Fundamentals of Computer Programming

CS-110

Course Instructor: Dr. Momina Moetesum

Getting Started with C++ Programming

Week 1

```
51 template <typename T, typename I>
52 void TestIntegral(const T values[], int num_values) {
53   for (int i = 0; i < num_values; ++i) {
54     T t0 = values[i];
55     I i0 = absl::bit_cast<I>(t0);
56     T t1 = absl::bit_cast<T>(i0);
57     I i1 = absl::bit_cast<I>(t1);
58     ASSERT_EQ(0, memcmp(&t0, &t1, sizeof(T)));
59     ASSERT_EQ(i0, i1);
60   }
61 }
62
63 TEST(BitCast, Bool) {
64   static const bool bool_list[] = { false, true };
65   TestMarshall<bool>(bool_list, ABSL_ARRAYSIZE(bool_list));
66 }
67
68 TEST(BitCast, Int32) {
69   static const int32_t int_list[] =
70     { 0, 1, 100, 2147483647, -1, -100, -2147483647, -2147483647-1 };
71   TestMarshall<int32_t>(int_list, ABSL_ARRAYSIZE(int_list));
72 }
73
74 TEST(BitCast, Int64) {
75   static const int64_t int64_list[] =
76     { 0, 1, 1LL << 40, -1, -(1LL<<40) };
77   TestMarshall<int64_t>(int64_list, ABSL_ARRAYSIZE(int64_list));
```

Learning Objectives

- Understand what programming is
- Why we use C++
- Write & run first program
- Learn program structure (main, cout)
- Gain confidence as beginner programmers

What is Programming?

```
template <typename T, typename I>
void TestIntegral(const T values[], int num_values) {
 for (int i = 0; i < num_values; ++i) {
  T t0 = values[i];
  I i0 = absl::bit_cast<I>(t0);
  T t1 = absl::bit_cast<T>(i0);
  I i1 = absl::bit_cast<I>(t1);
  ASSERT_EQ(0, memcmp(&t0, &t1, sizeof(T)));
  ASSERT_EQ(i0, i1);
 }
}

TEST(BitCast, Bool) {
 static const bool bool_list[] = { false, true };
 TestMarshall<bool>(bool_list, ABSL_ARRAYSIZE(bool_list));
}

TEST(BitCast, Int32) {
 static const int32_t int_list[] =
  { 0, 1, 100, 2147483647, -1, -100, -2147483647, -2147483647-1 };
 TestMarshall<int32_t>(int_list, ABSL_ARRAYSIZE(int_list));
}

TEST(BitCast, Int64) {
 static const int64_t int64_list[] =
  { 0, 1, 1LL << 40, -1, -(1LL<<40) };
 TestMarshall<int64_t>(int64_list, ABSL_ARRAYSIZE(int64_list));
```

Program

Computer

Application

Why Learn C++?

- C++ is a general-purpose programming language that was developed by **Bjarne Stroustrup** as an enhancement of the C language to add object-oriented paradigm.
- It is considered as a **middle-level** language as it combines features of both **high-level** and **low-level** languages.
- **Syntax similarity** with C, Java, and C# makes it easier to switch languages.
- C++ provides one of the **fastest execution speeds** among high level languages, which can be a deciding factor in Competitive Programming or high-performance applications.

https://www.stroustrup.com/

Main Features of C++

Simple: It is a simple language in the sense that programs can be broken down into logical units and parts, and has a rich library support and a variety of datatypes.

Machine Independent: C++ code can be run on any machine as long as a suitable compiler is provided.

