C++ Programming Language

Author:
Youssef Lamkhantar

1.Introduction

- Developed by **Bjarne Stroustrup in 1983** as an extension of C.
- C++ supports both **procedural** and **object-oriented** programming, making it **versatile**.
- Widely used in systems programming, game development, embedded systems, and more.

2.Key Features of C++

- **Object-Oriented Programming (OOP)**: Encapsulation, inheritance, polymorphism, and abstraction.
- **Portability**: Write once, compile anywhere.
- **Low-Level Manipulation**: Like C, C++ allows direct memory access via pointers.
- **Extensibility**: Allows the definition of complex data types and overloading operators.

3.Basic Syntax

- **Variables**: Declaring and initializing variables.
- **Data Types**: Primitive types (`int`, `float`, `char`, `double`, etc.) and derived types (arrays, pointers, references).
- **Operators**: Arithmetic, relational, logical, bitwise, and assignment operators.
- **Comments**: Single-line (`//`) and multi-line (`/* */`).

4.Control Structures:

- **Conditional Statements**: `if`, `else`, `else if`, `switch`.
- **Loops**:
  - `for`: Loop that repeats a block of code a certain number of times.
  - `while`: Repeats a block as long as a condition is true.
  - `do-while`: Similar to `while` but ensures the block is executed at least once.
- **Jump Statements**: `break`, `continue`, `return`, and `goto`.

5.Functions:

- **Defining Functions**: Syntax, return types, and parameters.
- **Function Overloading**: Defining multiple functions with the same name but different parameter types or counts.
- **Inline Functions**: Functions defined inside the class body.
- **Default Arguments**: Functions with parameters that have default values.

5.Functions:

Input/Output Functions:

These functions handle standard input and output operations in C++.

- `cin`: Reads input from the user.
- `cout`: Outputs data to the console.
- `cerr`: Outputs error messages to the console.
- `clog`: Outputs log messages to the console.
- `getline()`: Reads a whole line of input.

5.Functions:

String Handling Functions (from `<string>` library):

C++ provides a set of functions to manipulate strings using the `std::string` class.

- **`length()` or `size()`**: Returns the length of a string.
- **`substr()`**: Extracts a substring from a string.
- **`find()`**: Searches for a substring or character.
- **`append()`**: Adds more characters to the end of a string.
- **`insert()`**: Inserts characters at a specific position.
- **`erase()`**: Deletes a part of the string.
- **`replace()`**: Replaces a substring with another substring.
- **`c_str()`**: Converts a string to a C-style string (character array).
- **`compare()`**: Compares two strings.

5.Functions:

Mathematical Functions (from `<cmath>` library):

These functions perform mathematical calculations and operations.

- `abs()`: Returns the absolute value of an integer.
- `fabs()`: Returns the absolute value of a floating-point number.
- `sqrt()`: Returns the square root of a number.
- `pow()`: Raises a number to a power.
- `ceil()`: Rounds a number up to the nearest integer.
- `floor()`: Rounds a number down to the nearest integer.
- `sin()`, `cos()`, `tan()`: Trigonometric functions.
- `log()`: Returns the natural logarithm of a number.
- `exp()`: Returns the exponential value of a number.

5.Functions:

Memory Management Functions:

C++ provides functions for dynamic memory management.

- `new`: Allocates memory dynamically.
- `delete`: Deallocates memory dynamically.
- `malloc()`: Allocates dynamic memory (from C).
- `free()`: Deallocates dynamic memory (from C).
- `realloc()`: Reallocates dynamic memory (from C).

5.Functions:

File Handling Functions (from `<fstream>` library):

Functions used to read from and write to files.

- `ifstream`: Input file stream (for reading files).
- `ofstream`: Output file stream (for writing to files).
- `fstream`: For both input and output to files.
- `open()`: Opens a file.
- `close()`: Closes a file.
- `read()`: Reads from a file.
- `write()`: Writes to a file.
- `eof()`: Checks for the end of the file.

5.Functions:

Time Functions (from `<ctime>` library):

Functions to manipulate and retrieve time and date information.

- `time()`: Returns the current calendar time.
- `clock()`: Returns the processor time consumed by the program.
- `difftime()`: Computes the difference between two time values.
- `localtime()`: Converts time to a local time structure.
- `strftime()`: Formats the date and time.

5.Functions:

Character Handling Functions (from `<cctype>` library):

Functions to work with individual characters.

