C++ Prog Lang In A Nutshell Vol # 1

COMPILED BY BILAL AHMAD KHAN AKA MR. BILRED

CppNotesByBILRED

cout << "*KNOWLEDGE SHOULD BE SHARED\n*

ONLY WITH THE ONE\n

WHO KNOWS ITS TRUE WORTH\n

NOT EVERYONE DESERVES IT" << endl;

"Imagine Everybody Living With Peace
Just Imagine"

"We don't need more divisions; we need unity and love to make this world a BETTER PLACE"

BILRED

Preface

If you're reading these notes, should I assume that?

1. **Your exam is right around the corner,** and you're scrambling for help.
2. **You already know a little bit of C++**, enough to make sense of what's written here.
3. **Or maybe you're an extremely ambitious person!**

I can't say these notes will make you a pro overnight, but hey, they might just help you get a quick know-how when you need it most.

I used AI to assist in compiling these notes (shout out to tech advancements). And hey, Believe it or not, the idea sparked when someone asked me for C++ notes. That tinkled my **CppNotesByBILRED** sense, and I seized the opportunity to create this resource (maybe maybe maybe).

So, thank me later ;)

I have uploaded some knowy on my github account, visit when you're free, it might help you, and yeah, its free, open-source, I guess so!
www.github.com/bilalahmadkhankhattak/cppnotes

By The Way,

Gone are the days when acquiring knowledge required relentless effort and deep exploration. There was a time when people had to sift through libraries, seek mentors, or dedicate years of their lives to uncover the truths of the world. Today, knowledge is literally at our fingertips, just a tap away, available to anyone with a smartphone or internet connection.

The real challenge today is no longer finding knowledge but discerning what is meaningful, what is true, and what is worth holding on to in a sea of distractions. Knowledge, after all, is not about how much you can access but about how wisely you can use it to make a difference, for yourself and for the world around you…

C++ Prog Lang In A Nutshell VOL # 1

CppNotesByBILRED

Compiled By Bilal Ahmad Khan AKA Mr. BILRED

26th January 2025

Price: Prayers only /-

C++ Prog Lang In A Nutshell

Introduction to C++

C++ is a general-purpose programming language developed by Bjarne Stroustrup in 1979. It is an extension of the C language, adding object-oriented and generic programming features.

Why Learn C++?

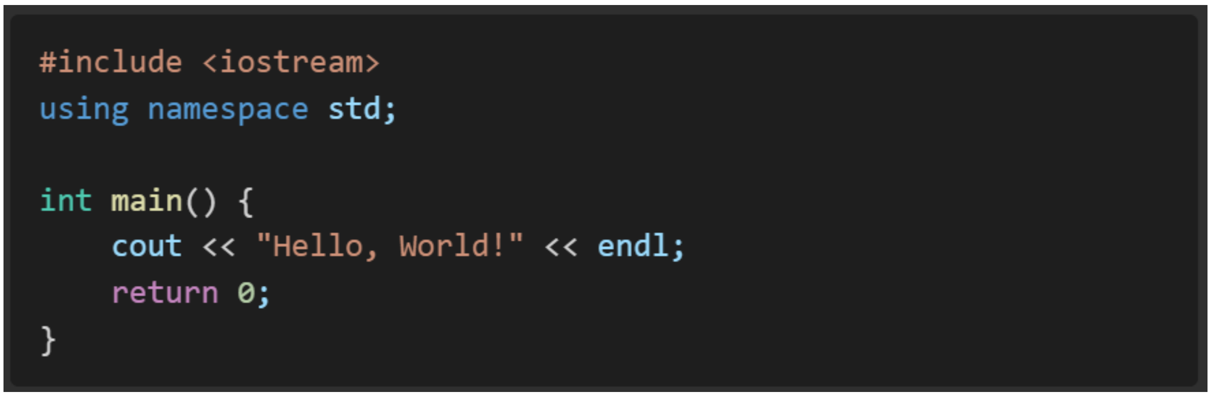
1. **Widely Used:** C++ is employed in various domains, including system programming, game development, embedded systems, and competitive programming.
2. **Performance:** It offers high performance and efficient memory management, making it suitable for resource-intensive applications.
3. **Versatility:** Combines low-level programming capabilities with high-level abstractions.
4. **Industry Relevance:** Learning C++ builds a strong foundation for understanding other programming languages like Java, Python, and C#.

Key Features

1. **Object-Oriented Programming (OOP):** Classes, inheritance, polymorphism, encapsulation, and abstraction.
2. **Low-Level Manipulation:** Supports pointer arithmetic, direct memory access.
3. **Standard Template Library (STL):** Provides a rich set of data structures and algorithms.
4. **Platform Independence:** Code can be compiled and run on different platforms with minimal changes.
5. **High Performance:** Closer to hardware, hence faster execution.

Basic Syntax

Hello World Program:



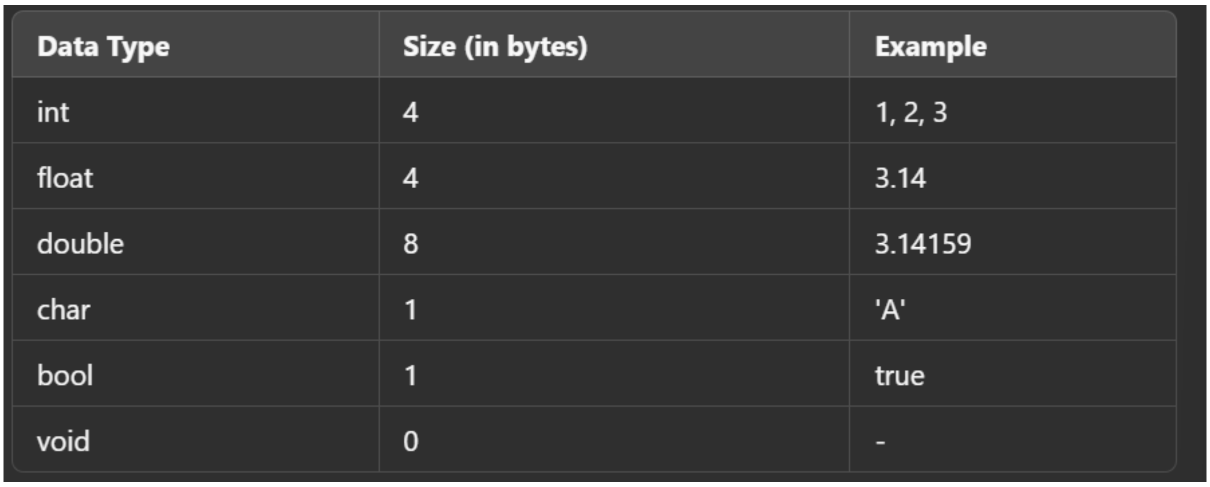
```cpp
#include <iostream>
using namespace std;

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

Explanation:

- #include <iostream>: Includes the Input/Output stream library for standard input and output operations.
- using namespace std;: Allows the use of standard library functions without prefixing them with std::.
- int main(): Entry point of the program.
- cout: Used to display output.

Data Types

Fundamental Data Types:



Data Type	Size (in bytes)	Example
int	4	1, 2, 3
float	4	3.14
double	8	3.14159
char	1	'A'
bool	1	true
void	0	-

Theory:

- **Integer (int):** Used to store whole numbers.
- **Floating-Point (float/double):** Used to store decimal numbers. Double has higher precision.
- **Character (char):** Used to store single characters or ASCII values.
- **Boolean (bool):** Represents true or false values.
- **Void:** Represents the absence of a value, often used for functions that do not return anything.

Derived Data Types:

- **Array**: int arr[10];
- **Pointer**: int* ptr;
- **Reference**: int& ref = x;
- **Function**: void myFunction();

User-Defined Data Types:

- **Struct**
- **Class**
- **Enum**
- **Typedef/using**

Types of Errors

1. **Syntax Errors:** Mistakes in the program's syntax, such as missing semicolons or incorrect keywords. These are caught by the compiler.

```
int x = 10  // Missing semicolon
```

2. **Runtime Errors:** Errors that occur during program execution, such as division by zero or accessing invalid memory.

```
int a = 10, b = 0;
cout << a / b; // Division by zero
```

3. **Logical Errors:** Errors in the program's logic, leading to incorrect results. These are the hardest to detect.

```
int sum = 2 * 5; // Logic mistake instead of addition
```

4. **Linker Errors:** Occur when the program refers to undefined functions or libraries.

```
void myFunction();
int main() {
   myFunction(); // Error: Function not defined
}
```

5. **Semantic Errors:** Misuse of language constructs that violate the program's meaning.

Control Flow

Conditional Statements:

Conditional statements allow programs to execute different code blocks based on certain conditions. The if statement checks the condition, and if it evaluates to true, the associated block of code executes. If false, the else block executes.

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

Loops:

1. **For Loop**:

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

2. **While Loop**:

```
while (condition) {
    // Code
}
```

3. **Do-While Loop**:

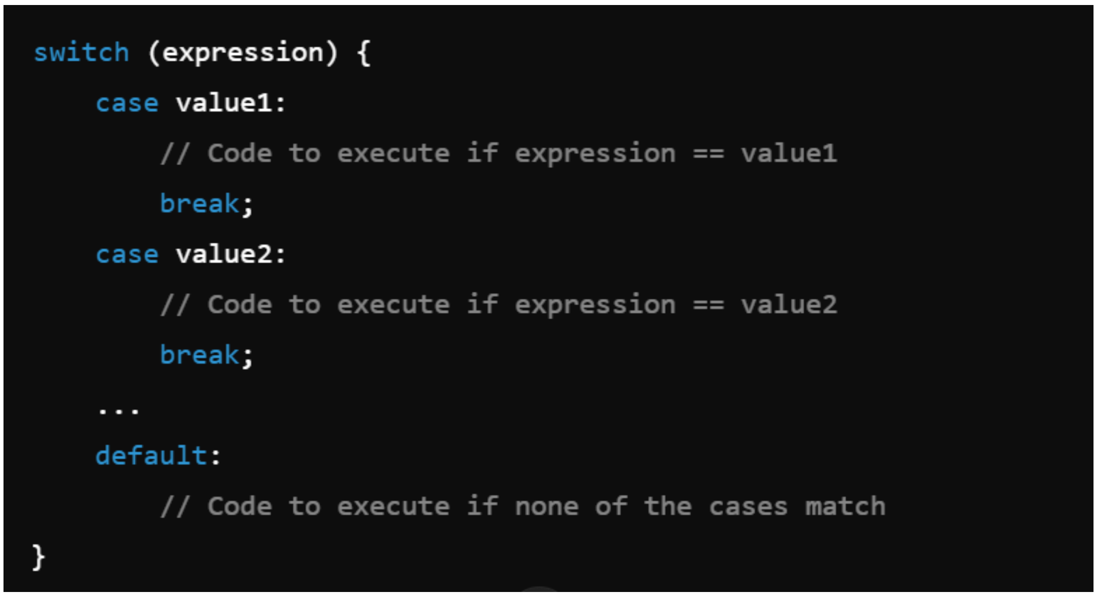
```
do {
    // Code
} while (condition);
```

Theory:

Loops are used to execute a block of code repeatedly. The for loop is used when the number of iterations is known. The while loop is preferred when the condition is checked before the first iteration. The do-while loop guarantees at least one execution of the code block.

Switch-Case Statement

The switch statement allows you to execute one block of code from multiple options based on a variable's value. It's a great alternative to using multiple if-else statements.


