Phase Field Tutorial
C++ Programming and Calculation Examples

Zihang Wang

Central South University

November 21, 2024

Quick Review

What have we got in the last tutorial?

- Python Installation
- Programming Basics (Python)
- Forward / Backward Euler Method
- OOP and Numerical Integral
- OOP and Gradient & Laplacian

While, with these numerical methods implemented with Python, Let's try to combine them together, with *C++*.

C++, a 'better C'

C++, a programming language designed by Bjarne Stroustrup in 1979, was originally designed to be an improved C, has become one of the most popular programming languages in the world. It supports OOP, template programming, and many otehr useful features. It's so efficient that can be compared with C. All these features enabling it to be an ideal choice for scientific computation.

C++, a 'better C'

C++, a programming language designed by Bjarne Stroustrup in 1979, was originally designed to be an improved C, has become one of the most popular programming languages in the world. It supports OOP, template programming, and many otehr useful features. It's so efficient that can be compared with C. All these features enabling it to be an ideal choice for scientific computation.

However, it is also a complicated language There are messive details of grammar, and some features make it not so friendly to programmers. Some common things in other language could be not that easy in C++.

C++, a 'better C'

C++, a programming language designed by Bjarne Stroustrup in 1979, was originally designed to be an improved C, has become one of the most popular programming languages in the world. It supports OOP, template programming, and many otehr useful features. It's so efficient that can be compared with C. All these features enabling it to be an ideal choice for scientific computation.

However, it is also a complicated language There are messive details of grammar, and some features make it not so friendly to programmers. Some common things in other language could be not that easy in C++.

But, fortunately, to learn and do some calculation with C++, all you need is almost there.

What’s our plan?

So, where shall we start from? What shall we do? Repeat the algorithm in the last tutorial, and just implement them in C++?

What's our plan?

So, where shall we start from? What shall we do? Repeat the algorithm in the last tutorial, and just implement them in C++?

That's half-right. We won't cover all details in implementing these algorithm. What we are going to do are:

1. C++ Environment Setup;
2. C++ Grammar and Features;
3. Algorithms, again;
4. Simple Example: 1D Heat Transfer Question.

What's our plan?

So, where shall we start from? What shall we do? Repeat the algorithm in the last tutorial, and just implement them in C++?

That's half-right. We won't cover all details in implementing these algorithm. What we are going to do are:

1. C++ Environment Setup;
2. C++ Grammar and Features;
3. Algorithms, again;
4. Simple Example: 1D Heat Transfer Question.

That's a lot. Let's start our very first step: environment setup.

Make a Choice

Okay, you must can't wait to download something. But there are bunch of choices, and some of you might end up with wrong configuration, frustrated and stay away from C++ to retain your mental health. So to save your time, I recommend three way to get started with C++:

Make a Choice

Okay, you must can't wait to download something. But there are bunch of choices, and some of you might end up with wrong configuration, frustrated and stay away from C++ to retain your mental health. So to save your time, I recommend three way to get started with C++:

- The newest Visual Studio Community Edition if you are runing on Windows.
- The GCC compiler tool chain chain with Linux or WSL if you are on Linux or want to give it a try.
- The Clang/LLVM compiler tool chain if you are on Mac OS or you just don't like the former two.

Make a Choice

Okay, you must can't wait to download something. But there are bunch of choices, and some of you might end up with wrong configuration, frustrated and stay away from C++ to retain your mental health. So to save your time, I recommend three way to get started with C++:

- The newest Visual Studio Community Edition if you are runing on Windows.
- The GCC compiler tool chain chain with Linux or WSL if you are on Linux or want to give it a try.
- The Clang/LLVM compiler tool chain if you are on Mac OS or you just don't like the former two.

Let's see what are they and their features.

