C++ Programming Reference Guide

Introduction to C++

1. Key Differences Between Procedural Programming and Object-Oriented Programming (OOP)

Procedural Programming (POP):

- Program is organized as a collection of functions
- Data and functions are separate entities
- Top-down approach
- Global data is accessible by all functions
- Code reusability is limited
- Problem is broken down into smaller sub-problems

Object-Oriented Programming (OOP):

- Program is organized as a collection of objects
- Data and functions are encapsulated within objects
- Bottom-up approach
- Data hiding through encapsulation
- High code reusability through inheritance
- Problem is modeled as real-world entities

2. Main Advantages of OOP over POP

- **Modularity**: Code is organized into separate, independent modules (classes)
- **Reusability**: Code can be reused through inheritance and polymorphism
- **Security**: Data hiding prevents unauthorized access to internal data
- **Maintainability**: Easier to modify and maintain code
- **Flexibility**: Easy to add new features without affecting existing code
- **Problem Solving**: Natural way to model real-world problems
- **Debugging**: Easier to locate and fix errors in modular code

3. Steps to Set Up C++ Development Environment

1. **Choose a Compiler**: Install GCC, Clang, or Microsoft Visual C++

2. **Install IDE/Text Editor**: Code::Blocks, Dev-C++, Visual Studio, or VS Code
3. **Configure Compiler Path**: Set environment variables for compiler access
4. **Create Project Structure**: Organize source files, headers, and libraries
5. **Set Build Configuration**: Configure debug/release settings
6. **Test Installation**: Compile and run a simple "Hello World" program

4. Main Input/Output Operations in C++

Standard I/O Streams:

- `cin` - Standard input stream
- `cout` - Standard output stream
- `cerr` - Standard error stream
- `clog` - Standard logging stream

Basic Syntax:

```
cout << "Output text" << variable;
cin >> variable;
```

File I/O:

```
#include <fstream>
ifstream inputFile("filename.txt");
ofstream outputFile("filename.txt");
```

Variables, Data Types, and Operators

1. Different Data Types in C++

Fundamental Data Types:

Integer Types:

- `int` - Integer (4 bytes)
- `short` - Short integer (2 bytes)
- `long` - Long integer (8 bytes)
- `long long` - Extended long integer (8 bytes)

Floating-Point Types:

- `float` - Single precision (4 bytes)
- `double` - Double precision (8 bytes)
- `long double` - Extended precision (12-16 bytes)

Character Types:

- `char` - Character (1 byte)
- `wchar_t` - Wide character (2-4 bytes)

Boolean Type:

- `bool` - Boolean value (true/false)

Derived Data Types:

- Arrays, Pointers, References, Functions

User-Defined Types:

- Classes, Structures, Unions, Enumerations

2. Implicit vs Explicit Type Conversion

Implicit Type Conversion (Automatic):

- Performed automatically by compiler
- Occurs when compatible types are mixed
- Also called type promotion
- Example: `int` to `float`, `char` to `int`

Explicit Type Conversion (Manual):

- Performed manually by programmer
- Uses casting operators
- More control over conversion process
- Syntax: `(type)variable` or `type(variable)`

3. Types of Operators in C++

Arithmetic Operators:

- `+` (Addition), `-` (Subtraction), `*` (Multiplication)
- `/` (Division), `%` (Modulus)

Relational Operators:

- `==` (Equal), `!=` (Not equal)
- `<` (Less than), `>` (Greater than)
- `<=` (Less than or equal), `>=` (Greater than or equal)

Logical Operators:

- `&&` (Logical AND), `||` (Logical OR), `!` (Logical NOT)

Assignment Operators:

- `=` (Simple assignment)
- `+=`, `-=`, `*=`, `/=`, `%=` (Compound assignment)

Increment/Decrement:

- `++` (Increment), `--` (Decrement)
- Pre-increment: `++var`, Post-increment: `var++`

Bitwise Operators:

- `&` (AND), `|` (OR), `^` (XOR), `~` (NOT)
- `<<` (Left shift), `>>` (Right shift)

4. Constants and Literals

Constants:

- Variables whose values cannot be changed after initialization
- Declared using `const` keyword
- Must be initialized at declaration
- Syntax: `const datatype variable_name = value;`

Literals:

- Fixed values that appear directly in program
- **Integer Literals**: 123, 0xFF (hexadecimal), 077 (octal)
- **Floating Literals**: 3.14, 2.5e10

- **Character Literals**: 'A', '\n', '\t'
- **String Literals**: "Hello World"
- **Boolean Literals**: true, false

Control Flow Statements

1. Conditional Statements

if-else Statement:

```
if (condition) {
    // statements
} else if (condition) {
    // statements
} else {
    // statements
}
```

switch Statement:

```
switch (expression) {
    case value1:
        // statements
        break;
    case value2:
        // statements
        break;
    default:
        // statements
}
```

2. Loop Statements

for Loop:

- Used when number of iterations is known
- Syntax: `for (initialization; condition; update) { statements }`
- Best for counting-based iterations

while Loop:

- Used when number of iterations is unknown
- Tests condition before executing loop body
- Syntax: `while (condition) { statements }`

do-while Loop:

- Executes loop body at least once
- Tests condition after executing loop body
- Syntax: `do { statements } while (condition);`

3. break and continue Statements

break Statement:

- Terminates the loop immediately
- Control jumps to statement after loop
- Also used in switch statements

continue Statement:

- Skips current iteration
- Control jumps to beginning of loop for next iteration
- Only affects innermost loop

4. Nested Control Structures

Nested structures involve placing one control structure inside another:

- Nested if statements for complex decision making
- Nested loops for multi-dimensional data processing
- Mixed nesting of loops and conditional statements
- Inner structure executes completely for each iteration of outer structure

---

Functions and Scope

1. Functions in C++

Function Declaration (Prototype):

- Tells compiler about function name, return type, and parameters

- Syntax: `return_type function_name(parameter_list);`

Function Definition:

- Contains actual implementation of function
- Syntax: `return_type function_name(parameter_list) { function_body }`

Function Calling:

- Invoking function by its name with required arguments
- Syntax: `function_name(arguments);`

2. Variable Scope

Local Scope:

- Variables declared inside a function or block
- Accessible only within that function/block
- Created when block is entered, destroyed when exited
- Also called block scope

Global Scope:

- Variables declared outside all functions
- Accessible throughout the program
- Exist for entire program duration
- Can be accessed by any function

Scope Resolution:

- Local variables hide global variables with same name
- Use scope resolution operator `::` to access global variables

3. Recursion

Definition:

- Function calling itself directly or indirectly
- Must have base case to prevent infinite recursion
- Each recursive call creates new set of local variables

Components:

- **Base Case**: Condition to stop recursion
- **Recursive Case**: Function calls itself with modified parameters
- **Stack Management**: Each call stored on call stack

4. Function Prototypes

Purpose:

- Inform compiler about function before its actual definition
- Enable forward declarations
- Allow functions to be defined after main function
- Required for separate compilation

Benefits:

- Early error detection
- Better code organization
- Enables mutual recursion
- Required for function overloading

Arrays and Strings

1. Arrays in C++

Single-Dimensional Arrays:

- Collection of elements of same data type
- Elements stored in contiguous memory locations
- Accessed using index (0-based indexing)
- Syntax: `datatype array_name[size];`

Multi-Dimensional Arrays:

- Arrays of arrays
- Most common is 2D array (matrix)
- Syntax: `datatype array_name[rows][columns];`
- Can have more than 2 dimensions

2. String Handling

C-Style Strings:

- Array of characters terminated by null character '\0'
- Declared as character arrays
- Functions from <cstring> library

C++ String Class:

- Part of Standard Template Library (STL)
- Include <string> header
- Provides built-in methods for string manipulation
- Dynamic size management

3. Array Initialization

1D Array Initialization:

```
int arr1[5] = {1, 2, 3, 4, 5};      // Complete initialization
int arr2[5] = {1, 2};               // Partial initialization
int arr3[] = {1, 2, 3};             // Size determined automatically
```

2D Array Initialization:

```
int matrix[3][3] = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}};
int matrix2[2][3] = {1, 2, 3, 4, 5, 6};
```

4. String Operations and Functions

C-Style String Functions:

- `strlen()` - Find string length
- `strcpy()` - Copy string
- `strcat()` - Concatenate strings
- `strcmp()` - Compare strings

C++ String Class Methods:

- `length()` or `size()` - Get string length
- `substr()` - Extract substring
- `find()` - Find substring position

- `replace()` - Replace part of string
- `append()` - Add to end of string

Introduction to Object-Oriented Programming

1. Key Concepts of OOP

Encapsulation:

- Bundling data and methods within a single unit (class)
- Data hiding through access specifiers
- Provides interface to interact with object

Inheritance:

- Creating new classes based on existing classes
- Promotes code reusability
- Establishes "is-a" relationship

Polymorphism:

- Same interface for different underlying data types
- Function overloading and operator overloading
- Virtual functions for runtime polymorphism

Abstraction:

- Hiding complex implementation details
- Showing only essential features
- Achieved through abstract classes and interfaces

2. Classes and Objects

Class:

- Blueprint or template for creating objects
- Defines attributes (data members) and behaviors (member functions)
- Syntax: `class ClassName { access_specifier: members; };`

Object:

- Instance of a class
- Actual entity that occupies memory
- Created using class constructor
- Syntax: `ClassName objectName;`

Access Specifiers:

- `private`: Accessible only within the class
- `public`: Accessible from anywhere
- `protected`: Accessible within class and derived classes

3. Inheritance

Definition:

- Mechanism to create new class from existing class
- Parent class (base class) properties inherited by child class (derived class)
- Promotes code reusability and hierarchical classification

Types of Inheritance:

- **Single Inheritance**: One base class, one derived class
- **Multiple Inheritance**: One derived class, multiple base classes
- **Multilevel Inheritance**: Chain of inheritance
- **Hierarchical Inheritance**: Multiple derived classes from single base
- **Hybrid Inheritance**: Combination of multiple types

Syntax:

```
class DerivedClass : access_specifier BaseClass {
    // class body
};
```

4. Encapsulation

Definition:

- Wrapping data and functions together in a single unit
- Restricting direct access to object's components
- Achieved through access specifiers

Implementation:

- Private data members
- Public member functions to access private data
- Getter and setter methods for controlled access

Benefits:

- Data security and integrity
- Code maintainability
- Flexibility in changing internal implementation
- Better control over how data is accessed and modified

Data Hiding:

- Core principle of encapsulation
- Internal object details hidden from outside world
- Only necessary information exposed through public interface

---

Summary

This reference guide covers the fundamental concepts of C++ programming, from basic syntax and data types to object-oriented programming principles. Each topic builds upon previous concepts, creating a solid foundation for C++ development.

Key Takeaways:

- C++ supports both procedural and object-oriented programming paradigms
- Strong type system with various data types and operators
- Flexible control flow mechanisms for program logic
- Function-based code organization with scope management
- Array and string handling capabilities
- Object-oriented features for modern software design

This guide provides the essential knowledge needed to begin C++ programming and understand its core concepts.