```
switch (expression) {
    case value1:
        // Code to execute if expression == value1
        break;
    case value2:
        // Code to execute if expression == value2
        break;
    ...
    default:
        // Code to execute if none of the cases match
}
```

TAKE THIS EXAMPLE:

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

int main() {
   int day;
   cout << "Enter a day number (1-7): ";
   cin >> day;

   switch (day) {
      case 1:
         cout << "Monday" << endl;
```

```
            break;
        case 2:
            cout << "Tuesday" << endl;
            break;
        case 3:
            cout << "Wednesday" << endl;
            break;
        case 4:
            cout << "Thursday" << endl;
            break;
        case 5:
            cout << "Friday" << endl;
            break;
        case 6:
            cout << "Saturday" << endl;
            break;
        case 7:
            cout << "Sunday" << endl;
            break;
        default:
            cout << "Invalid day!" << endl;
    }
    return 0;
}
```

Expression: The switch statement evaluates the expression once and compares it with each case.
Break Statement: Exits the switch block after a matching case is executed to prevent fall-through.
Default Case: Executes when no case matches (optional).
Limitations: Only works with integral or enumerated types (int, char, etc.).

Functions

Functions in C++ are reusable blocks of code that perform a specific task. They allow you to group code into logical units,

making your program more organized and easier to read. Functions can take input parameters, perform operations, and return a value.

Key Points:

Function Definition: This includes the return type, function name, parameters, and the body of the function.

Function Call: You invoke a function by its name followed by parentheses. If the function takes parameters, you pass the arguments inside the parentheses.

Return Type: The type of value that the function returns to the caller. If no value is returned, the return type is void.

Declaration and Definition:

```
int add(int a, int b) {
    return a + b;
}
```

Function Overloading:

```
int sum(int a, int b);
float sum(float a, float b);
```

Inline Functions:

```
inline int square(int x) {
    return x * x;
}
```

Object-Oriented Programming (OOP)

1. What is OOP in C++?

Object-Oriented Programming in C++ is a programming style that organizes software design around data (objects) and the functions that operate on the data (methods).

As you proceed, please read these carefully:

4. **Class:** Blueprint.
5. **Object:** Instance of the class.
6. **Encapsulation:** Hiding data with private and accessing it through public methods.
7. **Inheritance:** Inheriting properties and methods from another class.
8. **Polymorphism:** Ability to use the same method name for different functions (Overloading/Overriding).

9. **Abstraction:** Hiding complex details and showing only the necessary parts (abstract classes).

2. Basic OOP Concepts in C++

a) Classes and Objects

- **Class:** A blueprint for creating objects. It defines a datatype.
- **Object:** An instance of a class.

Example:

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

// Class definition
class Dog {
public:
    string name;  // Attribute (data member)
    string breed;

    // Method (function) inside the class
    void speak() {
        cout << name << " says Woof!" << endl;
    }
};

int main() {
    // Creating an object of Dog class
    Dog myDog;
    myDog.name = "Buddy";  // Assigning values to object attributes
    myDog.breed = "Golden Retriever";

    myDog.speak();  // Calling the method
    return 0;
}
```

b) Encapsulation

- **Encapsulation** is the concept of hiding the internal details of how an object works and only exposing what is necessary.

Example:

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

class Account {
private:
    double balance;  // Private attribute

public:
    Account(double balance) {  // Constructor to initialize balance
        this->balance = balance;
    }
    void deposit(double amount) {
        balance += amount;  // Method to modify the balance
    }

    double getBalance() {
        return balance;  // Accessor method to get the balance
    }
};

int main() {
    Account acc(1000);  // Creating Account object with initial balance
    acc.deposit(500);    // Depositing money
    cout << "Balance: " << acc.getBalance() << endl;  // Displaying balance
    return 0;
}
```

c) Inheritance

- **Inheritance** allows one class (child class) to inherit the properties and methods of another class (parent class).

Example:

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

class Animal {
public:
    void eat() {
        cout << "Eating..." << endl;
    }
};

class Dog : public Animal {  // Dog inherits from Animal
public:
    void bark() {
        cout << "Barking!" << endl;
    }
};

int main() {
    Dog myDog;
    myDog.eat();  // Inherited method
    myDog.bark();  // Dog's own method
    return 0;
}
```

d) Polymorphism

- **Polymorphism** allows one interface to be used for a general class of actions. It can be achieved through **function overloading** or **method overriding**.

Example (Function Overloading):

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

class Printer {
public:
    void print(int num) {
        cout << "Printing number: " << num << endl;
    }

    void print(string text) {
        cout << "Printing text: " << text << endl;
    }
};

int main() {
    Printer p;
    p.print(10);      // Prints number
    p.print("Hello"); // Prints text
    return 0;
}
```

Example (Method Overriding):

