中南大学FYT机器人战队

FYT视觉组第三次培训

Be committed to achieving success Be realistic and innovative

主讲人：智能2105 邹承甫

目录/CONTENT

第一部分

Hello World

第二部分

C++基础

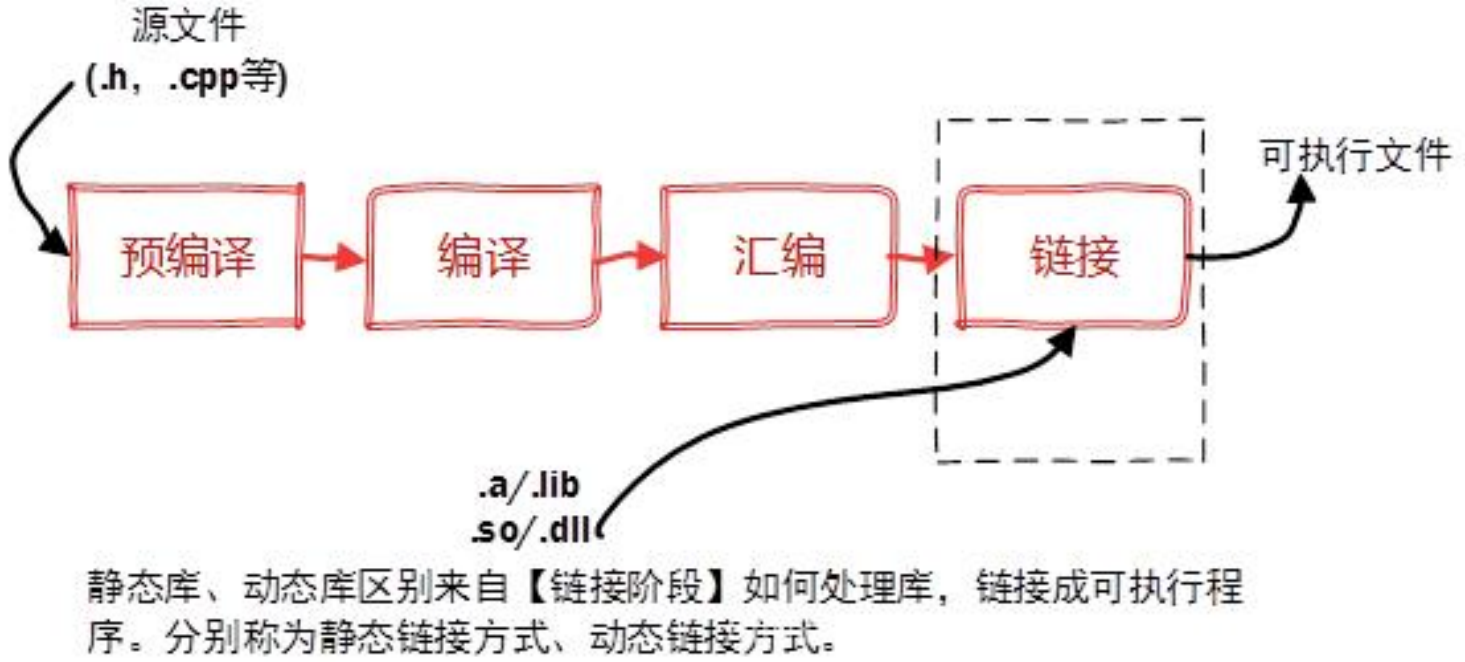
Hello World

Hello World

在main.cpp输入以下内容

```
#include <iostream>

int main() {
  std::cout<<”hello world!\n”;
  return 0;
}
```



静态库、动态库区别来自【链接阶段】如何处理库，链接成可执行程序。分别称为静态链接方式、动态链接方式。

使用g++编译代码

```
g++ main.cpp -S main.s  #预编译+编译，生成汇编代码
g++ main.s -c main.o #汇编，将汇编代码翻译为二进制指令
g++ main.o -o main #连接，将外部的库连接到代码中，这样就可以在运行中调用外部的函数
./main #运行，输出hello world!
# 可以直接使用g++ main.cpp -o main同时执行四个操作
```

多个文件编译

genshin.hpp（放在include下）

```
#include <iostream>

void genshin();
```

genshin.cpp

```
#include “genshin.hpp”

void genshin() {
  std::cout<<”原神，启动!\n”；
}
```

main.cpp

```
#include “genshin.hpp”

int main() {
  genshin();
}
```

编译

```
g++ main.cpp genshin.cpp -I ../include -o main
```

启动

```
./main
```

使用第三方库

安装fmt库

```
sudo apt install libfmt-dev
```

使用fmt库

```
#include “genshin.hpp”
#include <fmt/core.h>
#include <string>

int main() {
  genshin();
  std::string str = “hello world”;
  fmt::print(“str = {}\n”, str);
  return 0;
}
```

终端输入

```
g++ main.cpp genshin.cpp -I ../include -o hello -lfmt -std=c++17
```

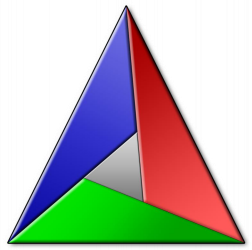
启动

```
./hello
```

想象一下，你有一百个源文件，用到了一百个库，你要怎么编译?