VS: The Best IDE for Windows C++ Dev (Maybe[1])

Visual Studio (abbr. as *VS*), an IDE from Microsoft for developing C++ or .Net on Windows, usually is the best choice for any C++ programmer using Windows. As an IDE, VS is fully equipped with editor, linter, compiler, debugger and analyzer. After some *right* clicks, you are then ready for C++ progrmming, and just skip the tiring prerequisite. You can focus yourself on programming and solving problems.

[1]There are still many other choice, and everyone has their own choice, too. For me myself, VS is good.

VS: The Best IDE for Windows C++ Dev (Maybe[1])

Visual Studio (abbr. as *VS*), an IDE from Microsoft for developing C++ or .Net on Windows, usually is the best choice for any C++ programmer using Windows. As an IDE, VS is fully equipped with editor, linter, compiler, debugger and analyzer. After some *right* clicks, you are then ready for C++ progrmming, and just skip the tiring prerequisite. You can focus yourself on programming and solving problems.

However, there are flaws in VS. VS is a *BIG* software needing a lot of drive space to install it. And, some components must be installed under your `C` drive. But if that's okay for you, VS should be the first choice.

[1]There are still many other choice, and everyone has their own choice, too. For me myself, VS is good.

GCC: Old, but powerful.

Well, if you are not a Windows user, or want to compile C++ for other platform, or just don't want to use VS and MSVC (Compiler used by VS), Then you can try GCC, the GNU Compiler Collection.

GCC: Old, but powerful.

Well, if you are not a Windows user, or want to compile C++ for other platform, or just don't want to use VS and MSVC (Compiler used by VS), Then you can try GCC, the GNU Compiler Collection.

What is GNU? GNU is Not Unix! That will be a long story to tell about GNU, Linux, GCC and open source software. But if you are going to write your code on Linux, your first choice will always be GCC tool chain, explicitly, `g++`, `gdb` and `makefile`, which are compiler, debugger and build system, respectively. All of these tools have long histories, but are powerful and never outdated.

GCC: Old, but powerful.

Well, if you are not a Windows user, or want to compile C++ for other platform, or just don't want to use VS and MSVC (Compiler used by VS), Then you can try GCC, the GNU Compiler Collection.

What is GNU? GNU is Not Unix! That will be a long story to tell about GNU, Linux, GCC and open source software. But if you are going to write your code on Linux, your first choice will always be GCC tool chain, explicitly, `g++`, `gdb` and `makefile`, which are compiler, debugger and build system, respectively. All of these tools have long histories, but are powerful and never outdated.

If you are interested in GCC and Linux, but already a Windows user, I recommend you to install 'WSL' on your Windows. That will give you almost the original experience of a Linux system.

Clang/LLVM: Progressive New Force

If you are on Mac OS, after install gcc and check your installation, you will be surprised by prompt from ‘Clang’. That’s because Mac OS use Clang/LLVM as its default C/C++ compiler.

Clang/LLVM: Progressive New Force

If you are on Mac OS, after install gcc and check your installation, you will be surprised by prompt from 'Clang'. That's because Mac OS use Clang/LLVM as its default C/C++ compiler.

Clang is a part of LLVM Project, which is originally a research project at University of Illinois. Clang is a compiler that compatible with many backends, such as MSVC and GCC. And, as a fully modularized compiler, it has a great potential to be extended and developed, and is a good example for whom studying compiler theory and language design.

Clang/LLVM: Progressive New Force

If you are on Mac OS, after install gcc and check your installation, you will be surprised by prompt from ‘Clang’. That’s because Mac OS use Clang/LLVM as its default C/C++ compiler.

Clang is a part of LLVM Project, which is originally a research project at University of Illinois. Clang is a compiler that compatible with many backends, such as MSVC and GCC. And, as a fully modularized compiler, it has a great potential to be extended and developed, and is a good example for whom studying compiler theory and language design.

Well, we don’t actually need to learn compiler or language design. What’s good for choosing Clang? If you are using Mac OS, that will be your first choice. Or you can just give it a try. Clang is famous for its readable and helpful error/warn messages. Maybe you will like it.