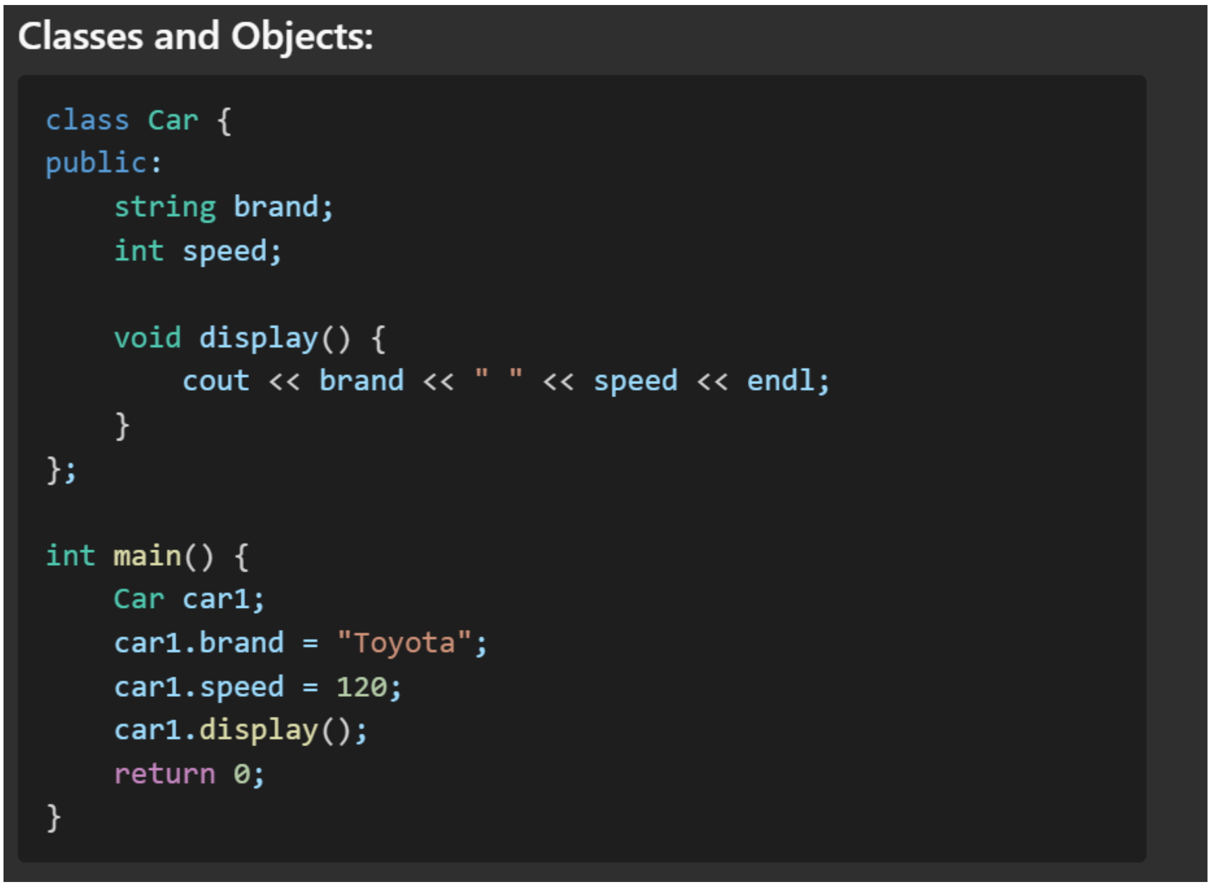
```
#include <iostream>
using namespace std
class Animal {
public:
    virtual void sound() {  // Virtual method
        cout << "Animal sound!" << endl;
    }
};

class Dog : public Animal {
public:
    void sound() override {  // Overriding the base class method
        cout << "Woof!" << endl;
    }
};

int main() {
    Animal* animal = new Dog();
    animal->sound();  // Calls Dog's version of sound()
    return 0;
}
```

e) Abstraction

- **Abstraction** hides complex implementation details and shows only the essential features of the object.

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

class Shape {
public:
    virtual void draw() = 0;  // Pure virtual function
};

class Circle : public Shape {
public:
    void draw() override {
        cout << "Drawing Circle" << endl;
    }
};

int main() {
    Shape* shape = new Circle();  // Circle object
    shape->draw();  // Calls Circle's draw method
    return 0;
}
```

REVISE!



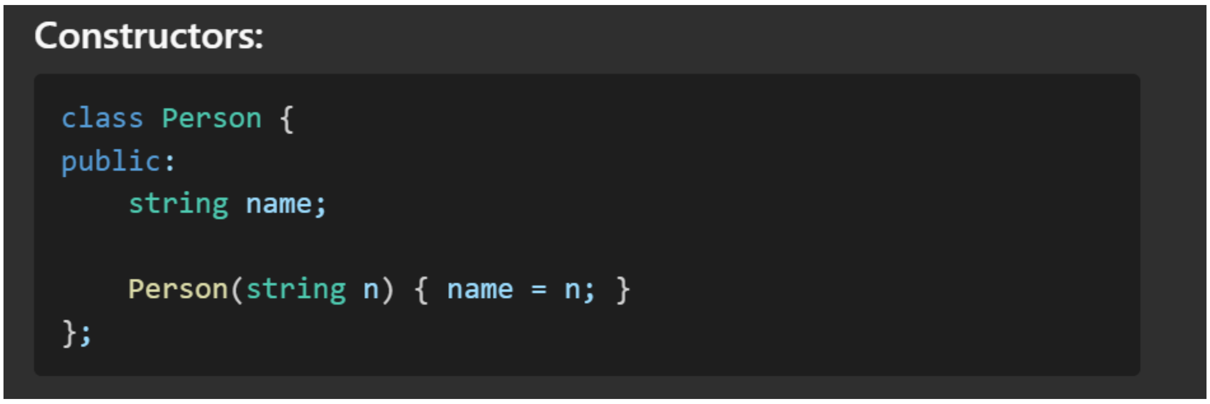
Classes and Objects:

```
class Car {
public:
    string brand;
    int speed;

    void display() {
        cout << brand << " " << speed << endl;
    }
};