万一你不知道某个头文件的路径怎么编译?

使用cmake



CMake

cmake是一个用于构建大型c++项目的工具，使用cmake能一键生成makefile指导编译器使用正确的命令编译程序

创建一个CMakeLists.txt

```
# 要求该项目至少使用cmake3.15版本构建
cmake_minimum_required(VERSION 3.15)
# 项目名叫helloworld
project(helloworld)
# 搜索fmt库
find_package(fmt REQUIRED)
# 项目会编译出名字为hello的可执行文件，用到main.cpp和genshin.cpp
add_executable(hello src/main.cpp src/genshin.cpp)
# 相当于-I include
target_include_directories(hello PUBLIC include)
# 相当于-lfmt
target_link_libraries(hello fmt::fmt)
```

编译程序

```
mkdir build
cd build
cmake ..
make
# 运行
./hello
```

C++项目的一般结构

src /用于放源文件

include/用于放头文件

build/用于放编译过程中产生的中间文件以及最后的可执行文件

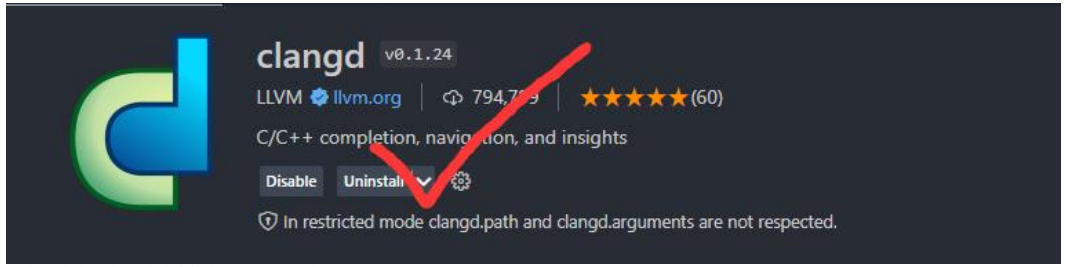
大部分的c++项目都可以用以下操作编译安装

```
mkdir build & cd build
cmake ..
make
sudo make install
```

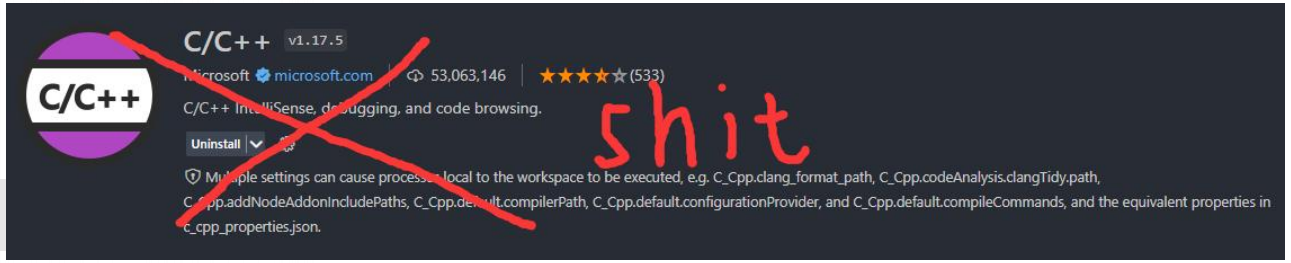
```
.
├── CMakeLists.txt
├── README.md
├── include
│   └── lms
│       ├── database
│       │   ├── author.hpp
│       │   ├── book.hpp
│       │   ├── database.hpp
│       │   └── sale.hpp
│       └── gui
│           ├── add_book_window.hpp
│           ├── different_view_window.hpp
│           ├── edit_book_window.hpp
│           ├── main_window.hpp
│           ├── query_window.hpp
│           └── sorted_books_window.hpp
└── src
    ├── database
    │   ├── author.cpp
    │   ├── book.cpp
    │   ├── database.cpp
    │   └── sale.cpp
    ├── gui
    │   ├── add_book_window.cpp
    │   ├── different_view_window.cpp
    │   ├── edit_book_window.cpp
    │   ├── main_window.cpp
    │   ├── query_window.cpp
    │   └── sorted_window.cpp
    └── main.cpp
```

