# C++ Study Notes
## Introduction to C++
- **Definition**: C++ is a high-level programming language developed by Bjarne Stroustrup in 1979, known for its performance and flexibility, and is widely used for system/software development and game programming.
- **Paradyms**: It supports procedural, object-oriented, and generic programming paradigms.
### Key Features of C++
- Object-oriented programming (OOP)
- Low-level manipulation
- Rich library support
- Portability
- Performance
- Memory management
---
## Setting Up C++
### Installing a Compiler
- **Compilers**: Easily work with a popular compiler like GCC or Visual Studio.
- **IDE**: Integrated Development Environments (IDE) like Code::Blocks, Eclipse, or Visual Studio to write and manage code.
### First Program
```cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello, World!" << endl;
    return 0;
}
```
- **Explanation**:
  - `#include <iostream>`: Preprocessor directive that includes the input-output stream library.
  - `using namespace std;`: Enables the use of standard library names without the `std::` prefix.
  - `cout`: Standard output stream.
  - `endl`: Ends the line and flushes the output buffer.
---
## C++ Basics
### Variables and Data Types
- **Definition**: Variables are containers for storing data values.
- **Data Types**:
  - `int`: Integer Type
  - `float`: Floating Point Type
  - `double`: Double Precision Float
  - `char`: Character Type
  - `bool`: Boolean Type
### Operators
- **Arithmetic Operators**: +, -, *, /, %
- **Relational Operators**: ==, !=, <, >, <=, >=
- **Logical Operators**: && (AND), || (OR), ! (NOT)
- **Bitwise Operators**: &, |, ^, ~, <<, >>
### Control Structures
#### Conditional Statements

- **if Statement**:
```cpp
if (condition) {
    // code to execute if condition is true
}
```
- **switch Case**:
```cpp
switch (variable) {
    case value1:
        // code
        break;
    case value2:
        // code
        break;
    default:
        // default case
}
```
#### Loops
- **for Loop**:
```cpp
for (initialization; condition; increment) {
    // code
}
```
- **while Loop**:
```cpp
while (condition) {
    // code
}
```
- **do-while Loop**:
```cpp
do {
    // code
} while (condition);
```
---
## Functions in C++
### Definition
- **Function**: A block of code designed to perform a specific task.
- **Syntax**:
```cpp
returnType functionName(parameters) {
    // function body
}
```
### Passing Arguments
- **By Value**: Copies the actual value of an argument into the function.
- **By Reference**: Uses references to access and modify the original variable.
### Example
```cpp
```

```
int add(int a, int b) {
    return a + b;
}
```
---
## Object-Oriented Programming (OOP) Concepts
### Classes and Objects
- **Class**: A blueprint for creating objects (contains data and methods).
- **Object**: An instance of a class.
#### Example
```cpp
class Car {
public:
    string brand;
    void drive() {
        cout << "Driving a car" << endl;
    }
};
Car myCar; // Object creation
myCar.brand = "Toyota";
myCar.drive();
```
### Encapsulation
- **Definition**: Bundling the data and methods that operate on the data within one unit (class).
### Inheritance
- **Definition**: Mechanism where a new class derives properties and behavior from an existing class.
#### Example
```cpp
class Vehicle {
public:
    void honk() {
        cout << "Honk!" << endl;
    }
};
class Bike : public Vehicle { }; // Bike inherits Vehicle
Bike b;
b.honk(); // Outputs: Honk!
```
### Polymorphism
- **Definition**: Ability to process objects differently based on their data type or class.
#### Example
```cpp
class Animal {
public:
    virtual void sound() {
        cout << "Animal makes sound" << endl;
    }
};
class Dog : public Animal {
public:
```

```
    void sound() override {
        cout << "Woof!" << endl;
    }
};
Animal* a = new Dog();
a->sound(); // Outputs: Woof!
```
---
## Advanced Concepts
### Templates
- **Definition**: Enables writing generic programs.
#### Example
```cpp
template <typename T>
T max(T a, T b) {
    return (a > b) ? a : b;
}
```
### Exception Handling
- **Definition**: Mechanism to handle runtime errors.
#### Example
```cpp
try {
    // code that may throw an exception
} catch (exception &e) {
    // handle exception
}
```
---
## Conclusion & Summary
- **C++ Overview**: A versatile language that supports multiple programming paradigms.
- **Core Components**: Variables, data types, functions, control structures, and OOP principles.
- **Advanced Features**: Templates and exception handling allow for robust and reusable code.
**Remember**: Practice coding, understand the logic behind each construct, and gradually explore more complex topics as you become more comfortable with the language!