- `isalpha()`: Checks if a character is an alphabet letter.
- `isdigit()`: Checks if a character is a digit.
- `isalnum()`: Checks if a character is alphanumeric.
- `isspace()`: Checks if a character is a white-space character.
- `toupper()`: Converts a character to uppercase.
- `tolower()`: Converts a character to lowercase.

5.Functions:

Utility Functions (from `<cstdlib>` library):

Various utility functions, including those for program control and conversion.

- `exit()`: Terminates the program.
- `system()`: Executes a system command.
- `atoi()`: Converts a string to an integer.
- `atof()`: Converts a string to a floating-point number.
- `itoa()`: Converts an integer to a string (non-standard, but often used).
- `rand()`: Generates a random number.
- `srand()`: Seeds the random number generator.

5.Functions:

Standard Template Library (STL) Functions:

These functions are part of the Standard Template Library and are used for operations on containers.

- `push_back()`: Adds an element to the end of a container (like `std::vector` or `std::deque`).
- `pop_back()`: Removes the last element from a container.
- `insert()`: Inserts elements at a specific position.
- `erase()`: Removes elements from a container.
- `size()`: Returns the number of elements in a container.
- `sort()`: Sorts elements in a container.
- `find()`: Searches for an element in a container.
- `begin()`: Returns an iterator to the beginning of a container.
- `end()`: Returns an iterator to the end of a container.

5.Functions:

Exception Handling Functions:

Functions used for handling runtime errors.

- `throw`: Throws an exception.
- `try`: Used to define a block of code where exceptions may occur.
- `catch`: Catches and handles the exceptions thrown by the `try` block.

6.Object-Oriented Programming (OOP) Concepts:

- **Classes and Objects**: Fundamental building blocks in C++.
  - **Class Declaration**: Syntax for defining classes and creating objects.
  - **Member Functions**: Functions defined inside a class.
- **Encapsulation**: Combining data and functions that operate on data in a single unit (class).
- **Inheritance**: Creating new classes from existing ones.
  - **Types of Inheritance**: Single, multiple, multilevel, hierarchical.

6.Object-Oriented Programming (OOP) Concepts:

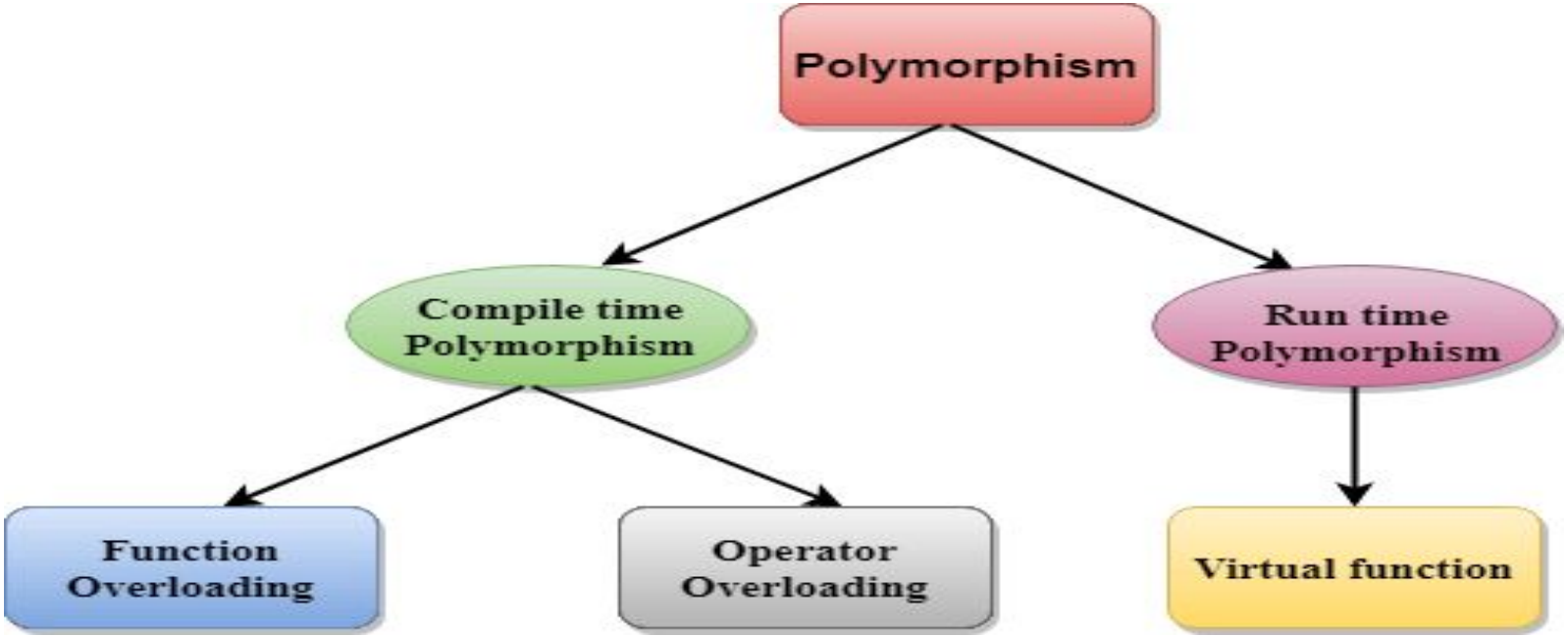
Polymorphism: Ability to redefine methods in derived classes.

