Chapter 1

→ Code needs to be "translated" for computers to understand (0 and 1's)

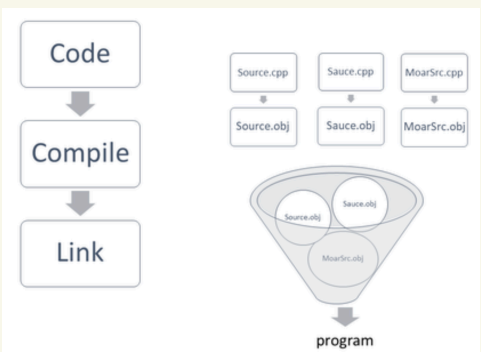
↳ python/JavaScript are Interpreted meaning tools read the code and decide what to do dynamically/at runtime

↳ Java/C# compile to an intermediate language (more efficient than interpreting code every time it is run)

↳ C++ source code is transformed directly into something computer understands in 2 steps:

1) Compiler reads code and produces object files that are specif to the target machine such as 32-bit window, 64-bit linux

2) Linker stitches the object files together to produce a library, for you to use in another codebase or program to run directly



(This is the fastest process)

Using Tools:

→ C++ programs are composed of functions which group together statements into a block

↳ Functions can call other functions

↳ Function may return a value or not (if not use keyword null)

→ function that returns value must have its type specified.

↳ int: negative/positive numbers

→ program requires a main function that always returns an int for sucess or fail

```
int main() ❶
{ ❷
} ❷
```

❶ Function head

❷ Function body

→ Almost any C++ function has a return type, a function name and parentheses indicating any parameters or values given to the function

```
int main()
```

name

forms a function header

return type

no parameters

→ after function header set of curly braces with statements in them comprising the function body

→ The function main is special, returns int 0 to indicate no errors

↳ only one main function in the program

Instruction for Compiler

```
tool_name [optional flags] source_name.cpp -o output_name
```

```
clang++ -Wall -std=c++2b empty.cpp -o empty
```

for Mac

Running Program

→ use ./empty on mac.

Writing to the Screen:

→ The Println function is part of the Standard library

↳ use # include

```
#include <print>❶

int main()❷
{
}
```

→ use Println to print

```
#include <print>

int main()
{
    std::println("Hello, world!"); ❶
}
```

prints then goes to a new line

prints message

→ The print header also uses a different kind of "scope"
  ↳ groups code into a space called a namespace
    ↳ All standard library facilities live inside the standard namespace std
→ Prefix function needed with std:: to indicate where it comes from
  ↳ consists of the name std and the scope-resolution operator (::)
→ Can create your own namespaces, and specify where the compiler can find the function you want to use
→ Without std:: compiler/linker will look for a function called println outside the standard namespace and won't find it
  ↳ error unresolved symbol

using cout:

→ before C++ 23, std::cout printed instead of println
  ↳ in the iostream header

prints and stays on the line

```
#include <iostream> ❶

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

❶ Includes the `iostream` header

❷ Prints a message

```
#include <iostream>

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

moves to the next line

In depth:

```
int main()
{
}
```

return type
name
Parenthesis/input
name

→ function head followed by a semicolon declares the function
  ↳ tells compiler this function exists somewhere
→ Function definition is code in curly braces

```
void println(); ↘ overloads
void println(/* maybe some parameters*/);
```

→ overload function have the same name but take different parameters

↳ println(); has 0 parameters and moves to the next line

↳ println("..."); takes 1 parameter

→ Void means nothing is returned

→ Can use Cout with the Stream insertion operator (<<) to display text

```
std::cout << "Hello, world!" << '\n';
```

↳ You can chain calls together

↳ Compiler reads Left to right

↳ Left-most argument is the message so it is used first, then the << operator is applied a second time with the newline character

```
(std::cout << "Hello, world!") << '\n';
```

→ Std::endl can be used instead of `\n'

↳ 1) appends a new line

↳ 2) flushes the buffer

→ flushing the buffer ensures that everything in the buffer is written

↳ if Program crashes without this buffered characters may not make it to the Stream

→ Std::endl is a helper function to control a stream, called a manipulator, and using it is equal to: `std::cout << '\n' << std::flush;`