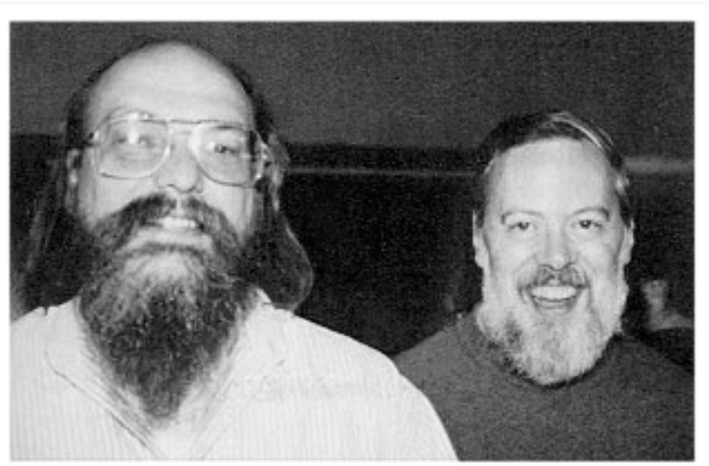
INTRODUCTION TO COMPUTATIONAL PHYSICS

Kai-Feng Chen
National Taiwan University

A LITTLE BIT OF THE HISTORY...

- The **C language** was developed at Bell lab by Dennis Ritchie in 1972, as a language for *building operating systems* for computers.
- The original goal was design a minimalistic language
  - Easy to compile;
  - **Allowed efficient access to memory.** ⇒ *as "middle-level" language*
  - Self-contained, do not need to rely on other programs;
  - As a high-level language and platform independent (good portability);
- Ken Thompson and Dennis Ritchie rewrote the **Unix operating system** using C in 1973 (Assembly was used for most of the OS developments at that time):
  - C's portability makes Unix becoming popular;
  - Unix's portability makes C popular, too!

Ken Thompson & Dennis Ritchie ⇒

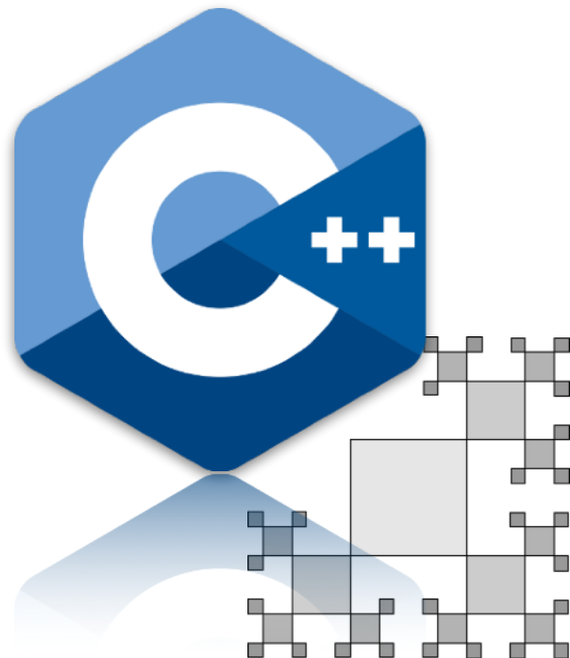


THE "STANDARD C"

- Computing languages also evolve!
- Brian Kernighan and Dennis Ritchie published a book called "The C Programming Language" in 1978.
  - ➡ The famous K&R standards, widely adopted in early compliers.
- ANSI C: American National Standards Institute establish a formal standard in 1989 as the C89/ANSI C standard; International Organization for Standardization (ISO) adopted ANSI C plus few modifications:
  - ➡ **C90 standards (most adopted version).**
- In 1999, the ANSI committee released the new C99 standards, although many many features had already included as extensions, or implemented in C++.