- **Compile-time Polymorphism**: Function overloading and operator overloading.
- **Run-time Polymorphism**: Virtual functions and overriding.

Abstraction: Hiding the internal details and showing only the functionality.
Access Specifiers: `public`, `private`, `protected`.

Constructors and Destructors: Special member functions for object creation and cleanup.

Example of Polymorphism



7.Pointers and References

- **Pointers**: Variables that store the memory address of other variables.
- **Pointer Arithmetic**: Operations on memory addresses.
- **References**: Aliases for variables, created using the `&` symbol.
- **Smart Pointers**: Part of the C++ Standard Library for automatic memory management (`std::unique_ptr`, `std::shared_ptr`).

8.Dynamic Memory Management

- **new** and **delete**: Allocating and deallocating memory dynamically.
- **Pointers to objects**: Creating objects dynamically and accessing members using pointers.

9.Operator Overloading

- Overloading built-in operators such as `+`, `-`, `*`, `/`, `==`, `[ ]`, etc.
- Syntax for defining operator overloading functions inside classes.

10.Exception Handling

- **try-catch-finally**: Handling runtime errors using exceptions.
- **throw**: Raising exceptions when an error occurs.
- **Custom Exception Classes**: Defining user-defined exceptions.

11.Templates

- **Function Templates**: Writing generic functions that work with any data type.
- **Class Templates**: Defining generic classes.
- **Template Specialization**: Customizing the behavior of a template for a specific data type.

12.Standard Template Library (STL)

- **Containers**: Pre-defined data structures like `vector`, `list`, `deque`, `set`, `map`, `stack`, `queue`.
- **Iterators**: Used to traverse containers.
- **Algorithms**: Functions like `sort()`, `find()`, `count()`, etc.
- **Function Objects (Functors)**: Objects that can be called as functions.

13.File Handling

- Reading from and writing to files using streams.
- **ifstream**: Input stream for reading from files.
- **ofstream**: Output stream for writing to files.
- **fstream**: For both reading and writing.
- File operations: Opening, closing, reading, writing, and appending.

14.Namespaces

- **std namespace**: Avoid name conflicts in large programs.
- **Creating custom namespaces** to group code logically.

15.Preprocessor Directives:

- **#include**: Including header files.
- **#define**: Defining macros.
- **Conditional Compilation**: Using `#if`, `#endif`, `#ifdef`.

16.Lambda Expressions

- Anonymous functions introduced in C++11.
- Syntax: `[]() { //code }`

17.Multithreading (C++11 onwards)

- Using the `thread` class for concurrent programming.
- **Mutexes**: For synchronizing shared resources between threads.
- **Thread Creation**: Launching multiple threads to execute code concurrently.

18.New Features in Modern C++ (C++11, C++14, C++17, C++20)

- **C++11**: `auto`, `nullptr`, range-based for loops, lambdas, `std::array`, smart pointers.
- **C++14/17**: Enhanced templates, `std::variant`, `std::optional`, structured bindings, fold expressions.
- **C++20**: Concepts, ranges, coroutines, modules, enhanced constexpr.

19.Debugging and Error Handling

- Debugging tools like GDB, Visual Studio Debugger.
- Common mistakes: Segmentation faults, memory leaks, dereferencing null pointers.

20.Applications of C++:

- **Game Development**: Used in game engines like Unreal Engine.
- **Operating Systems**: Parts of Windows and macOS use C++.
- **Browser Development**: Google Chrome’s rendering engine.
- **Embedded Systems**: C++ in control systems, automotive software.
- **Financial Systems**: High-performance software for stock exchanges.

Conclusion

- C++ is a powerful, versatile language used for a wide range of applications from system-level programming to game development and high-performance applications.
- By understanding the key features, OOP concepts, templates, and modern enhancements, developers can harness the full potential of C++.