Ebook on C++ Programming

Compiled By - Mr. Kamal D. Agrahari

Chapter 1: Introduction to C++

What is C++?

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language. It is a statically typed, compiled language that supports object-oriented programming. C++ is one of the most popular programming languages in the world, and is used in a wide variety of applications, including operating systems, embedded systems, and game development.

History of C++

C++ was first released in 1985. It was originally designed as a way to add object-oriented features to C, but it has since evolved into a powerful and versatile language. C++ has been updated several times over the years, with the most recent major update being C++20, which was released in 2020.

C++ programming environment

To write C++ programs, you will need a C++ compiler and a text editor. There are many different C++ compilers available, such as GCC and Clang. You can also use an integrated development environment (IDE) such as Visual Studio or Xcode.

Basic C++ syntax

C++ syntax is similar to that of C. However, there are some important differences. For example, C++ uses classes and objects, while C does not.
Here is a simple example of a C++ program:
C++

```
#include <iostream>

int main() {
  std::cout << "Hello, world!" << std::endl;
  return 0;
}
```

This program prints the message "Hello, world!" to the console.

Summary

This chapter has introduced the C++ programming language. We have learned what C++ is, its history, its programming environment, and its basic syntax. In the next chapter, we will learn about data types in C++.

Chapter 2 : Data types in C++:

Data types

Data types are used to store data in a C++ program. C++ has a variety of data types, including integers, floating-point numbers, characters, strings, and Booleans.

Integers

Integers are whole numbers, such as 1, 2, 3, and -4. There are several different types of integers in C++, including:

- int: The default integer type. It can store values from -2147483648 to 2147483647.
- short: A smaller integer type. It can store values from -32768 to 32767.
- long: A larger integer type. It can store values from -2147483648 to 2147483647.
- long long: The largest integer type. It can store values from -9223372036854775808 to 9223372036854775807.

Floating-point numbers

Floating-point numbers are numbers that have a decimal point, such as 3.14159 and -2.71828. There are two types of floating-point numbers in C++, float and double:

- float: The default floating-point type. It can store values with about 7 decimal digits of precision.
- double: A larger floating-point type. It can store values with about 15 decimal digits of precision.

Characters

Characters are single letters, digits, and special symbols. They are represented by the char data type.

Strings

Strings are sequences of characters. They are represented by the string data type.

Booleans

Booleans are values that can be either true or false. They are represented by the bool data type.

Declaring variables

Variables are used to store data in a C++ program. They must be declared before they can be used. The declaration specifies the data type of the variable and the name of the variable.
Here is an example of a variable declaration:

```
C++
int age;
```

This declaration declares a variable named age of type int.

Assigning values to variables

Once a variable is declared, you can assign a value to it. The assignment operator is used to assign values to variables.
Here is an example of an assignment statement:

```
C++
age = 18;
```

This statement assigns the value 18 to the variable age.

Using variables

Variables can be used in expressions and statements. For example, the following statement prints the value of the variable age:

```
C++
std::cout << "Your age is " << age << std::endl;
```

Summary

This chapter has introduced data types in C++. We have learned about the different types of data types, how to declare variables, and how to assign values to variables. In the next chapter, we will learn about operators in C++.

Chapter 3: Operators

Operators are symbols that are used to perform operations on data. C++ has a variety of operators, including arithmetic operators, logical operators, relational operators, assignment operators, and bitwise operators.

Arithmetic operators

Arithmetic operators are used to perform mathematical operations on data. The most common arithmetic operators are:

- +: Addition
- -: Subtraction
- *: Multiplication

- /: Division
- %: Modulus

Logical operators

Logical operators are used to perform logical operations on data. The most common logical operators are:
- &&: Logical AND
- ||: Logical OR
- !: Logical NOT

Relational operators

Relational operators are used to compare data. The most common relational operators are:
- ==: Equal to
- !=: Not equal to
- <: Less than
- <=: Less than or equal to
- >: Greater than
- >=: Greater than or equal to

Assignment operators

Assignment operators are used to assign values to variables. The most common assignment operators are:
- =: Assignment
- +=: Addition assignment
- -=: Subtraction assignment
- *=: Multiplication assignment
- /=: Division assignment
- %=: Modulus assignment