int main() {
    Car car1;
    car1.brand = "Toyota";
    car1.speed = 120;
    car1.display();
    return 0;
}
```

- **Class:** A blueprint for creating objects. It encapsulates data and functions that operate on the data.
- **Object:** An instance of a class. Objects represent real-world entities in OOP.

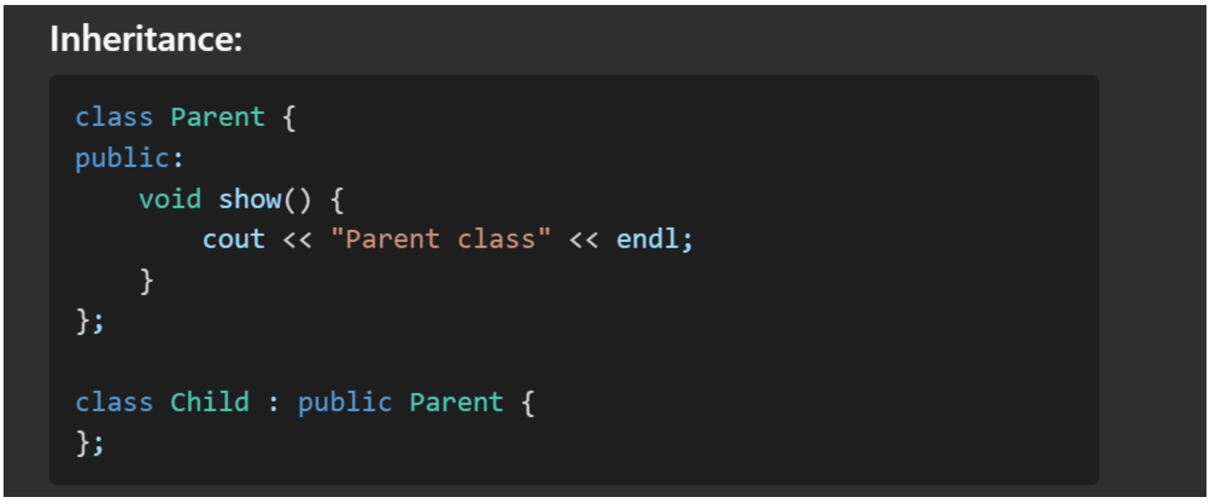
Four Pillars of OOP:

- **Encapsulation:** Bundling data and methods together while restricting access using access modifiers (e.g., private, protected, public).
- **Inheritance:** Mechanism for creating a new class from an existing one to reuse code.
- **Polymorphism:** Ability to call the same method on different objects and have different behaviors (e.g., function overriding).
- **Abstraction:** Hiding implementation details and exposing only essential features using abstract classes or interfaces.



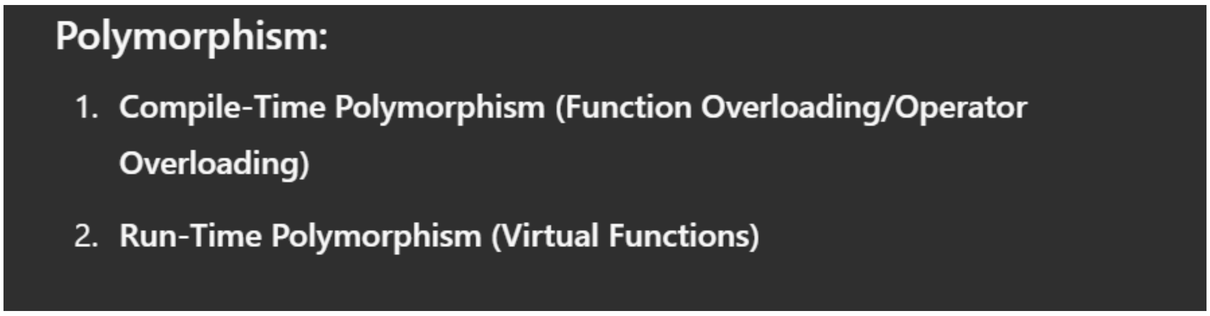
Constructors:

```
class Person {
public:
    string name;

    Person(string n) { name = n; }
};
```



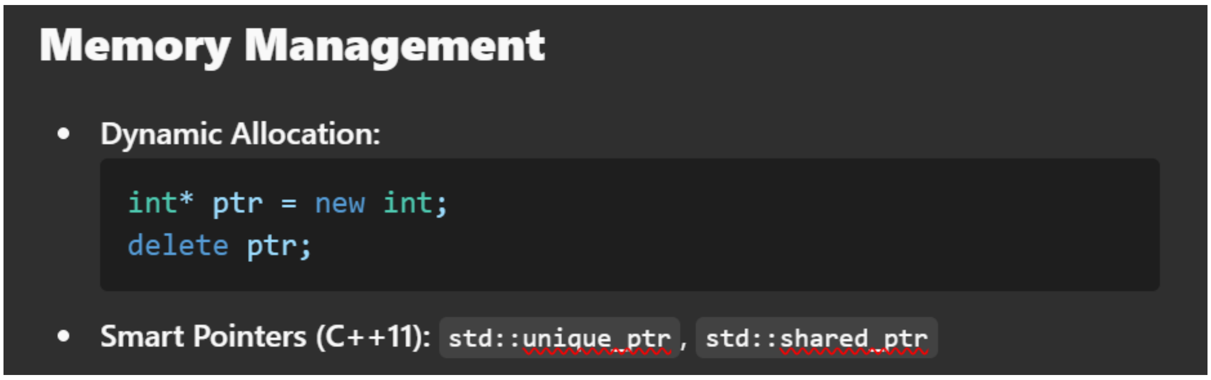
Inheritance:

```
class Parent {
public:
    void show() {
        cout << "Parent class" << endl;
    }
};