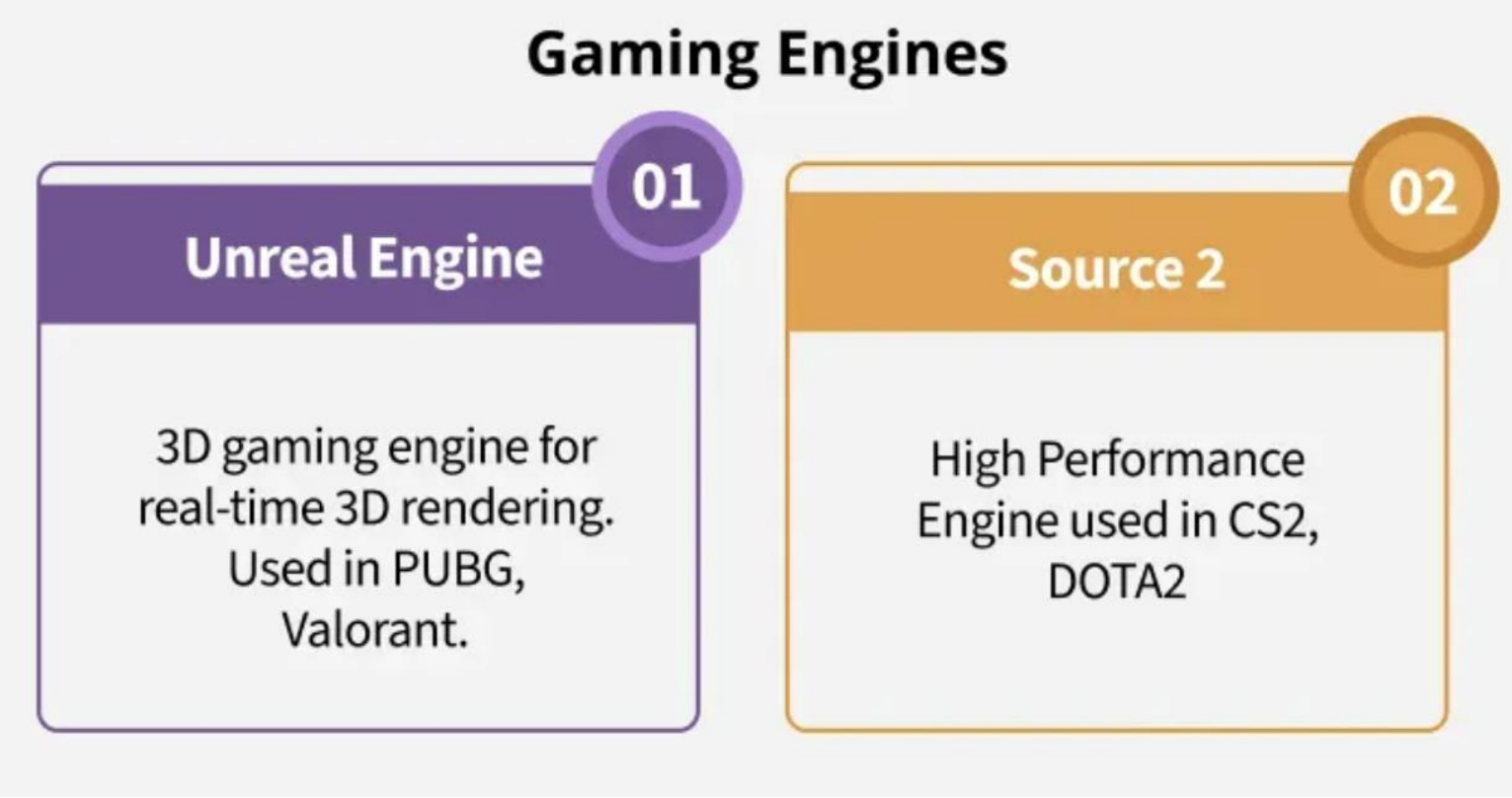
Low-level Access: C++ provides low-level access to system resources, which makes it a suitable choice for system programming and writing efficient code.

Fast Execution Speed: C++ is one of the fastest high-level languages. There is no additional processing overhead in C++, it is blazing fast.

Object-Oriented: One of the strongest points of the language which sets it apart from C. Object-Oriented support helps C++ to make maintainable and extensible programs. i.e. large-scale applications can be built.