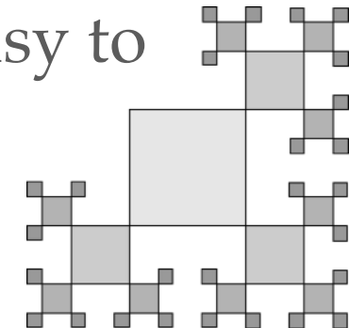
THEN IT'S THE TIME FOR C++

- Starting in 1979, C++ was developed by Bjarne Stroustrup at Bell Labs (*was as an extension to* C).
  - Many new features were introduced, as a **"superset of C"**. *(not exactly in fact!)*
  - **An object-oriented language.**
- ISO standardised C++ in 1998, with a minor update released in 2003.
  - As the C++03 standards.
- Recently there were major updates to the C++ language in 2011/2014/2017…and 2020:
  - Called as C++11, C++14, and C++17, and C++20.
  - **C++11 added a huge number of new capabilities, and is widely considered the new baseline.**
  - **Baseline of our course, too!**



PHILOSOPHY OF C/C++

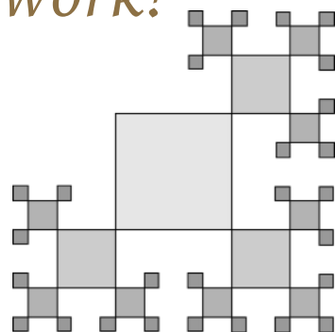
- The design philosophy of C/C++ can be summed up as "trust the programmer".
  - High degree of freedom to do whatever you want to do.
  - **Powerful but also dangerous — will not stop you from doing bad/ stupid coding.**
  - **Knowing what you shouldn't do is nearly as important as knowing what you should do!**
- Pro/con of C/C++:
  - **Pro:** when the performance is the key factor (OS development, productivity applications, scientific computing, entertainment/ gaming, etc…).
  - **Con:** difficult to learn for new comers, harder to turn around, easy to mess things up, etc…



My analogy:
C/C++ are sharp kitchen knifes, you definitely need them to become a great cook; but you may also cut your fingers if not trained…

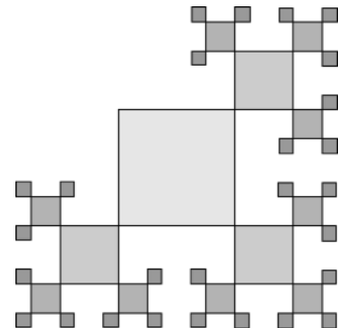
SOME PROGRAM DEVELOPING TIPS

- Preparations before real coding works are essential in fact:
  - **Define your problem first**
    - Coming up with the initial idea for what you would like to resolve. *e.g. a program to solve a set of linear equations.*
  - **Determine how to solve the problem**
    - There are generally multiple ways to solve the same problem, however the solution could be good or bad…
    - Good solutions are generally **straightforward**, **well documented**, **robust and portable**.
    - Immediately start coding usually results a fragile solution or even a *buggy* one! ⇒ *and debugging/maintenance is the REAL work!*
- Then it's the time to work out your program!



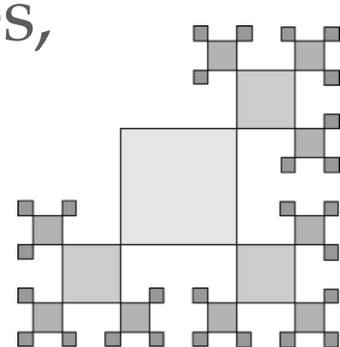
SOME PROGRAM DEVELOPING TIPS (II)

- Regarding writing the program — a good editor does help!
  - It is totally possible to develop your code with any text editors (e.g. notepad), but a good editor works better!
  - Good editors for programming should feature:
    Line numbering (*which line is which?*)
    Syntax highlighting (*sense a typo at the first look!*)
    Unambiguous monospace font (*not to mix 0 and O, etc.*).
    ……
- Look for an **IDE (Integrated development environment)**?
  - IDE usually integrates the *editor*, *compiler*, and *debugger*, and maybe even a *project manager*, it could be a good choice for beginners.
  - Not mandatory for experienced person in general — everyone can find the most comfortable way to develop!



