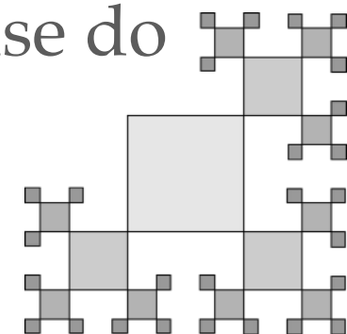
INTRODUCTION TO COMPUTATIONAL PHYSICS

Kai-Feng Chen
National Taiwan University

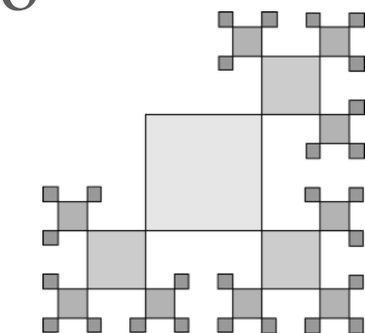
OVERVIEW

- Starting from this module, we start to look into several topics that are essential to every C/C++ program, *in a rather shallow manner*, as the main goal is to quickly construct a simple but workable program:
  - Statements, the structure of a C/C++ program
  - Variables, type and value
  - Practice with the iostream
  - C/C++ Keywords
  - Operators and Expressions
- Surely some of the topics will be revisited again in the upcoming modules to patch up missing points!
- Again our regular lecture hours will be used to discuss and please do not hesitate to bring your problems found to the class.



STATEMENTS

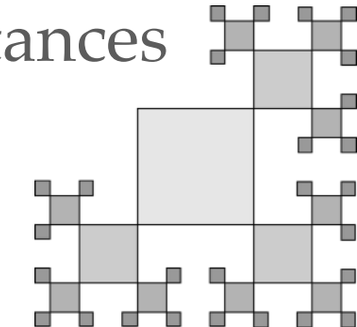
- **A computer program is a sequence of instructions** that tell the computer what to do; a statement is the instruction which asks the program to perform the actions.
- **Statements are the smallest independent unit of computation.**
  - ➡ Just like the natural language we speak daily, we typically express words in sentences (*not in random*).
  - ➡ In C/C++, when we want to ask our computer to do something, we write statements in the program.
- Most of the statements in C/C++ end with a **semicolon (;)**
- As C/C++ is a high-level language (closer to human language, less machine dependent), a single statement usually compiles into multiple instructions in machine language.



Missing semicolon is probably the most common error for C/C++ programming...

FUNCTIONS & THE MAIN FUNCTION

- In C/C++ statements are typically grouped into **functions**; a function is a collection of statements that executes in sequence.
  - Commonly written together with a pair of parenthesis to the end of the function's name;
  - Functions are generally designed to do a specific job. e.g.
    - A **"sin(x)"** function is calculating the sine value of x;
    - A **"printf(str)"** function is print the given string str on the screen;
- Every program by default has a special function **main()**, which is the entry point. Program execution starts from the first statement of function main and continues sequentially.
- Programs typically terminate or finish the tasks when the last statement inside function main is executed, although there are some circumstances where the program ends earlier.



DISSECTING HELLO WORLD

- Let's return to the "Hello World" program and take a look at what each line does in detail:

Preprocessor directive indicates to use the iostream library, which is the part of the C++ standard library that allows us to read/write text.

All of the programs are generally following this scheme, or a variation.

```
#include <iostream>

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

Definition of the main() function, entry point of the whole program.

Return statement, where the program finishes running, the program sends a value back to the operating system in order to indicate whether it ran successfully or not. (0 = everything is okay)

First statement within function main; std::cout and the << operator allow us to send string to the console.

SYNTAX & SYNTAX ERRORS

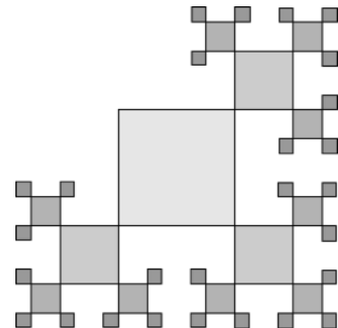
- **Syntax** is the rule about how your programs must be constructed in order to be considered valid. The compiler is responsible for ensuring your program follows the syntax of the C/C++ language. If you violate a rule, the compiler will complain and issue you a **syntax error**.
- For example if there is a typo in the code, e.g.