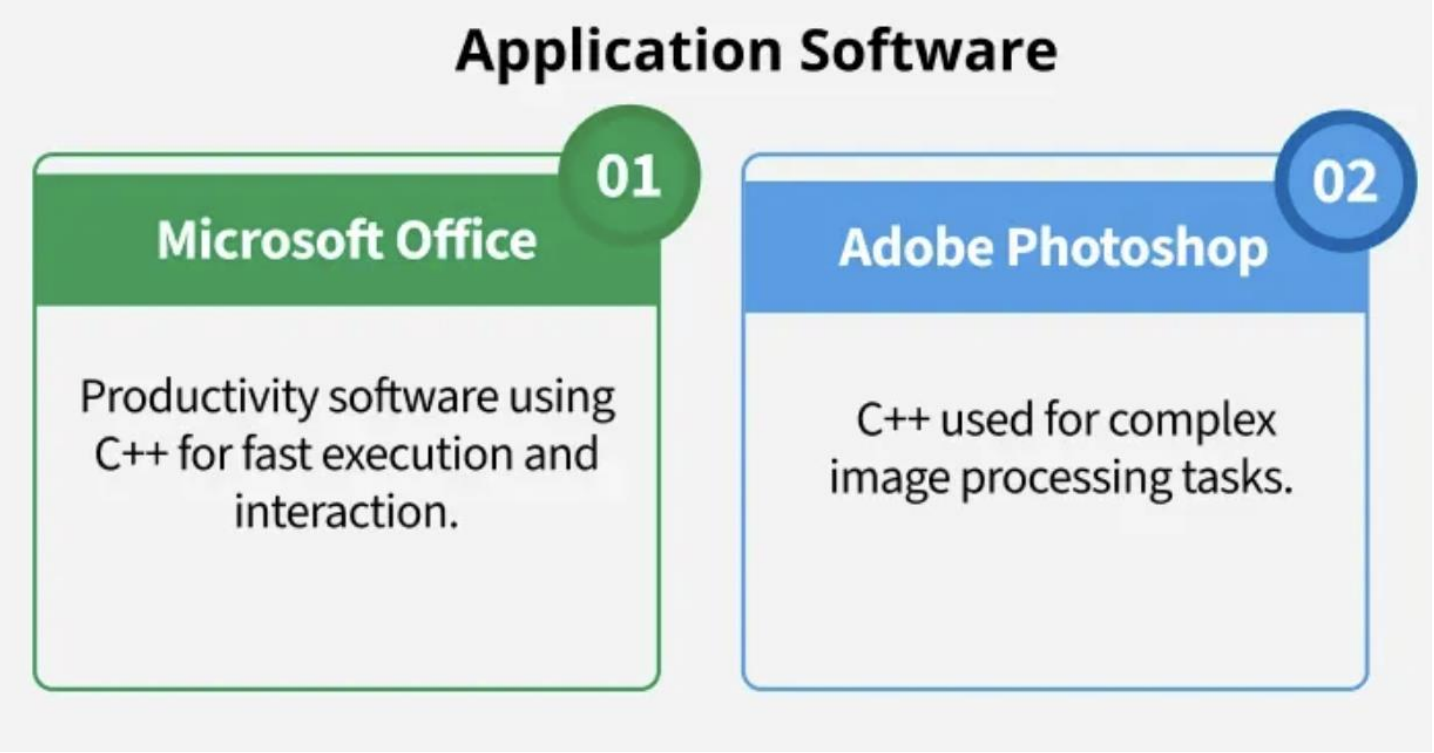
Applications of C++

- C++ is used in a wide range of applications from game engines and application software to operating systems and embedded systems.