Let's do it

By now you should have made your choice. Let's start to set our enviornment up.

Let's do it

By now you should have made your choice. Let's start to set our enviornment up.

If you choose to install Visual Studio from Microsoft, please download the installer (yes, you need install the installer first), and *select 'Desktop development with C++'*. Then you can just accept the default settings.

Let's do it

By now you should have made your choice. Let's start to set our enviornment up.

If you choose to install Visual Studio from Microsoft, please download the installer (yes, you need install the installer first), and *select 'Desktop development with C++'*. Then you can just accept the default settings.

If you choose to install WSL and GNU, I recommend reading WSL guide from Microsoft before install it, and installing VS Code. VS Code has good support to WSL and C++ programming with extension. Then after WSL setup, you can install GNU tool chain by `sudo apt install build-essential`.

Let's do it

By now you should have made your choice. Let's start to set our enviornment up.

If you choose to install Visual Studio from Microsoft, please download the installer (yes, you need install the installer first), and *select 'Desktop development with C++'*. Then you can just accept the default settings.

If you choose to install WSL and GNU, I recommend reading WSL guide from Microsoft before install it, and installing VS Code. VS Code has good support to WSL and C++ programming with extension. Then after WSL setup, you can install GNU tool chain by `sudo apt install build-essential`.

If you are using Mac OS, you can use `homebrew` to install Clang, and choose your favourite editor or IDE.

Your First C++ Code

So, let's try to write a little piece of C++ code.
Your first C++ code, as usual, will be a simple 'Hello World':

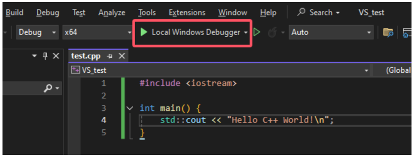
```
#include <iostream>

int main(){
        std::cout<< "Hello C++ World!\n";
    }
```

This code includes many points. You use `#include` to *Copy & Paste* file `iostream`, and your program will start executation from the function `main()`. Finally your code will use `std::cout` to output a string `"Hello C++ World"` to your screen, and print a line break using escape character `'\n'`. Details will be in the next section.

Run it

Now, if you are using Visual Studio, you can just click the button: 'Local Windows Debugger':



```
#include <iostream>

int main() {
    std::cout << "Hello C++ World!\n";
}
```

if you are using Linux or Mac OS, please type `g++ *.cpp` under the folder your source file locates. You will get a file called `a.out`. That will be your program and you can run it by typing the followings in your console: `./a.out`.

Or, if you are using editor with LSP support, like VSCode, you can just use the extension to run your code.

standard libraries, `#include` & `main`

To start with, first we encounter in C++ program is usually several lines looks like `#include <xxx>`. Here 'xxx' is usually one of the *standard libraries*, which are a set of *headers* together with *libraries* that you can use when you want certain functions. For example, if you want to input & output something, you use `iostream`; if you want to store something inside a `vector`, you use `vector` header.

Each C++ executable file must contains a `main` function in its source code. That's where our programs will start to execute from. This might be different from Python (although Python also supports such features, it's not default behaviour).

types and std

C++ is a type strict programming language. That's very different from Python: Python is dynamical type language, and value in Python doesn't check the type. While, C++ have statical types, value will have its type by your design, and C++ will check your code about the types before run.

There are several types you already familar: 'int', 'float', 'bool'(although in C++ both value should be lower case), 'char' and so on. You may ask: 'Where is 'string'?' In C++, we recommend 'std' string (although it also supports C-style raw string), which is actually a 'class' that you can declare and initialize a 'std' string by using `std::string str = "xxx"`.

There are also many other 'std' things, where 'std' is a 'namespace' that used by the standard libraries to avoid name conflict.

for loop

In C++, flow control such as loop and conditional statements are similar with Python. Here we demonstrate the `for` loop.