```
#include <iostream>

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

- The compiler tells you there is a syntax error on line 6; a semicolon after the return statement is expected.

```
$ g++ -std=c++11 hello.cc
hello.cc:6:13: error: expected ';' after return statement
    return 0
            ^
            ;
1 error generated.
```

COMMENTS IN CODE

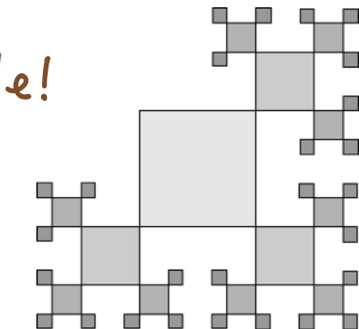
- A **comment** is a note that is inserted directly into the source code by the programmer. Comments help programmers to document the code, and are ignored by the compilers.
- Single-line comments: the compiler will ignore everything starting from the // symbol to the end of the line, e.g.

```
// Uses Newton's method to approximate the root of the equation.
double finding_root(double initial_value, double eps=1E-13)
```

- Multi-line comments: the /* and */ pair of symbols denotes a C-style multi-line comment. Everything in between the symbols is ignored. e.g.

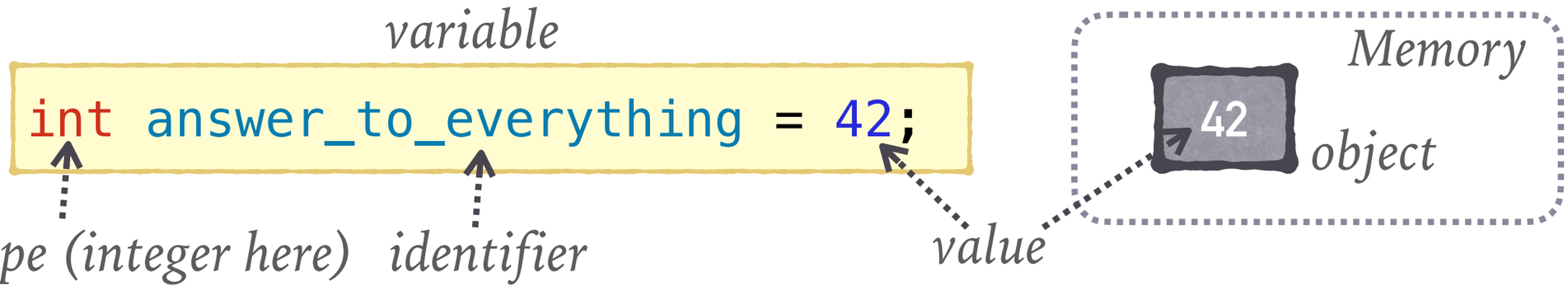
```
/* This function uses Newton's method to approximate the root.
   "initial_value" is the starting point
   "eps" is the target precision */
double finding_root(double initial_value, double eps=1E-13)
```

Write your comments as speaking to someone who has zero idea about the code!
Don't assume you'll remember every details afterwards!!



DATA & VARIABLES

- Programs are instructions that manipulate **data** to produce desired results, while the data on a computer is stored in a format, or **type**, that is efficient for storage or processing.
- A single piece of data, stored in computer memory somewhere, is called a **value**.
- The memory can be indirectly accessed with an **object**, which is placed in a region of storage associating with a value and other properties. When an object is defined, the compiler determines where the object will be placed in memory.
- Objects can be named or anonymous — a named object is called a **variable**, and the name of the object is called an **identifier**.



VARIABLE INSTANTIATION & DATA TYPES

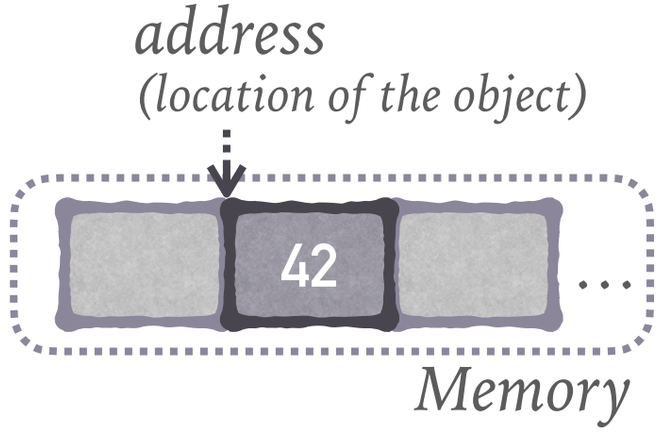
- A variable can be created with a **declaration statement**, e.g.

```
int answer_to_everything = 42;
```

- When compiling, the compiler takes this statement and makes a note to itself that we are defining a variable. The variable is of type **int** (integer), with the name **answer_to_everything**, and with the value of **42**.