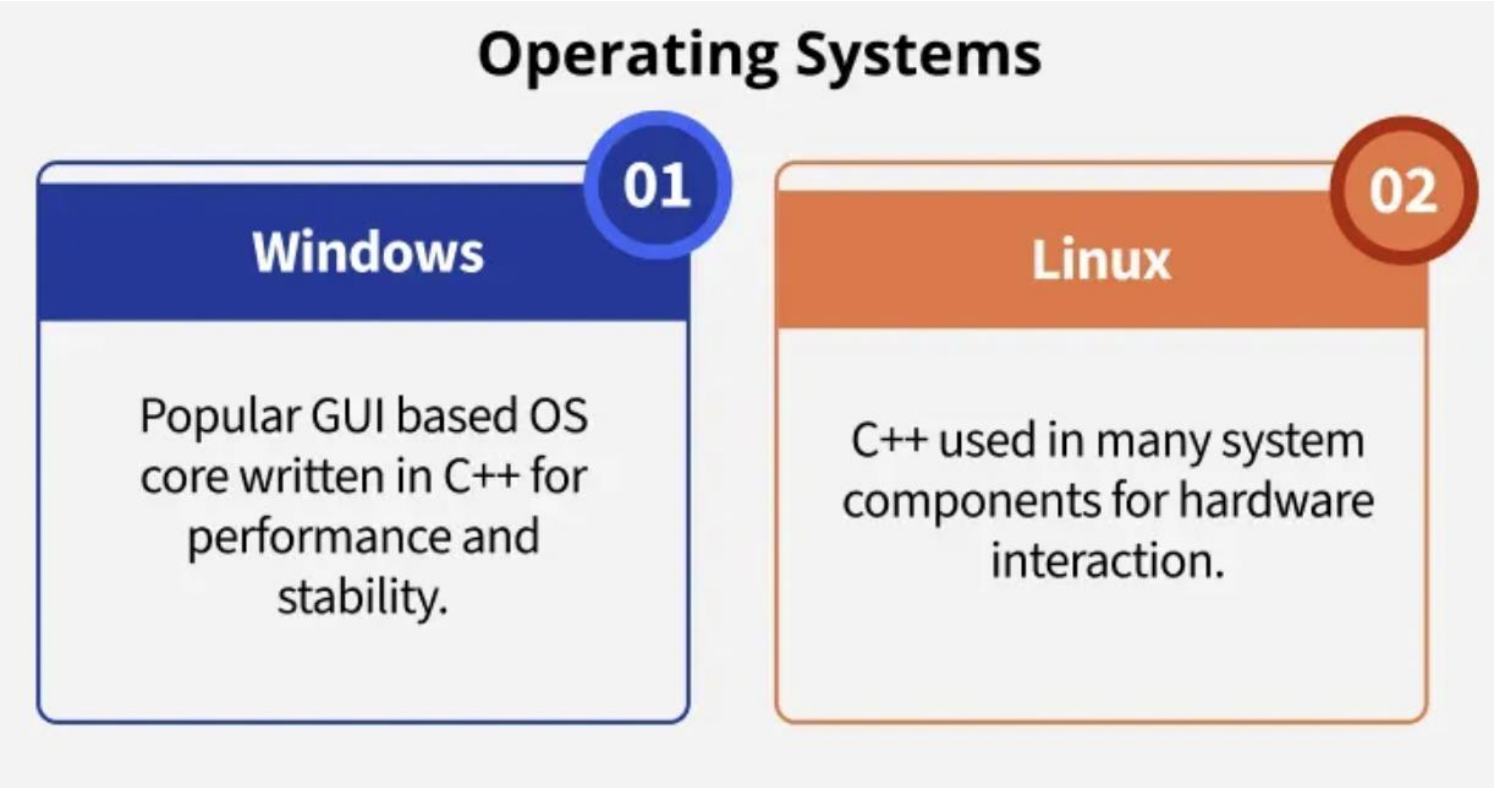
Applications of C++

- C++ is used in a wide range of applications from game engines and application software to operating systems and embedded systems.