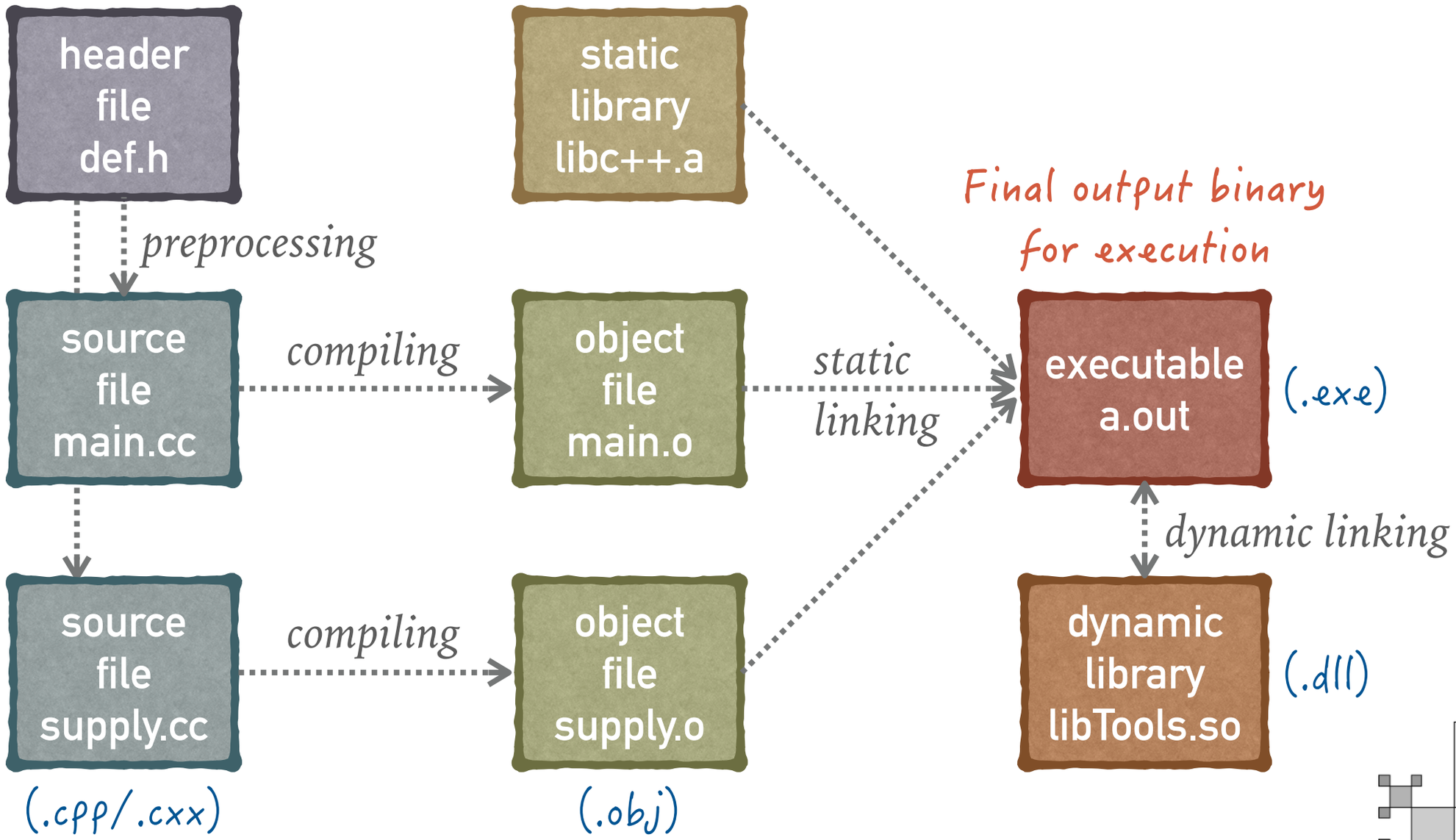
COMPILING & LINKING

- C/C++ program has to be *compiled* and *linked*, in order to produce the target **executable program**.
- The compiler process each source file (those .cc, .cpp, .cxx files) in your program:
  - The compiler first checks your code to ensure it follows the rules of the language.
  - Then the compiler translates your source file into a binary machine language file, named the **object file** (those .o or .obj).
- The linker then takes all the object files generated by the compiler, and combine them into a single executable program, together with the **library files** (those pre-compiled tools and packages, such as the *Standard Library*).