- At runtime, the variable will be **instantiated** as the object will be created and assigned a memory address. Variables must be instantiated before they can be used to store values.

- An instantiated object is sometimes also called an **instance**.



- The type of a variable must be known at compiling, and that type can not be changed without recompiling the program, although converting the value to a different type is possible. Just few more practices:

```
int a, b, c;
unsigned int mask = 0xffef; ← positive-only integer
double pi = 3.1415927; ← a double-precision float point number
```

VARIABLE ASSIGNMENT & INITIALIZATION

- After a variable is defined, and one give it a value with the **assignment operator =**. This operation is called **(copy) assignment**.

```
double hbar;
hbar = 1.0545718E-34; ← E-34 means 10^-34
```

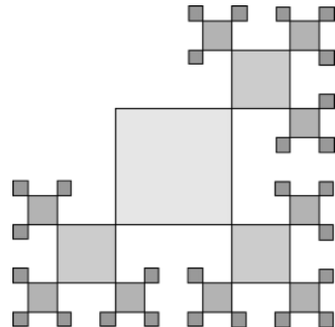
- As already shown before, the definition and value assignment can be combined into one operation, this is called **initialization**.
  - Classically this is called the **copy initialization**, by using an assignment operator:

```
double hbar = 1.0545718E-34;
```

  - One can do **direct initialization** by using parenthesis:

```
double hbar(1.0545718E-34);
```

Direct initialization is recommended *before* **C++11** in most cases because of the performance boost.



BRACE INITIALIZATION

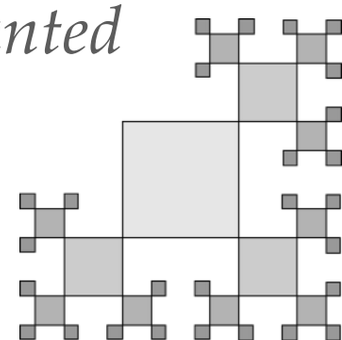
- In fact the direct initialization cannot be introduced for some cases (e.g. an object with a list of data does not work). **C++11** added a new syntax called **brace initialization**, e.g.:

```
int A{6};      // Direct brace initialization, preferred!
int B = {7};   // Copy brace initialization
int C{};       // Zero initialization, ie. initializes the variable to 0
```

- Brace initialization also disallows *"narrow"* conversions. ie. if you try to initialize a variable with a value which can not be safely held, the compiler will throw in an error. You may find your compiler complaint with an error about the second initialization, but only give a warning to the first one:

```
int A = 2.71828; // Warning, conversion between float-point to int (A=2)
int B{2.71828};  // Error
```

- Initialize your variables upon creation is a good practice (*avoid unwanted behavior*). Use brace initialization when possible is also good, too!



BASIC INPUT/OUTPUT WITH IOSTREAM

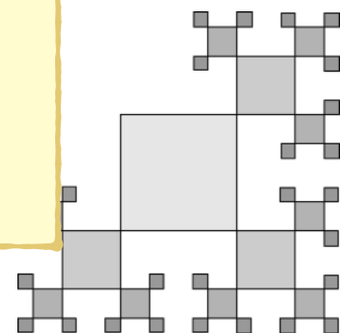
- Come back to the input/output library, like the iostream we have already used in the hello world example.
- To use the iostream library, the iostream header should be included at the beginning of the code, ie.

```
#include <iostream>
```

- **std::cout** — "character output" send data to the console to be printed:
  - In the hello world example we use std::cout, along with the **insertion operator <<**, and send the text "Hello world!" to the console.
  - Why there is a "std::" — this is due to the cout are belonging to the **namespace std**. The following code is also the way to use it:

```
using namespace std;

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

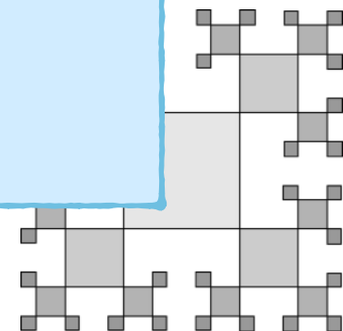



STD::COUT AND STD::ENDL

- std::cout can not only print text, it can also print numbers or variables; the insertion operator << can be used multiple times in a single statement to concatenate multiple print outs.
- By including **std::endl** your program prints a special newline character **\n** to the console plus a "flush" (makes sure that it shows up on the screen immediately, as did in **std::flush**).