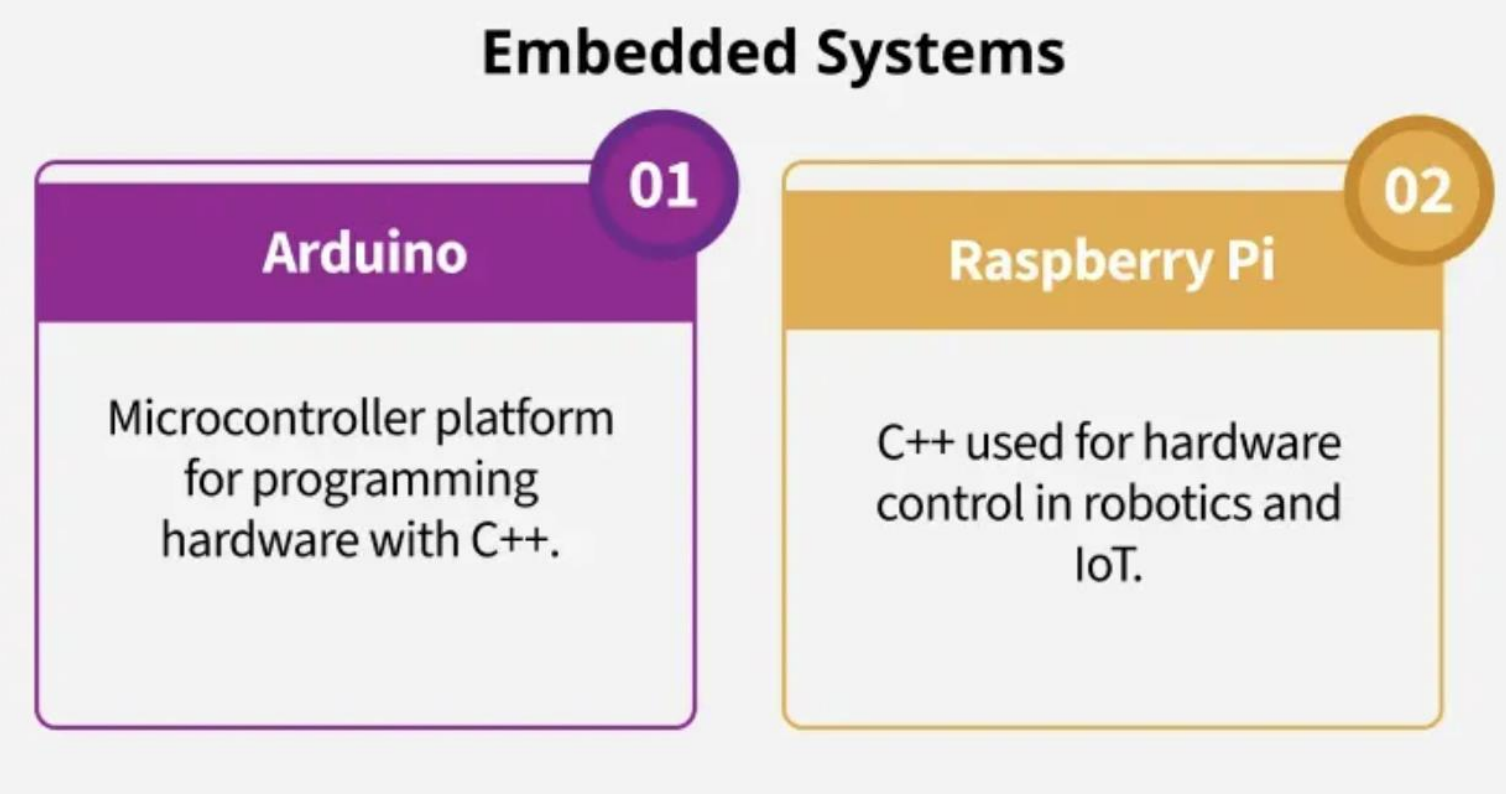
Applications of C++

- C++ is used in a wide range of applications from game engines and application software to operating systems and embedded systems.