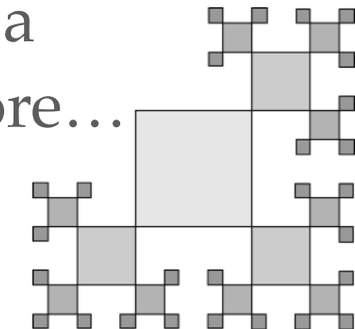
THE WORKFLOW

- Here are the full workflow in the construction of a C/C++ program, with a compiler and a linker. Note the file extensions are different for different OS or different compiler.



CHOICES OF COMPILERS

- Surely the possible choices of compilers depends on your system (your machine and OS).
- In this course we are using **command line GNU C/C++ compilers or Clang compilers** as the standard:
  - Those compilers generally come as a default options for Linux (usually GNU compilers are pre-installed, or can be activated easily) or Mac OSX (now the default compiler is Clang).
  - For windows users (*if you cannot switch to Linux*), it is possible to obtain an installation of GNU compilers; getting a copy of **Microsoft Visual Studio** can be also a good choice.
    - However it is better to get used to work on Unix/Linux systems for scientific computing.
    - If one day you need to "scale up" your work to run your job on a computing cluster of >1000 CPUs, no one uses Windows anymore…



FOR LINUX USERS

- Due to the large variation of Linux distributions, the compilers may not installed by default. In this case you can get them directly by the following commands in your **terminal** (*remove sudo if you are under root account*):

For Ubuntu/ Debian

```
$ sudo apt update
$ sudo apt install build-essential
```

For Redhat/ Centos

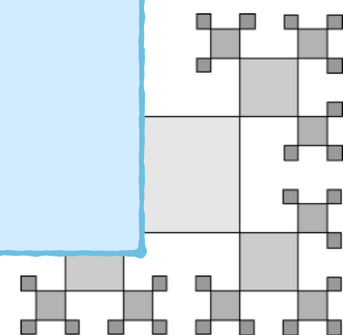
```
$ sudo yum update
$ sudo yum groupinstall "Development Tools”
```

For Fedora 23+

```
$ sudo dnf update
$ sudo dnf groupinstall "Development Tools”
```

Check the version (e.g. on Cento 7)

```
$ g++ -v
Using built-in specs.
COLLECT_GCC=g++
......
gcc version 4.8.5 20150623 (Red Hat 4.8.5-39) (GCC)
```



FOR MAC OSX USERS

- All you need to do is to install the **Xcode** (App Store ⇒ Search it ⇒ Install it).
- You may need to wait for a period of time, since Xcode is large (*as Xcode is a full IDE*).

- If you just want to have the Command Line Tools (those Clang compilers, etc, do not want to have the heavy full Xcode), just start a **terminal** and type-in the following command:

```
$ xcode-select --install
```

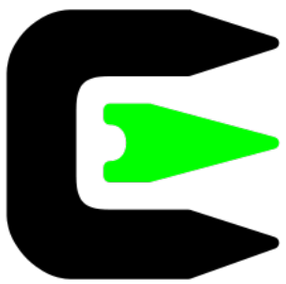
```
$ g++ -v
Configured with: --prefix=/Applications/Xcode.app/
Contents/Developer/usr
......
Apple clang version 11.0.0 (clang-1100.0.33.17)
```

You can check the version as well!

