESCRIPTION	EXAMPLE	OUTPUT
<code>\n</code>	Inserts a new line	<code>cout << "Hi\nThere";</code>	Hi There
<code>\t</code>	Inserts a tab space	<code>cout << "A\tB";</code>	A B
<code>\\</code>	Prints a backslash	<code>cout << "\\";</code>	\
<code>\"</code>	Prints a double quote	<code>cout << "\"Hello\"";</code>	"Hello"
<code>\'</code>	Prints a single quote	<code>cout << "It\'s OK";</code>	It's OK
<code>\a</code>	Produces a beep sound	<code>cout << "\a";</code>	(Beep sound if supported)
<code>\b</code>	Deletes the previous character	<code>cout << "ABC\bD";</code>	ABD
<code>\r</code>	Returns cursor to start of line	<code>cout << "12345\rABC";</code>	ABC45 (ABC overwrites start)



Why to use Escape Sequence



TO **FORMAT YOUR OUTPUT**
CLEARLY (NEW LINES, TABS).

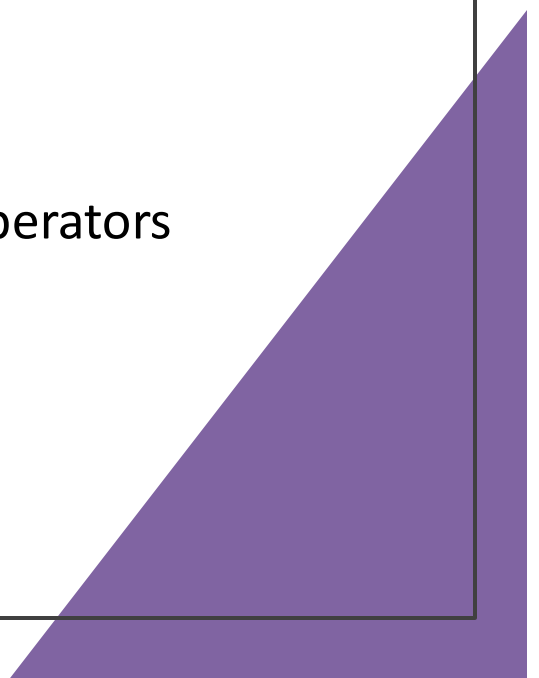


TO **INCLUDE SPECIAL**
CHARACTERS IN YOUR TEXT.



TO MAKE YOUR PROGRAM'S
OUTPUT **EASIER TO READ**
AND UNDERSTAND.

Input/Output in C++

- cin: Standard input (keyboard)
 - cout: Standard output (screen)
 - Requires iostream library
 - Used with insertion (<<) and extraction (>>) operators
- 

Insertion and Extraction Operators

<< : Outputs data to console (cout)

>> : Accepts input from user (cin)

Example:

```
int age;
```

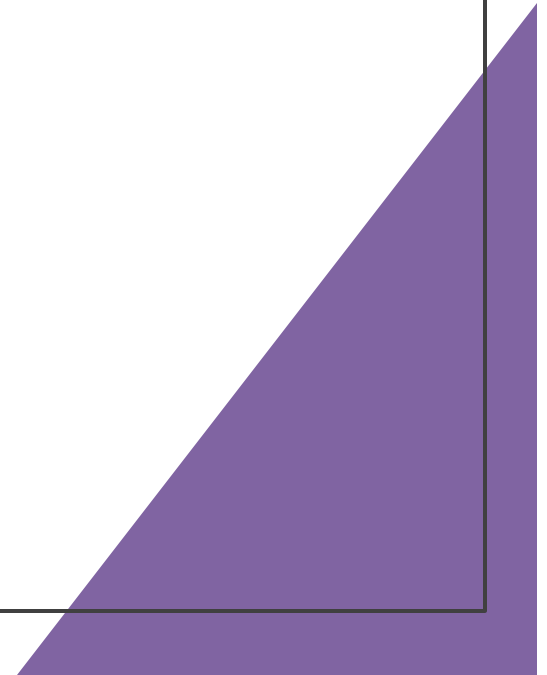
```
cin >> age;
```

```
cout << "Age is: " << age;
```