Bitwise operators

Bitwise operators are used to perform bitwise operations on data. Bitwise operations are performed on the bits of the data, rather than the values of the data. The most common bitwise operators are:
- &: Bitwise AND
- |: Bitwise OR
- ^: Bitwise XOR
- ~: Bitwise NOT
- <<: Left shift
- >>: Right shift

Operator precedence

Operator precedence determines the order in which operators are evaluated in an expression. The following table shows the operator precedence in C++:

Operator	Precedence
() [] -> .	10
++ -- ~ ! * / % + -	9
<< >>	8
< <= > >= == !=	7
&	6
`^	`
&&	4
`	`
=	2
()	1

Summary

This chapter has introduced operators in C++. We have learned about the different types of operators, their precedence, and how to use them in expressions. In the next chapter, we will learn about expressions in C++.

Chapter 4: Expressions

An expression is a combination of operators and operands that evaluates to a single value. Expressions can be used in statements, such as assignment statements and conditional statements.

Operands

Operands are the values that are operated on by operators. Operands can be variables, literals, or expressions.

Operators

Operators are symbols that are used to perform operations on operands. C++ has a variety of operators, including arithmetic operators, logical operators, relational operators, assignment operators, and bitwise operators.

Expression types

Expressions can be classified into the following types:

- Arithmetic expressions: Expressions that use arithmetic operators.
- Logical expressions: Expressions that use logical operators.
- Relational expressions: Expressions that use relational operators.
- Assignment expressions: Expressions that use assignment operators.
- Bitwise expressions: Expressions that use bitwise operators.

Expression evaluation

Expressions are evaluated according to their operator precedence. The operator precedence table shows the order in which operators are evaluated.

Parentheses

Parentheses can be used to group expressions together. Parentheses have the highest operator precedence, so expressions inside parentheses are evaluated first.

Summary

This chapter has introduced expressions in C++. We have learned about the different types of expressions, how to evaluate expressions, and how to use parentheses.
I hope this helps!
Here are some examples of expressions in C++:

- 1 + 2: An arithmetic expression that evaluates to the sum of 1 and 2.
- true && false: A logical expression that evaluates to the logical AND of true and false.
- x < y: A relational expression that evaluates to true if x is less than y.
- a = b: An assignment expression that assigns the value of b to the variable a.
- ~x: A bitwise expression that performs the bitwise NOT of x.

Chapter 5: Statements

Chapter 5: Statements

Statements are the basic building blocks of a C++ program. They are instructions that the computer executes.

Types of statements

There are many different types of statements in C++, including:

- Assignment statements: Statements that assign values to variables.
- Conditional statements: Statements that execute different code depending on the value of an expression.
- Loop statements: Statements that execute a block of code repeatedly.
- Function calls: Statements that call a function.
- Return statements: Statements that return from a function.
- Break statements: Statements that break out of a loop.
- Continue statements: Statements that continue to the next iteration of a loop.

Statement execution

Statements are executed in the order in which they appear in the program. However, there are some statements, such as conditional statements and loop statements, that can cause the execution of statements to be skipped or repeated.

Summary

This chapter has introduced statements in C++. We have learned about the different types of statements, how they are executed, and how they can be used to control the flow of a program.
I hope this helps!
Here are some examples of statements in C++:

- a = 10: An assignment statement that assigns the value 10 to the variable a.
- if (x > 0) { // Do something if x is greater than 0. }: A conditional statement that executes a block of code if the value of x is greater than 0.
- while (x < 10) { // Do something while x is less than 10. }: A loop statement that executes a block of code repeatedly while the value of x is less than 10.
- printf("Hello, world!\n");: A function call that prints the message "Hello, world!" to the console.
- return 0;: A return statement that returns the value 0 from the function.
- break;: A break statement that breaks out of the current loop.
- continue;: A continue statement that continues to the next iteration of the current loop.

Chapter 6: Functions

A function is a block of code that is executed when it is called. Functions can be used to modularize code, make code reusable, and improve readability.