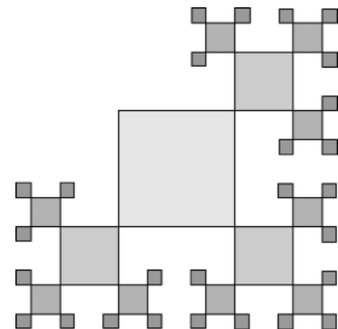
FOR WINDOWS USERS

- **STRONGLY RECOMMEND** to install a copy of Linux on your machine if possible. You can install it as a **dual boot** (systems are installed in different partitions; need reboot to switch the OS), or though a **virtual machine** software (e.g. VMWare, VirtualBox, etc, which simulate a PC under your OS), or through the **WSL** (Windows Subsystem for Linux, if you have the Win10 installed).
  - Note the dual boot has the highest performance by construction, and WSL is also a good choice nowadays.
- If not, you can still install a copy of GNU compilers though **MingGW-W64** (Minimalist GNU for Windows, http://mingw-w64.org), or **Cygwin** (as a full Unix-operation on top of Windows, https://www.cygwin.com).
- Or just install the Visual Studio (*good IDE for local coding in fact*).

MingGW-W64


Cygwin


MS Visual Studio



GET YOUR VERY FIRST PROGRAM TO RUN

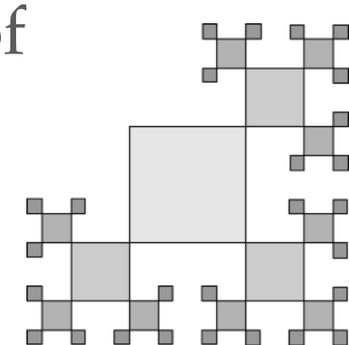
- Now let's get our very first program to work. Everyone should start with the "Hello, World!" program, right?

Just open an editor, type-in the code as given, save the contents into a typical source code file hello.cc:

```cpp
#include <iostream>

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

hello.cc

- Some quick comments regarding this program:
  - The first line, "#include <iostream>", is a preprocessor directive, which indicates that we would like to use the iostream library.
  - Every C/C++ program must have a special **"main"** function When the program is run, execution starts with the first statement inside of function main.



COMPILE/LINK/RUN YOUR FIRST PROGRAM

- Suppose you already get your compilers installed, let's build your program accordingly. Please start a terminal and type:

```
$ g++ hello.cc
```

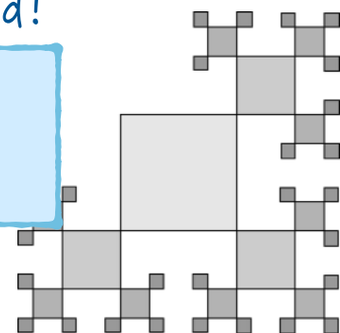
- If everything goes well, you should see no error message but a new file **a.out** has been generated:

```
$ ls -l a.out
-rwxr-xr-x  1 kfjack  staff  18908 Jul 11 12:05 a.out
```

- And this is actually the output executable file, and you can run it by type-in **./a.out** in your terminal (the "./" indicates your executable file is just "here", instead of regular path):

And get the "Hello, World!" printed!

```
$ ./a.out
Hello, World!
```



SOME TYPICAL GNU COMPILER OPTIONS

- I would like to output the executable to a different name:

the executable name will be "hello".

```
$ g++ -o hello hello.cc
```

- I would like to compile the source, without linking the executable:

this will generate hello.o instead of a.out.

```
$ g++ -c hello.cc
```

- I would like to link the object files to an executable:

merge hello. and tools.o into an executable.

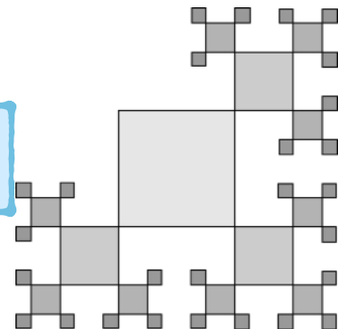
```
$ g++ -o hello hello.o tools.o
```

- Tell the compiler where to find the header files:

```
$ g++ hello.cc -I/usr/local/include
```

- Tell the compiler where to locate the library files:

```
$ g++ hello.cc -L/opt/root/lib -lCore
```



LANGUAGE STANDARD

- As already introduced, there are different standards of C++ available.
- The compiler usually picks a standard as default (often stay at ISO C++98 in fact…). If you wish to choose a different one, you'll have to configure your compiler to adopt the standard.
  - In this course we may use some **C++11** specific codes, if you do not config your compiler properly, the code cannot be compiled.
  - For GNU g++ this can be set by the **-std** flag, e.g.