`for` loop in C++ start from the key word `for`, then a parenthesis including three *sentences*: init-declaration, loop condition and an expression that usually increments the loop counter. Then after that, you open a braces and the loop body, for example:

```
    for(int i = 0; i < 3; i++){
            std::cout << i << " ";
        }
```

This `for` loop will print `0 1 2`.

`if-else` statements

Here we introduce the most general conditional statements: `if else` statements. `if else` statement in C++ is very similar to that of Python, with only some difference in details:

- Use parenthesis in conditions instead of write out after `if`;
- Use braces to open the body statements instead of open with colon;
- Use `else if` explicitly instead of short-hand form `elif` in Python

We don't cover the ternary operator `?:` and `switch-case` statements. Feel free to explore them by yourself.

pointers

We won't cover too much about the pointer here, but give it a brief introduction. A pointer is a special variable that will hold other variable's address (called 'points to'). Then you can access the value that this pointer pointing to by dereference the pointer using operator `*`.

For example, if you have a variable `int i = 1` and a pointer `int *p = &i`, then the pointer `p` will have the value of the address of `i`. You can then use `*p` to get the value of `i` and change it. As pointer gets the address of that value, the modification on the `*p` will also be the modification of `i`.

refernce

However, pointer is too low-level that you might find it dangerous and hard to use. In C++, there is another way to do the similar thing but with more friendly grammar: *reference*. For example, to create a reference of `i` declared before, you use `int &ref = i`, then `ref` is the reference of `i`. You can consider reference is just a nickname, such that you can use this nickname just like the original name.

The reference and pointer will appear in the *function* part and play a key role.

function

Just as Python do, C++ also have *function*, but with slightly different grammar. But here is a good example: the `main` function. You already have defined the `main` function for many times, and just like that, you can define another function with the following key pointes:

1. return value type: indicate the what type of the value the function will return
2. function name: of course the function should have a name (although there is anonymous function called *lambda expression*)
3. function parameter list: Define what parameter type will this function take. You may also set the parameter's tempory name here to use them inside of your function body.
4. function body: Things that this function will do.

function: simple example

And a typical example will be the `add` function:

```
        double add(double a, double b){
                double c = a+b;
                return c;
            }
```

Here, the first `double` indicates that this function will return a `double` value; `add` is this function's name; `double a` and `double b` are the function's parameter, indicates this function accpets two `double` variable as its parameters.

Then inside the function body, we initialized a new `double` value `c` with value `a+b`, and finally we `return` the value of `c`. Notice that we can't access the variable `c` outside of this function as it's a *local* variable only lives inside of this function.

function parameters and reference

Can function change the value of the parameter variable? The answer is: It depends. If you use plain variable as function parameters, the function will *copy* the value of the parameter to the the tempory variables and use these tempory varaibles inside of the function. These tempory variables are local and hence can't be accessed from outside.

While, if you use reference when you define the parameter list, then when you call the function, the function will not copy the parameter's value, but just use the variables themselves. By this way you can modify the parameters' value inside of the function.

Pointer, originally from C, can also achieve the similar thing. While, as C++ has *reference*, it's much more recommended to use reference instead of pointer.

Implement algorithms with functions

With the language points before, now we shall begin to try implementing the algorithms that we have implemented in Python. That should not be too hard, as we already implemented them once, and this time we won't use OOP paradigm. These algorithms are implemented in form of funtions, such that you can call them directly and easily.

However, there is still a piece of code that use OOP in C++ to handle the calculation of gradients and laplacians. You can check it out if you like.

Introduction

Now we are going to carry on a simple simulation: heat transfer problem. Although it's not a phase-field simulation, this simulation can still be a good example to demonstrate the basic simulation procedure, which differs from phase-field simulation basically only the underlying theories.

There is the question: the one-dimensional heat conduction equation without heat source can be expressed as:

$$\rho c_p \frac{\partial T}{\partial t} = \frac{\partial}{\partial x}\left(\lambda \frac{\partial T}{\partial x}\right),$$

where ρ is density, c_p is heat capacity, λ is thermal conductivity, T is the temperature, t and x are time and position, respectively.