Function declaration

A function declaration specifies the name of the function, the return type of the function, and the parameters of the function.
The following is an example of a function declaration in C++:

```
C++
int factorial(int n);
```

This declaration specifies that the function is named factorial, it has a return type of int, and it takes one parameter, which is an int.

Function definition

A function definition is the code that is executed when the function is called. The function definition must match the function declaration in terms of the return type, the parameters, and the body of the function.
The following is an example of a function definition in C++:

```
C++
int factorial(int n) {
  if (n == 0) {
    return 1;
  } else {
    return n * factorial(n - 1);
  }
}
```

This definition defines the function factorial, which takes an int as a parameter and returns an int. The function first checks if the parameter is 0. If it is, the function returns 1. Otherwise, the function recursively calls itself, passing the value of the parameter minus 1.

Function call

A function call is used to execute a function. The function call specifies the name of the function and the arguments to the function.
The following is an example of a function call in C++:

```
C++
int result = factorial(5);
```

This call executes the function factorial, passing the value 5 as an argument. The result of the function call is stored in the variable result.

Summary

This chapter has introduced functions in C++. We have learned about the different parts of a function, how to declare and define functions, and how to call functions.
I hope this helps!
Here are some additional things to keep in mind about functions in C++:

- Functions can be recursive, which means that they can call themselves.
- Functions can be overloaded, which means that there can be multiple functions with the same name but different parameters.
- Functions can be declared as inline, which means that the compiler will copy the body of the function into the code where the function is called.

Chapter 7: Classes

A class is a user-defined data type that defines the data and behavior of objects. Classes are the basic building blocks of object-oriented programming.

Class declaration

A class declaration specifies the name of the class, the members of the class, and the methods of the class.
The following is an example of a class declaration in C++:
C++
```
class Car {
  private:
    int _wheels;
    string _color;

  public:
    void setWheels(int wheels);
    string getColor();
};
```

This declaration specifies that the class is named Car, it has two private members, _wheels and _color, and it has two public methods, setWheels() and getColor().

Class members

The members of a class are the data and behavior that are associated with the class. The members of a class can be private, protected, or public.

- Private members are only accessible to the members of the class.
- Protected members are accessible to the members of the class and its subclasses.
- Public members are accessible to everyone.

Class methods

The methods of a class are the functions that are associated with the class. Methods can be used to access and modify the members of the class.

Class objects

An object is an instance of a class. Objects have the same members and methods as the class that they are an instance of.

Creating objects

Objects are created using the new keyword. The following code creates an object of the Car class:

```
C++
Car car;
```

Accessing object members

Object members can be accessed using the dot notation. The following code accesses the _wheels member of the car object:

```
C++
int wheels = car._wheels;
```

Modifying object members

Object members can be modified using the dot notation. The following code modifies the _wheels member of the car object:

```
C++
car._wheels = 4;
```

Summary

This chapter has introduced classes in C++. We have learned about the different parts of a class, how to declare classes, and how to create and use objects.
I hope this helps!
Here are some additional things to keep in mind about classes in C++:

- Classes can be inherited, which means that a new class can inherit the members and methods of an existing class.
- Classes can be templates, which means that they can be used to create generic objects.

Chapter 8: Inheritance

Inheritance is a feature of object-oriented programming that allows one class to inherit the members and methods of another class. The class that inherits the members and methods of another class is called the derived class. The class that is being inherited from is called the base class.

Types of inheritance

There are two types of inheritance in C++:

- Single inheritance: A derived class can inherit from only one base class.
- Multiple inheritance: A derived class can inherit from multiple base classes.

Access specifiers

The members of a base class can be inherited as public, protected, or private.

- Public members are inherited as public in the derived class.
- Protected members are inherited as protected in the derived class.
- Private members are not inherited in the derived class.

Overriding methods

Methods in a derived class can override methods in the base class. When a method is overridden, the method in the derived class takes precedence over the method in the base class.

Constructor and destructor

The constructors and destructors of a base class are not inherited by the derived class. However, the derived class can call the constructors and destructors of the base class.