```
$ g++ -std=c++11 hello11.cc
```

- You can try to compile the following "Hello World" and see if your compiler works properly with C++11 codes (*auto type & Lambda expression*):

hello11.cc

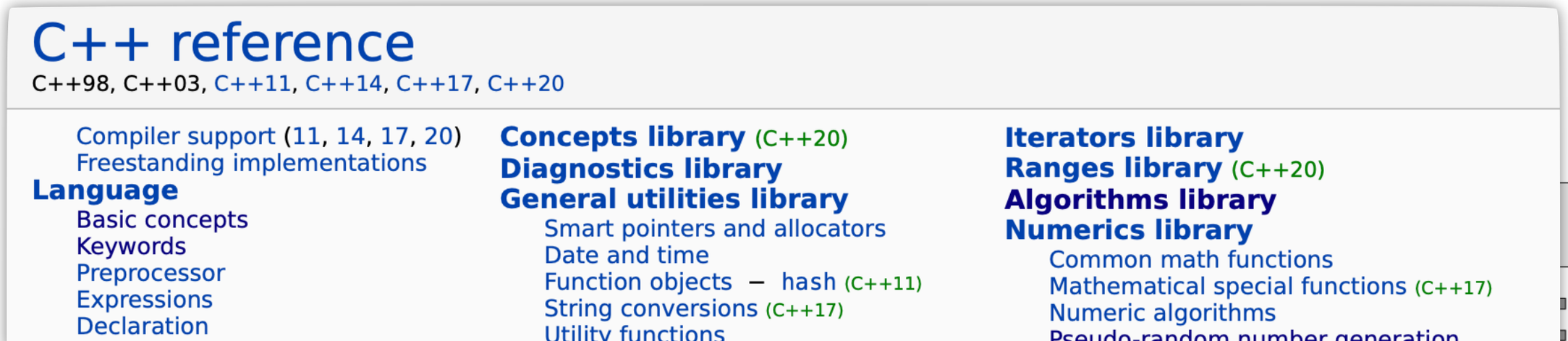
```
#include <iostream>