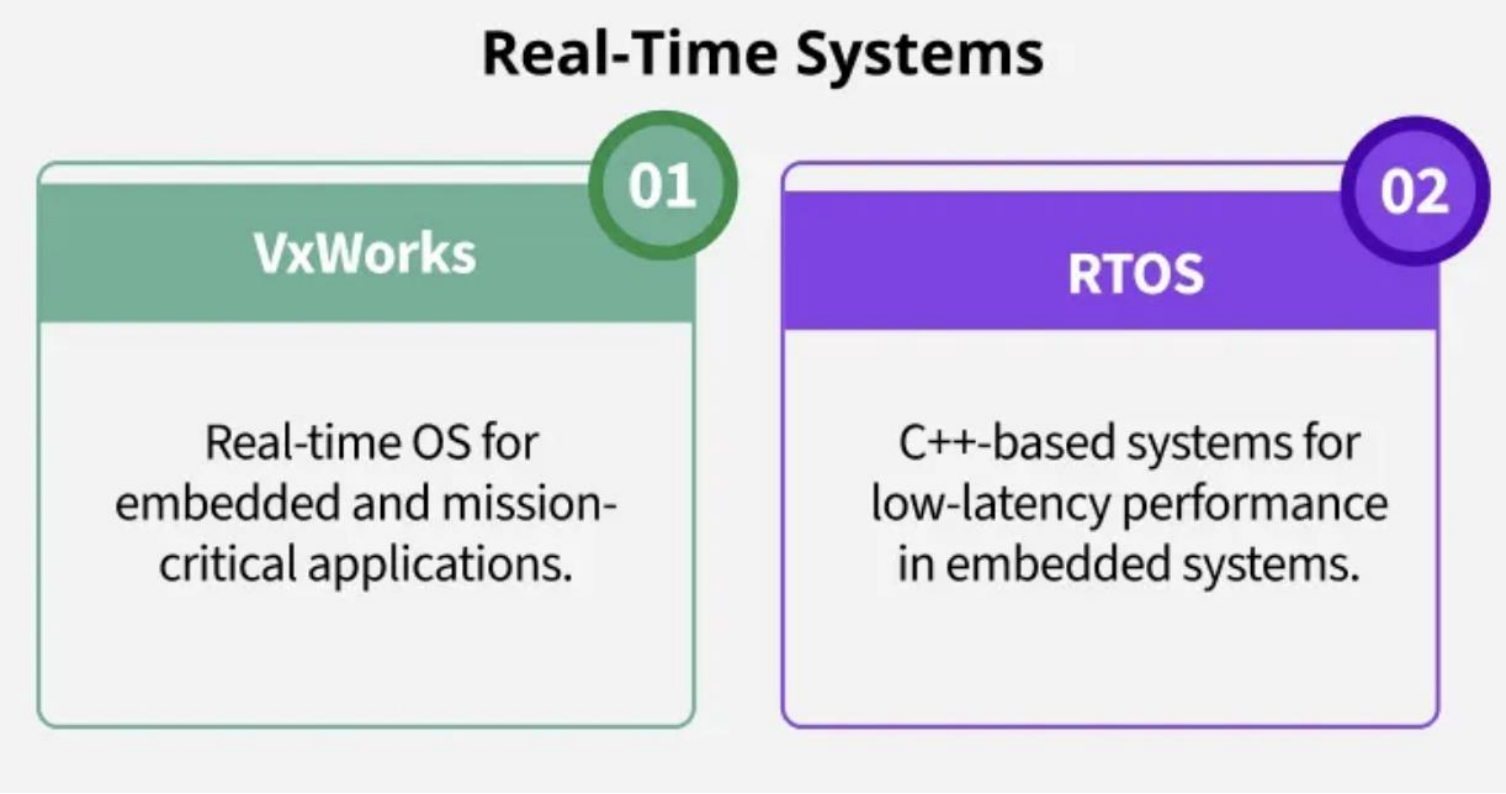
Applications of C++

- C++ is used in a wide range of applications from game engines and application software to operating systems and embedded systems.



Applications of C++

- C++ is used in a wide range of applications from game engines and application software to operating systems and embedded systems.



My First C++ Program

```cpp
// Necessary header files for input output functions
#include <iostream>
using namespace std;

// main() function: where the execution of
// C++ program begins
int main() {

    // This statement prints "Hello World"
    cout << "Hello World";

    return 0;
}
```

- `#include <iostream>` → tools for input/output
- `using namespace std;` → simplifies code
- `int main()` → program starts here
- `{ }` → code block
- `cout` → print to screen
- `return 0;` → end successfully