Summary

This chapter has introduced inheritance in C++. We have learned about the different types of inheritance, how to inherit members and methods, and how to override methods.
I hope this helps!
Here are some additional things to keep in mind about inheritance in C++:

- Inheritance can be used to reuse code and to create more complex and sophisticated classes.
- Inheritance can also be used to implement polymorphism, which is the ability of objects to behave differently depending on their type.

:

Chapter 9: Templates

Templates are a feature of C++ that allows you to create generic functions and classes. A template is a blueprint for a function or class that can be used with different types of data.

Function templates

A function template is a function that can be used with different types of data. The function template is defined using the template keyword.
The following is an example of a function template:
C++
```
template <typename T>
void swap(T& a, T& b) {
  T temp = a;
  a = b;
  b = temp;
}
```

This function template can be used to swap the values of two variables of any type.

Class templates

A class template is a class that can be used with different types of data. The class template is defined using the template keyword.
The following is an example of a class template:
C++
```
template <typename T>
class Stack {
  private:
    vector<T> data;

  public:
    void push(T value) {
      data.push_back(value);
    }

    T pop() {
      T value = data.back();
      data.pop_back();
      return value;
    }
};
```

This class template can be used to create a stack of any type of data.

Using templates

To use a template, you need to specify the type of data that you want to use with the template. The following code shows how to use the swap() function template to swap the values of two integers:

```
C++
int a = 1;
int b = 2;

swap(a, b);
```

The following code shows how to use the Stack class template to create a stack of integers:

```
C++
Stack<int> stack;

stack.push(1);
stack.push(2);

int value = stack.pop();
```

Summary

This chapter has introduced templates in C++. We have learned about the different types of templates, how to define templates, and how to use templates.
I hope this helps!
Here are some additional things to keep in mind about templates in C++:

- Templates can be used to create generic functions and classes that can be used with different types of data.
- Templates can be used to improve code readability and maintainability.
- Templates can be used to avoid code duplication.

chapter on Exceptions in C++:

Chapter 10: Exceptions

Exceptions are a way of handling errors in a program. When an error occurs, an exception is thrown. The exception is then handled by the program, which can either try to recover from the error or terminate the program.

Throwing an exception

An exception is thrown using the throw keyword. The throw keyword takes an object as its argument. This object is called the exception object.
The following code shows how to throw an exception:

```
C++
throw std::runtime_error("An error occurred");
```

This code throws a std::runtime_error exception with the message "An error occurred".

Catching an exception

An exception can be caught using the try and catch statements. The try statement specifies the code that might throw an exception. The catch statement specifies the code that will be executed if an exception is thrown.
The following code shows how to catch an exception:
C++
```
try {
  // Code that might throw an exception.
} catch (std::runtime_error& e) {
  // Code to handle the exception.
  std::cout << e.what() << std::endl;
}
```

This code first executes the code in the try statement. If an exception is thrown, the code in the catch statement is executed. The catch statement takes an object as its argument. This object is the exception object that was thrown.

Standard exceptions

C++ provides a number of standard exceptions. These exceptions are defined in the stdexcept header file.
Some of the most common standard exceptions are:

- std::runtime_error: This exception is thrown for runtime errors, such as division by zero.
- std::invalid_argument: This exception is thrown for invalid arguments, such as passing a non-integer to a function that expects an integer.
- std::out_of_range: This exception is thrown for out-of-range errors, such as accessing an element of an array that is beyond the bounds of the array.

Custom exceptions

You can also create your own custom exceptions. Custom exceptions are created by defining a class that inherits from the std::exception class.
The following code shows how to create a custom exception:
C++
```
class MyException : public std::exception {
  public:
    MyException(const std::string& message) : message_(message) {}

    virtual const char* what() const noexcept { return message_.c_str(); }

  private:
    std::string message_;
};
```

This code defines a custom exception called MyException. The MyException class inherits from the std::exception class. The MyException class has a constructor that takes a message as its argument. The MyException class also has a what() method that returns the message associated with the exception.