```
double pi = 3.1415927;
std::cout << "The value of pi = " << pi << std::endl; ← endl = new line + flush
std::cout << "A circle of radius " << 2.;
std::cout << " has an area of " << pi*2.*2. << "\n"; ← new line only
std::cout << "A ball of the same radius has the volume of ";
std::cout << 4.*pi/3.*8. << "\n" << std::flush; ← new line + flush
```

```
The value of pi = 3.14159
A circle of radius 2 has an area of 12.5664
A ball of the same radius has the volume of 33.5103
```



HOW ABOUT THE INPUT — STD::CIN

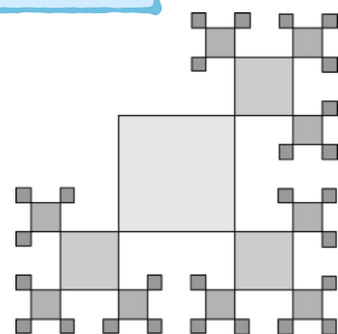
- **std::cin**, which is another predefined variable in the iostream library, stands for "character input", reads input from keyboard using the **extraction operator >>**. The input must be stored in a variable.

```
#include <iostream>

int main()
{
    int x(0);  ← define the variable x and zero-initialize it with zero
    std::cout << "Enter an integer please: ";
    std::cin >> x;    ← get number from keyboard, keep in x
    std::cout << "The value you have entered: " << x << "\n";
    return 0;
}
```

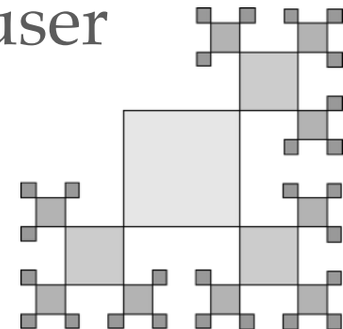
```
Enter an integer please: 42 ← if you key in 42
The value you have entered: 42
```

Note we don't need to use '\n' when accepting input from cin, as the user will need to press the enter key…



IOSTREAM SUMMARY

- Here are an easy way for you to remember what is what:
  - **std::cin** and **std::cout** always placed at the left-hand side of the statement;
  - **std::cout** is used to output a value (*cout = character output*);
  - **std::cin** is used to get an input value (*cin = character input*);
  - **<<** is used with **std::cout**, and shows the direction that data is moving;
  - **>>** is used with **std::cin**, and shows the direction that data is moving.
- Note the iostream library does not provide a way to detect the keyboard states direct (e.g. for video game inputs).
  - You'll have to use a different library to do so; many graphical user libraries have their own functions to do this task.



AN OLD, BUT STILL NICE WAY

- The iostream is the standard console I/O library for C++. For pure C, one can use the famous **printf** for output and **scanf** for input.
- Following is the declaration for **printf()** function and an example:

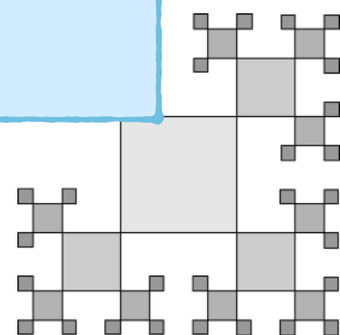
```
int printf(const char *format, ...)
```

```
#include <cstdio>   ← include stdio header file

int main()
{
    double pi = 3.1415927;          ↓ %f stands for "float-point" format
    printf("The value of pi = %f\n",pi);
    printf("Answer to everything is %d\n",42);
    return 0;                                ↑ %d stands for "signed integer" format
}
```