Question simplification

Here we consider a simplified form, such that the λ is a constant:

$$\frac{\partial T}{\partial t} = \mu \frac{\partial^2 T}{\partial x^2},$$

where $\mu = \frac{\lambda}{\rho c_p}$. By now we can utilize the tool we have acquired before, such as forward Euler method, finite difference algorithm and so on.

For this simulation, we shall use the following parameters:

parameter	value
μ	1.0
Δt	0.2
Δx	1

And we are going to use fixed boundary condition to handle this question.

Code structure

- Headers: need `vector` holding values, `string` to process strings, `fstream` to output files, and some other headers to increase usability,
- Constants: set the simulation related invariable to constants.
- Preprocess: set file output folder, set time counter and so on.
- Grid creation and mesh initialization: Assign values to a one-dimension vector according to the position.
- Time loop: loop as the time evolution, using forward Euler method.
- Mesh loop: iterate each points to calculate their values based on the last time step's result.
- File output: Record the result into the files.
- Postprocess: Calculate the used time and possible summarization of the program execution.

Code details

Here are some details that should be noticed:

- There will be two loops: one for time and another for space; you need to iterate time first and then iterate the space in each time step.
- As we are using fixed boundary condition, you have to assign each boundary the fixed value every time step.
- The result for the next step should be stored inside a tempory vector, such that the next step's result won't affect this step's calculation.
- To obtain the result of 0 step and 600 step, for example, you should set the time loop condition to `istep < 600+1` to make sure that `istep` can take 600.
- When operate the file io, one should make sure that the file is actually existed / opened.

Results process

After running this code, you are likely to get a set of files. To visualize these results, you have to process them into images. Unfortunately, C++ doesn't have such library that can offer an easy way to plot data. But as the data are stored inside files, you can plot them in many ways. Here we introduce two methods: *Paraview* and *Python*.

To use Paraview, you just install it from its website, then open it and from this software, and open your results in *line chart view*, then you can see your results and play the animation.

To use Python, you need load your data through `pandas` or `numpy` , take the two columns into arrays, and then plot them just like what you have done before.

Results

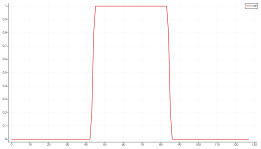


Figure: step 0

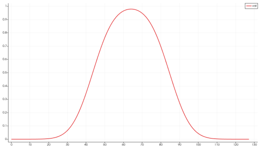


Figure: step 200

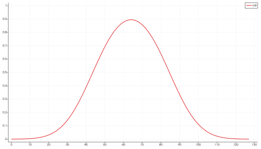


Figure: step 400

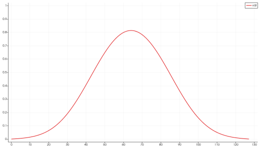


Figure: step 600

Summary

In this tutorial, we have set up C++ developing environment, learned basic C++ language, implemented the algorithms we have done with Python, and finally run a simple simulation about heat transfer procedure.

Please consider doing the exercises listed as follows:

1. Package the calculation step into a function;
2. Package the file output step into a function;
3. Use other boundary conditions (for example, periodic);
4. Modify the initial value distribution;
5. Change the heat transfer model.

There are also some optional exercises. Please do them as well if you like.

Resources

Here are some resources that might help you.

- Runoob C++ : A good C++ programming language tutorial website.
- *Programming Phase-Field Modeling*: The book we are going to reference a lot. The heat transfer question is from Chapter 4.2.
- *C++ Primer*: A dictionary-like C++ textbook that covers a lot of details about C++ programming language.
- WSL Installation guide from Microsoft. You can install the WSL to start progrmming with C++ on Linux.
- MinGW-w64 download page : A project to support gcc run on Windows. We use WinLibs to support compile and debug using gcc.

Thanks!

Any questions are welcomed!