Using exceptions

Exceptions can be used to improve the readability and maintainability of your code.
Exceptions can also be used to make your code more robust.
Here are some tips for using exceptions:

- Use exceptions to handle errors that are not recoverable.
- Do not use exceptions to handle errors that can be easily handled without exceptions.
- Document the exceptions that your code throws.
- Use a consistent naming convention for your exceptions.

chapter on the Standard Library in C++:

Chapter 11: Standard library

The C++ standard library is a collection of functions, classes, and objects that are provided by the C++ programming language. The standard library is used to perform common tasks, such as input and output, string manipulation, and mathematical operations.

Header files

The standard library is divided into header files. Each header file contains the declarations for the functions, classes, and objects that are defined in that header file.
Some of the most common header files in the standard library are:

- iostream: This header file contains the declarations for the input and output functions.
- string: This header file contains the declarations for the string class.
- vector: This header file contains the declarations for the vector class.
- algorithm: This header file contains the declarations for the algorithms that are used to sort, search, and manipulate data.
- math: This header file contains the declarations for the mathematical functions.

Using the standard library

To use the standard library, you need to include the header files that contain the declarations for the functions, classes, and objects that you want to use. You can then use the functions, classes, and objects in your code.
For example, to use the cout object to print a message to the console, you would first need to include the iostream header file. You would then use the cout object in your code like this:
C++

```
#include <iostream>

int main() {
  std::cout << "Hello, world!\n";
  return 0;
}
```

Standard library functions

The standard library provides a wide variety of functions. Some of the most common standard library functions are:

- cin: This function reads data from the standard input stream.
- cout: This function writes data to the standard output stream.
- string: This function creates and manipulates strings.
- vector: This function creates and manipulates vectors.
- sort: This function sorts a vector.
- search: This function searches a vector for a specific value.

Standard library classes

The standard library also provides a number of classes. Some of the most common standard library classes are:

- string: This class represents a string.
- vector: This class represents a vector.
- algorithm: This class provides a number of algorithms for sorting, searching, and manipulating data.
- math: This class provides a number of mathematical functions.

Using the standard library effectively

The standard library is a powerful tool that can be used to improve the readability, maintainability, and efficiency of your code. Here are some tips for using the standard library effectively:

- Learn the standard library functions and classes.
- Use the standard library functions and classes whenever possible.
- Document the use of the standard library in your code.
- Use a consistent naming convention for the standard library functions and classes.

chapter on Advanced topics in C++:

Chapter 12: Advanced topics

C++ is a large and complex language, and there are many advanced topics that can be learned. Some of the most important advanced topics include:

Template metaprogramming: This is a technique for generating code at compile time.

Threading: This is the ability to run multiple tasks simultaneously.

Concurrency: This is the ability to handle multiple tasks that can run at the same time.

Generic programming: This is the ability to write code that can be used with different types of data.

Data structures: This is the study of data structures and algorithms.

Algorithms: This is the study of algorithms for solving problems.

Object-oriented programming: This is a programming paradigm that focuses on objects.

Design patterns: This is a set of reusable solutions to common problems in software design.

Unit testing: This is a technique for testing software to ensure that it works correctly.

Pointers: Pointers are a way of storing the address of a memory location. Pointers can be used to access and manipulate data that is stored in memory.

Arrays: Arrays are a way of storing a collection of data of the same type. Arrays can be used to store data efficiently and to access data quickly.

Strings: Strings are a common data type, but they can be difficult to manipulate. There are a number of standard library functions that can be used to manipulate strings, such as strcat(), strcmp(), and strlen().

File I/O: File I/O is the ability to read and write data to files. File I/O can be used to store data permanently and to retrieve data that has been stored.

Multithreading: Multithreading is the ability to run multiple tasks simultaneously. Multithreading can be used to improve the performance of programs that can be divided into multiple tasks.

Debugging: Debugging is the process of finding and fixing errors in code. Debugging can be a challenging task, but it is essential for ensuring that programs work correctly.

These are just a few of the many advanced topics that can be learned in C++. Learning these topics can help you to become a more proficient C++ programmer.

Here are some additional tips for learning advanced topics in C++:

- Find a good book or online resource on the topic that you want to learn.
- Practice the concepts that you learn.
- Join an online forum or community where you can ask questions and get help from other C++ programmers.
- Attend conferences and workshops on C++.