class Child : public Parent {
};
```



Polymorphism:

1. Compile-Time Polymorphism (Function Overloading/Operator Overloading)
2. Run-Time Polymorphism (Virtual Functions)



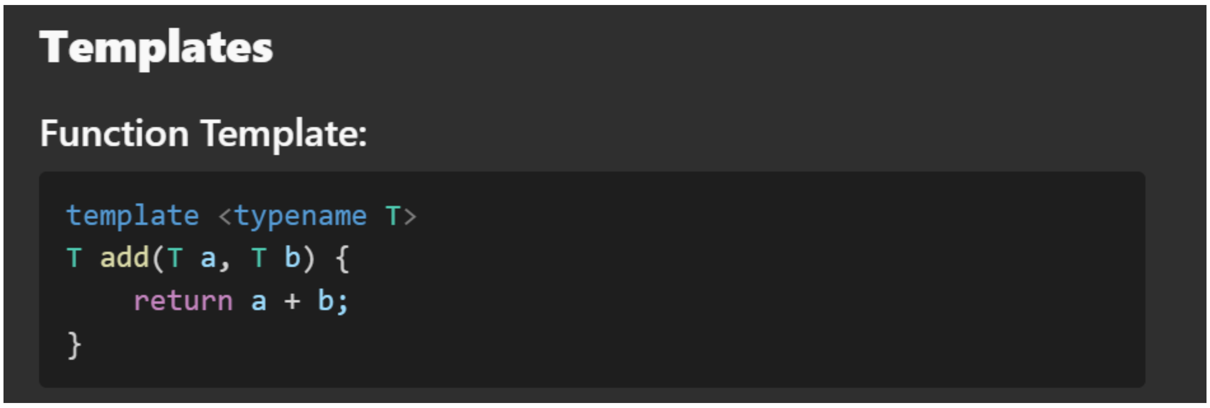
Memory Management

- Dynamic Allocation:

```
int* ptr = new int;
delete ptr;
```

- Smart Pointers (C++11): `std::unique_ptr`, `std::shared_ptr`

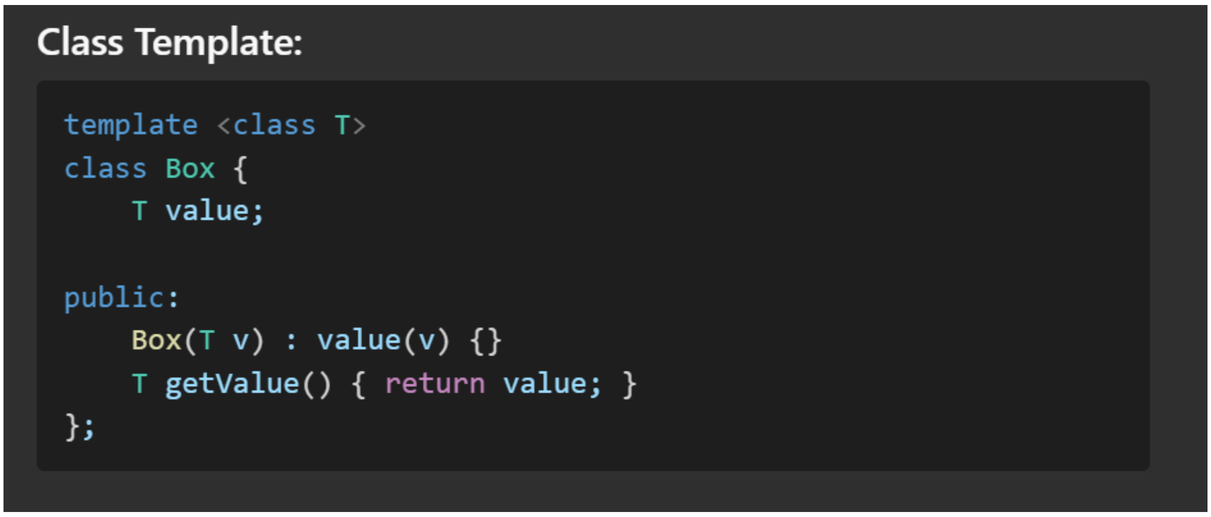
Memory management ensures efficient allocation and deallocation of memory during program execution. C++ provides manual control through dynamic memory allocation using new and delete. It also introduces smart pointers (e.g., unique_ptr, shared_ptr) to automate memory management and prevent memory leaks.



Templates

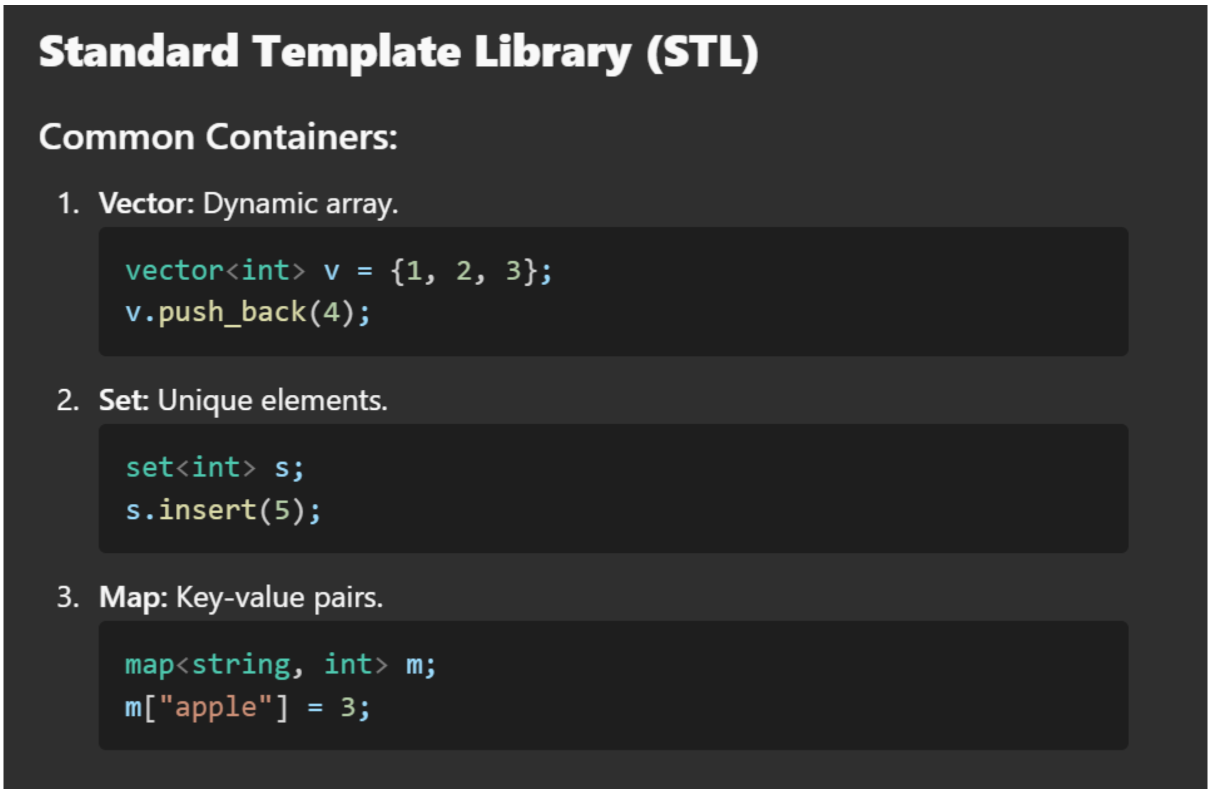
Function Template:

```
template <typename T>
T add(T a, T b) {
    return a + b;
}
```



Class Template:

```
template <class T>
class Box {
    T value;

public:
    Box(T v) : value(v) {}
    T getValue() { return value; }
};
```



Standard Template Library (STL)

Common Containers:

1. **Vector**: Dynamic array.

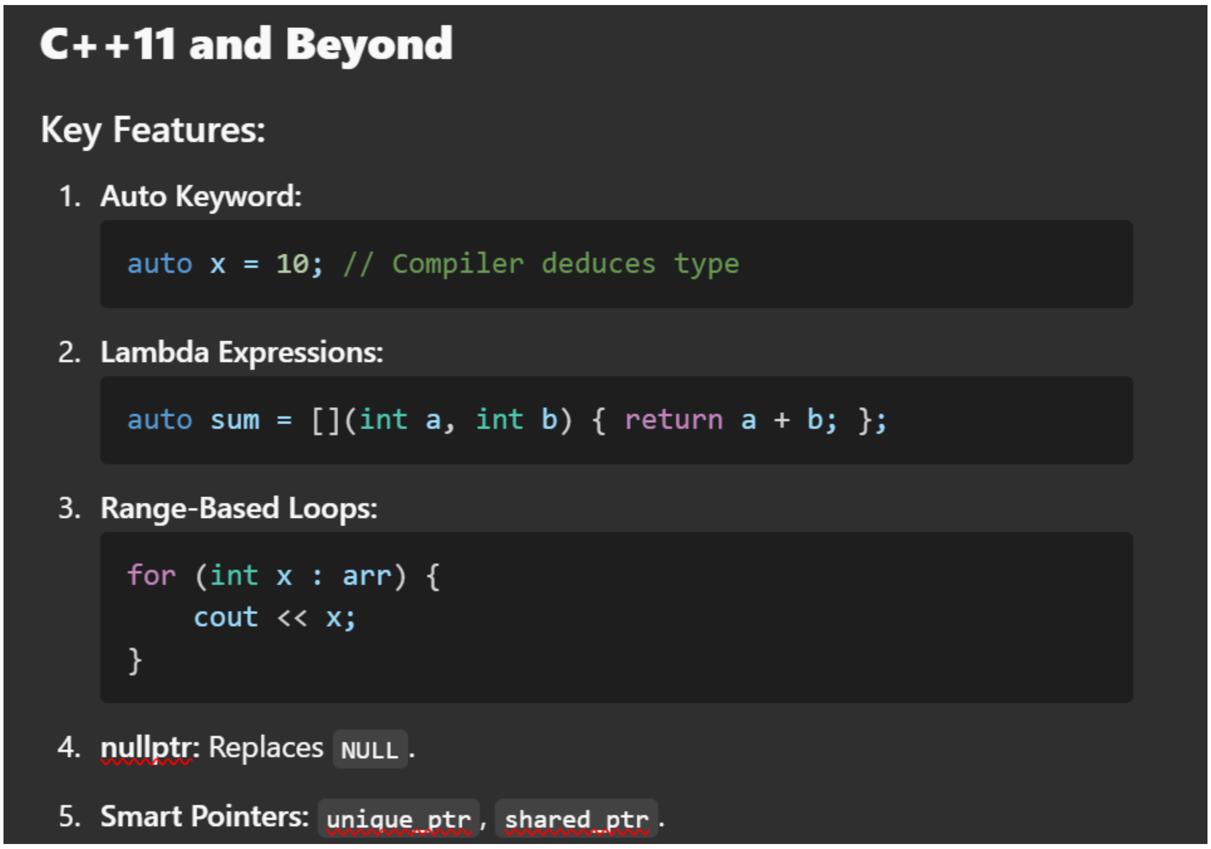
   ```
   vector<int> v = {1, 2, 3};
   v.push_back(4);
   ```

2. **Set**: Unique elements.

   ```
   set<int> s;
   s.insert(5);
   ```

3. **Map**: Key-value pairs.

   ```
   map<string, int> m;
   m["apple"] = 3;
   ```



C++11 and Beyond

Key Features:

1. **Auto Keyword:**