Output

```
Hello World
```

https://www.geeksforgeeks.org/cpp/writing-first-c-program-hello-world-example/

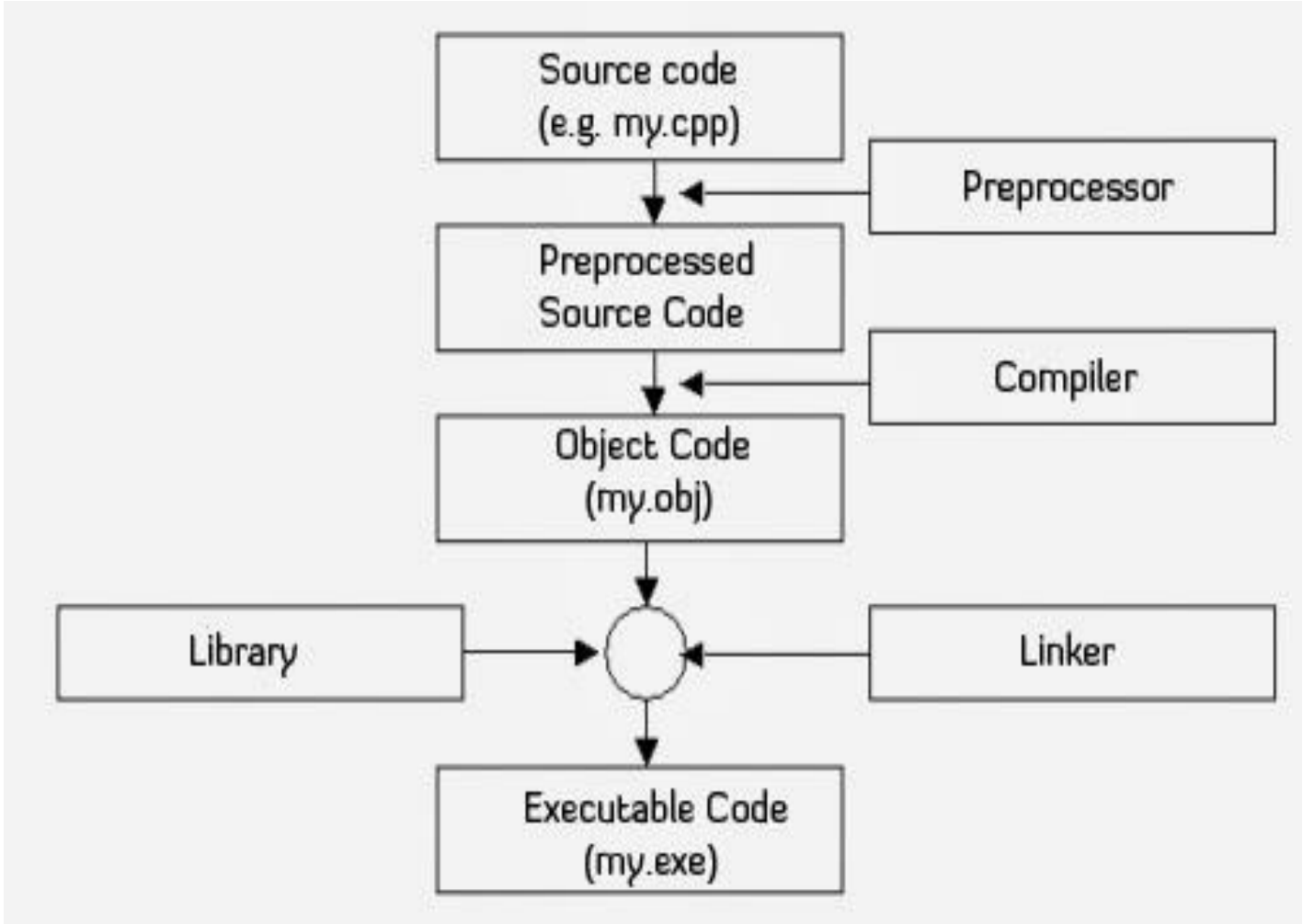
Practice Session

- Task: Modify program to print
  - Your name
  - Your hobby
  - Your dream job

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

int main() {
    cout << "My name is Sara.\n";
    cout << "I love painting.\n";
    cout << "I want to be a software engineer.";
    return 0;
}
```

Coding to Execution



- **Write Code** – You type instructions in C++.
- **Compile** – A compiler (like g++) translates C++ into machine code (0s and 1s).
- **Link** – Combines pieces into one executable file.
- **Run** – The CPU executes instructions and produces output.

How Compilers Works

- **Pre-processing:** First, the pre-processor reads the source code and performs macro expansions, inclusion of header files, and other operations as specified by pre-processor directives (#include, #define, #ifndef, etc.).
- **Compilation:** Second step performs the actual translation of the source code into object code. The object code is a machine-readable representation of the source code, but it is not yet executable.
- **Assembly:** The compiler then passes the object code to an assembler, which converts the object code into assembly code.
- **Linking:** The linker then combines assembly code with any library functions that are required by the program and resolves any references to external symbols/libraries. The linker produces an executable file to run on the target platform.