```
The value of pi = 3.141593
Answer to everything is 42
```

You can see the "%f" and "%d" have been replaced by the value of π and 42.



AN OLD, BUT STILL NICE WAY (II)

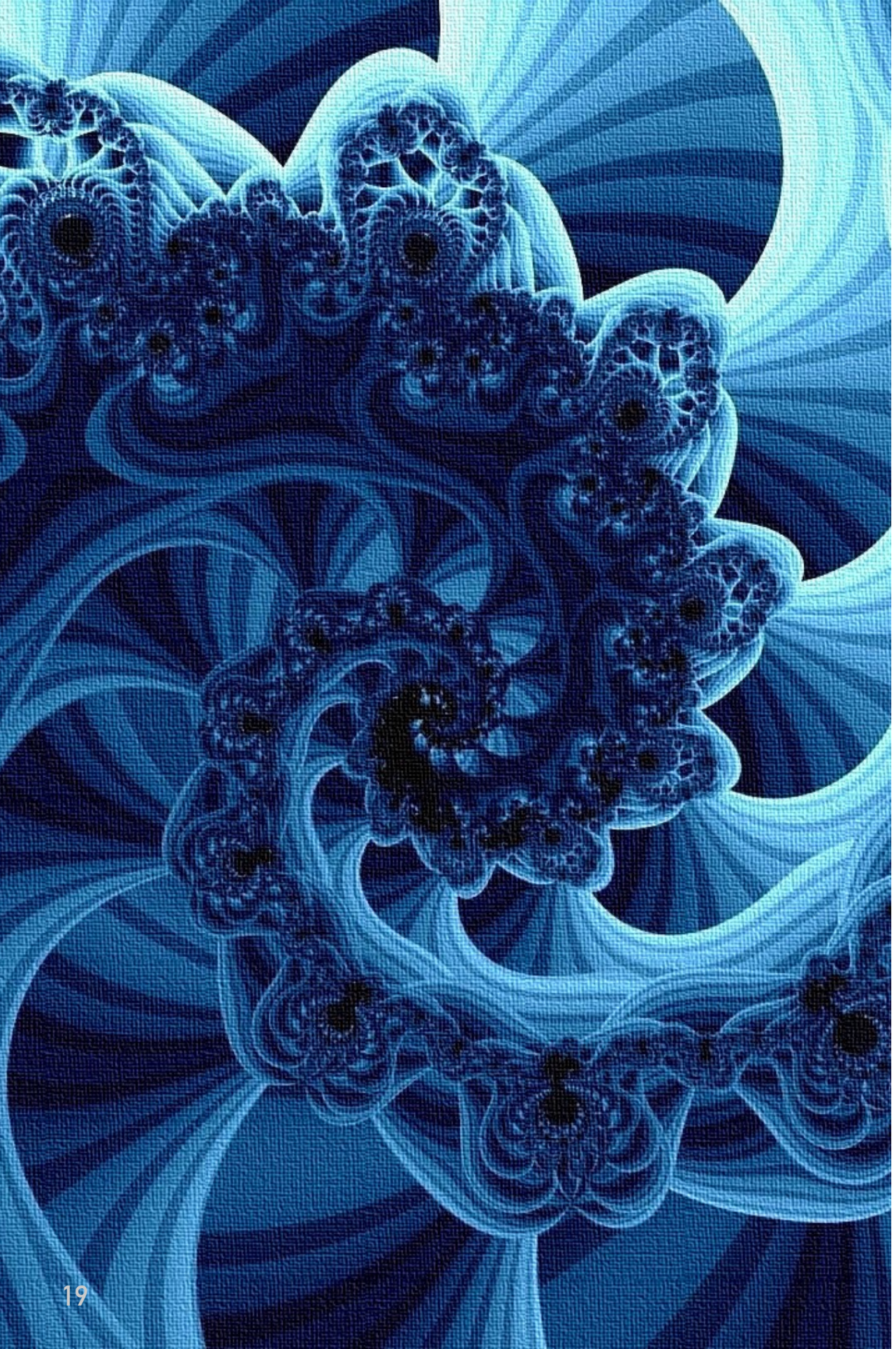
- For input, **scanf()** function can be introduced:

```
int scanf(const char *format, ...)
```

```
#include <cstdio>
int main()
{
    int x(0);
    printf("Enter an integer please: ");
    scanf("%d",&x); ← store the input integer to x;
                      the &x stands for the pointer of x (memory address of x)
    printf("The value you have entered: %d\n",x);
    return 0;
}
```

```
Enter an integer please: 42
The value you have entered: 42
```

Using std::cin/std::cout or printf()/scanf() can have different advantages; although the formatting of printf() can be a little bit cryptic for beginners…
We will revisit the formatting in a different module!



MODULE SUMMARY

- In this module we have introduced some of the basis of C/C++ language, including the basic statements, the structure of program (and the main function), variables, and a very brief touch of the iostream library.
- The next module we will revisit the variables definition, and restrictions to the variable naming (including keywords, etc), as well as C/C++ operators, and others.