```cpp
auto x = 10; // Compiler deduces type
```

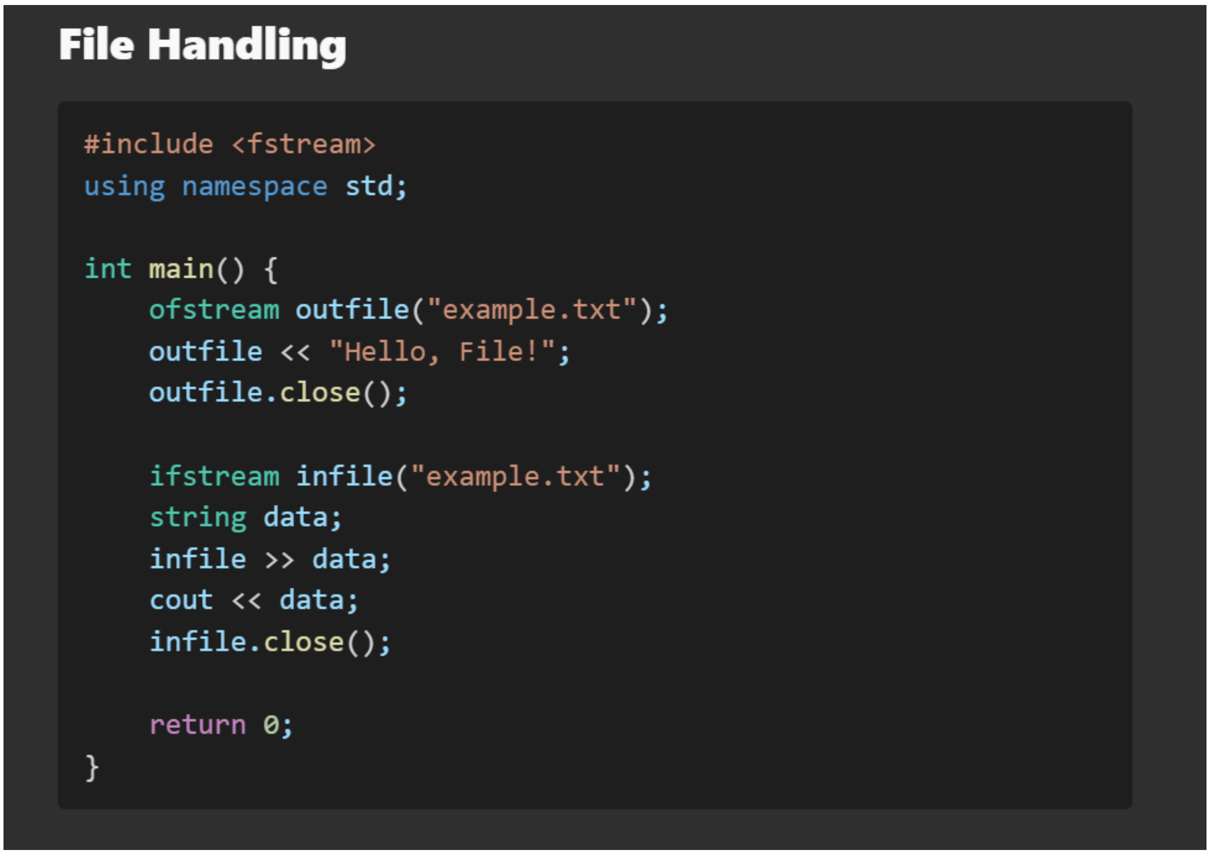
2. **Lambda Expressions:**

```cpp
auto sum = [](int a, int b) { return a + b; };
```

3. **Range-Based Loops:**

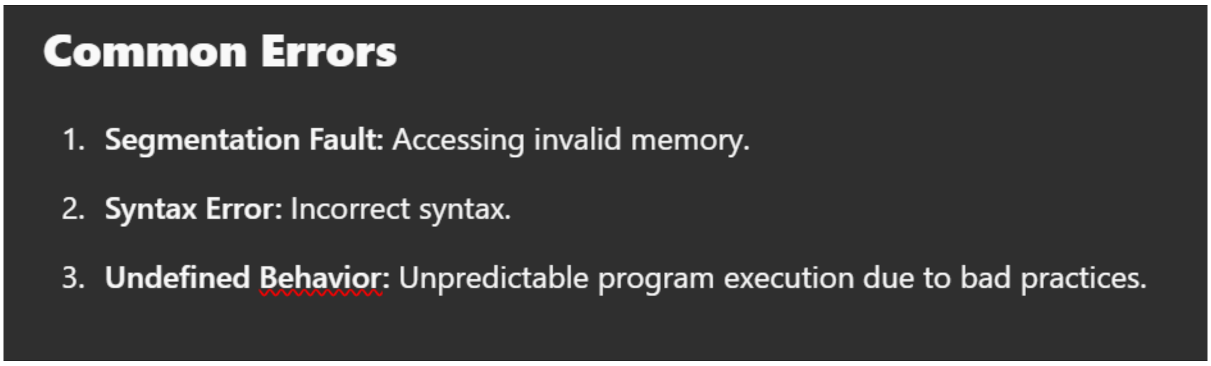
```cpp
for (int x : arr) {
    cout << x;
}
```

4. **nullptr:** Replaces `NULL`.
5. **Smart Pointers:** `unique_ptr`, `shared_ptr`.



File Handling

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

int main() {
    ofstream outfile("example.txt");
    outfile << "Hello, File!";
    outfile.close();

    ifstream infile("example.txt");
    string data;
    infile >> data;
    cout << data;
    infile.close();

    return 0;
}
```



Common Errors

1. **Segmentation Fault:** Accessing invalid memory.
2. **Syntax Error:** Incorrect syntax.
3. **Undefined Behavior:** Unpredictable program execution due to bad practices.

A Final Thought

So, hey, if you've read this far, can I ask you something?

Man, I'm just tired of this world's cruelty. We're all so divided by **comparison, jealousy, and hatred.** But deep down, don't we all want the same things; **love, kindness, and a little peace?**

I'm not saying I'm some perfect person, I'm far from it. I've made my mistakes, and I'm no saint. But I can still see this:

This world doesn't need more hate. It needs love. It needs people who can stand together, who can look past differences and just try to make things a little better.

Even a small step, **a kind word, a helping hand, or just a little understanding** can create a ripple effect.

So, I'm asking you: **Do something.** Do something to make this world a better place. It doesn't matter who you are or what you've done, just try. Because if enough of us try, maybe, just maybe, we can make a difference.

And Yeah, Who am i?
I am Bilal Ahmad Khan AKA Mr. BILRED

GOOD LUCK
TAKE CARE!