- **Execution:** Finally, the compiler produces and executable file that runs on the computer system.

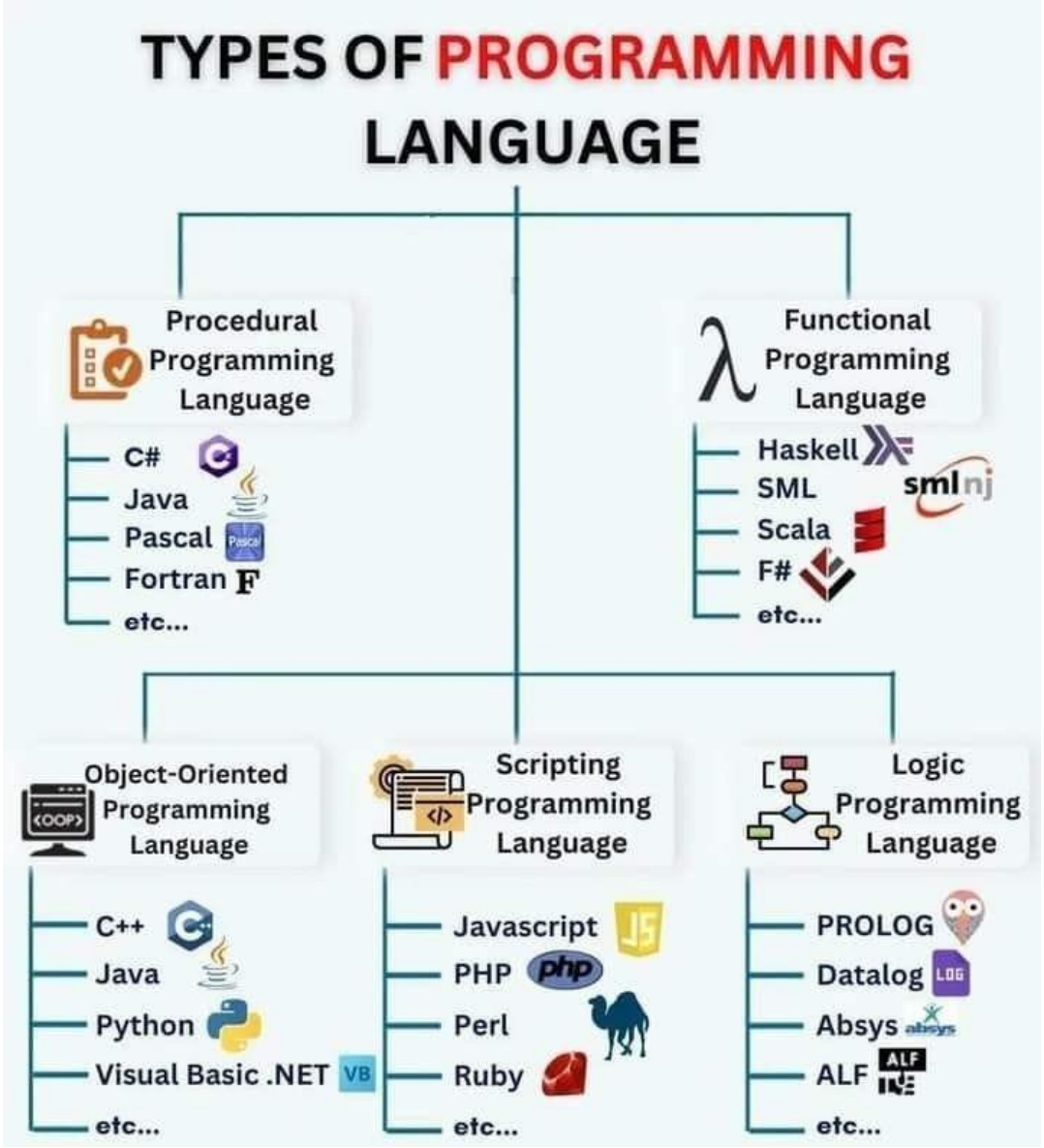
Some Popular C++ Compilers/IDEs

GCC (GNU Compiler Collection)	LLVM Clang	Microsoft Visual C++	C++ Builder	Dev-C++
Digital Mars C/C++	Eclipse IDE for C++	Qt Creator	Intel C++ Compiler	NetBeans IDE

Compiler vs Interpreter

No	Compiler	Interpreter
1	Compiler takes Entire program as input at a time.	Interpreter takes Single instruction as input at a time.
2	Intermediate Object code is generated	No Intermediate Object code is generated
3	It execute conditional control statements fastly.	It execute conditional control statements slower than Compiler
4	More memory is required.	Less memory is required.
5	Program need not to be compiled every time	Every time higher level program is converted into lower level program
6	It display error after entire program is checked	It display error after each instruction interpreted (if any)
7	Example: C, C++	Example: BASIC

Different Programming Languages



Resources

Material presented in these slides are collected from following resources:

- https://icarus.cs.weber.edu/~dab/cs1410/textbook/1.Basics/compiler_op.html
- https://www.sitesbay.com/cpp/cpp-compiler
- https://medium.com/@jepozdemir/programming-language-categories-6b786d70e8f7