int main() {
  auto hello = []() { std::cout << "Hello, World!" << std::endl; };
  hello();
}
```

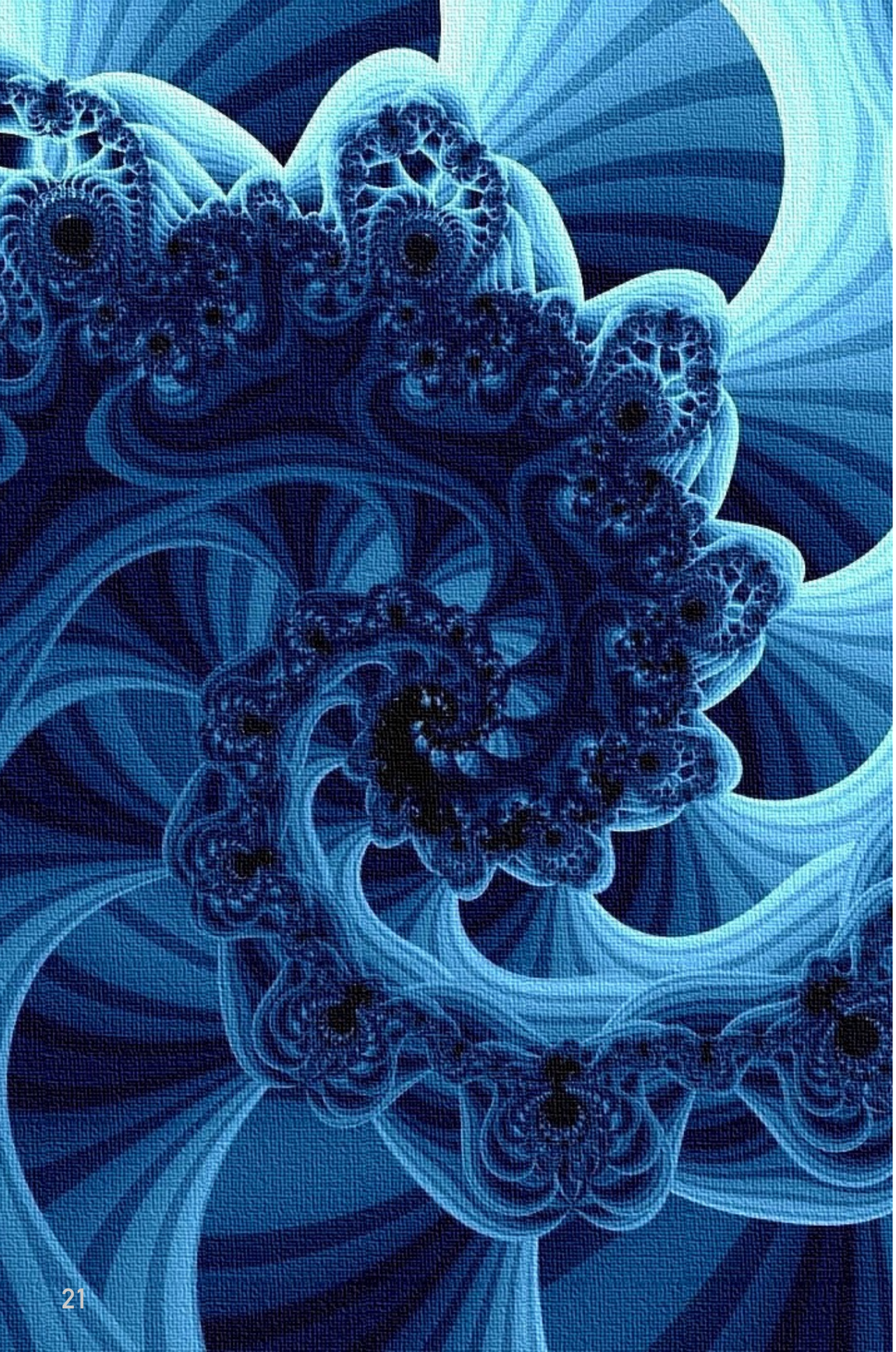
SUPPOSE EVERYTHING GOES WELL…

- Suppose everything goes well, you should have already complied the very first hello world program and execute it!
  - Note we assume you are using a terminal to execute the command line program, including the program you have just wrote.
  - If you are running your program by double-clicking the executable (or run it within an IDE), you may find your program *just pops up and ends quickly without waiting for you*.
  - In this case you can add a line like "std::cin.get();", right before the return statement, and it will wait for a keyboard input before real ending.
- There can be all kinds of "small problems" like this floating around. **Please ask for help**, or at least get them solved during our lecture hours!

PROPERLY USE THE REFERENCES

- Now it's the beginning of our journey though the C/C++ language, and it is impossible to cover all of the details in the slides. You will have to look for other references definitely (or a text book if you wish)!
- Note most of the slides are prepared based on the materials in https://www.learncpp.com, so you are recommended to check for details there!
- Also you may want to check the exact definitions (e.g. standard library classes), in this case https://en.cppreference.com/w/ is very helpful!
- Well, you may always find the information on https://stackoverflow.com, but it does require some knowledge to pick up the right answer!





MODULE SUMMARY

- In this module we have introduced the history of C/C++ programming language, discussed about the regular workflow for programming development, and build our very first program with GNU command-line compiler.
- Starting from the next module we will officially start to introduce the language itself.