I/O Example Program

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

int main() {
    int num;
    cout << "Enter a number: ";
    cin >> num;
    cout << "You entered: " << num;
    return 0;
}
```



What is Type Conversion?

- Converting data from one type to another
- Implicit: Done automatically by compiler
- Explicit: Manual casting by programmer

Implicit Type Conversion

- Occurs automatically during expressions
- Lower to higher type: int to float
- Example: `int x = 10; float y = x; // x is promoted`

Explicit Type Conversion

Also called type casting

Syntax: (data_type)variable;

Example: float a = 10.5; int b =
(int)a; // b = 10


Type Conversion Example

```
int a = 5, b = 2;  
float result = (float)a / b;  
cout << "Result: " << result;
```

Casts a to float to get decimal
result



endl

- endl stands for **end line**.
 - It is used with **cout** to **insert a newline** character.
 - It also **flushes the output buffer**, meaning it forces the output to be printed immediately.
- 

Example

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

int main() {
    cout << "Hello World" << endl; // Moves to next line after printing
    cout << "Welcome to C++" << endl;
    return 0;
}
```


Structure Definition

Collection of related variables under one name

Syntax:

```
struct Student {  
    int id;  
    char name[50];  
    float marks;  
};
```

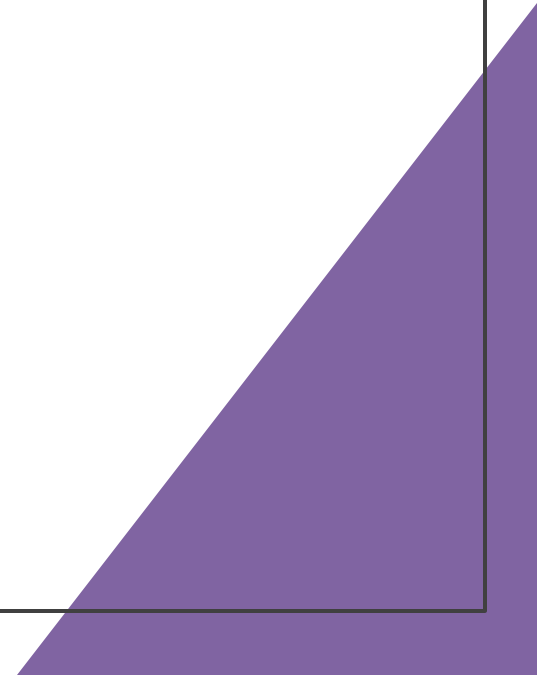
User-defined and Derived Types

User-defined types:

- struct: Combines variables under one type
- enum: Enumerated constants

Derived types:

- Arrays, pointers, references



Accessing Structure Members

Use dot operator (.) with
variable

Example:

Student s1;

s1.id = 1;

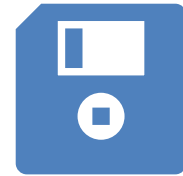
cout << s1.id;

Structure Example Program

```
struct Book {  
    int id;  
    char title[50];  
};  
  
int main() {  
    Book b1 = {1, "C++ Basics"};  
    cout << "ID: " << b1.id << ", Title: " << b1.title;  
    return 0;  
}
```

What is Dynamic Memory Allocation?

- Memory is allocated at runtime (heap)
- Useful when size is unknown at compile-time
- new and delete operators are used



new Operator

- Allocates memory and returns pointer

Example:

```
int* p = new int;
```

```
*p = 10;
```

```
cout << *p;
```

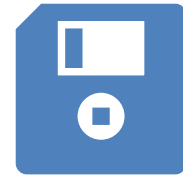


delete Operator

- Frees dynamically allocated memory
- Must be used to avoid memory leaks

Example:

- `delete p; // p was allocated with new`



Dynamic Arrays

Allocate arrays dynamically

```
int* arr = new int[5];
```

```
for (int i = 0; i < 5; i++)
```

```
    arr[i] = i * 2;
```

```
delete[] arr; // Use delete[] for arrays
```



Dynamic Memory Example

```
int* num = new int(25);  
cout << "Value: " << *num;  
delete num;
```

- Shows allocation, usage, and deallocation

Summary of Topics

- C++ program starts with main()
- Variables and constants store data
- cin/cout used for I/O
- Type conversion changes data type
- Structures group related data
- Dynamic memory helps manage heap memory