Clangd+VScode开发环境

VScode安装clangd插件



禁用VScode官方C/C++ 插件



在CMakeLists.txt增加一行

```
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

使用cmake命令，这时候会自动生成compile_commands.json文件

clangd会自动识别这个文件，搜索代码的依赖，实现提示和补全

C++基础
一

命名空间namespace

- 命名空间（namespace）用于划分代码，使得代码具有更清晰的树形结构
- 在C语言中，两个函数是不能重名的，当出现重名函数时必须要修改其中的一个函数名，否则编译会出现错误，这在大型项目的开发中会造成许多麻烦
- 在C++中，命名范围是由命名空间划分的，不同的命名空间中的函数可以重名

```
namespace A {
    void print() {
      std::cout<<”A\n”;
    }
}
```

```
namespace B {
    void print() {
      std::cout<<”B\n”;
    }
}
```

```
void testNamespace() {
  A::print(); // 输出A
  B::print(); // 输出B
}
```

命名空间namespace

- 使用using namespace 某命名空间的名字; 可以让后面的代码自动在该命名空间下找到函数和对象（不推荐）

```
using namespace std;
cout<<”hello world\n”// 等同于std::cout
```

- 命名空间可以嵌套

```
namespace A {
namespace a {
  void foo() {} // 使用A::a::foo()调用
}
}
```

引用

- 引用与指针类似，它可以直接指向内存中某个具体的对象
- 在一些教科书中，会称引用为"起别名"，其实类比指针更好理解
- 使用&符号创建一个引用
- 当你修改引用的时候，原对象也会一起被修改
- 引用在创建时，必须指明引用的对象，不能创建空的引用。

```
int a = 5;
int &ref_to_a = a; //ref_to_a是a的一个引用
std::cout<<ref_to_a<<"\n"; // 5
ref_to_a += 1;
std::cout<<a<<"\n"; // 6，说明修改引用，原始对象也会一起修改
```

引用

- 对比下面两段代码 值传递
- 使用引用传递，与指针传递一样能修改原变量的值，而且不需要写大量丑陋的"*"，提高了代码的可读性

```
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}
int x = 5, y = 6;
swap(&x, &y);
```

```
void swap(int &a, int &b) {
  int temp = a;
  a = b;
  b = temp;
}
int x = 5, y = 6;
swap(x, y);
```

引用

- C++在函数调用的过程，会对传入的对象进行拷贝。对于一些内存占用比较大的对象（例如一张图像），值拷贝会非常耗时，而引用拷贝会很快（引用像指针一样，只占固定的内存），所以对于一些大的对象，我们通常会把参数写成引用的格式。

```
cv::Mat detectArmors(const cv::Mat &src); //使用引用传参
```

类与对象

- C++中最重要的概念就是类和对象，在引入类之前我们先来做一个编程联系
- 假设你要设计一款叫做原神的游戏，游戏里有两种角色，分为Hero和Dragon，Hero和Dragon采用回合制战斗，Hero和Dragon有自己攻击力和防御力，每个角色的进攻回合会对敌人造成（攻击力-防御力）的伤害

```
struct Hero {
  int attack_power;
  int defense_power;
  int HP;
}

struct Dragon {
  int attack_power;
  int defense_power;
  int HP;
}
```

```
void HeroAttack(Hero &hero, Dragon &dragon) {
  int damage = hero.attack_power - dragon.defense_power;
  if (damage < 0) {
    damage = 0;
  }
  dragon.HP -= damage;
}
```

类与对象

- 为了实现代码的可扩展性，编程语言出现了面向对象的思想。
- 对象将属性和操作封装在一起，用户只需要关心他怎么用，而不再关心其内部的实现

```
class Hero {
public:
  Hero(int HP, int AP, int DP);
  void attack(Dragon &dragon);
  int HP;
  int attack_power;
  int defense_power;
private:
  int magic();
  int mana;
}
```

```
while (Zhongli.HP > 0 && Midir.HP > 0) {
  Zhongli.attack(Midir);
  if (Midir.HP < 0) {
        std::cout<<”钟离Win\n”;
        break;
  }
  Midir.attack(Zhongli);
  if (Zhongli.HP < 0) {
        std::cout<<”米迪尔Win\n”;
        break;
  }
}
```

类与对象

- 类的方法里有个隐藏的变量，this指针，它指向这个方法的调用者
- 静态方法和静态属性是所有这个class实例化出来的对象所共有的
- public是对外可见的，private和protected是对外不可见的

```
class Hero {
public:
  int HP;
  int attack_power;
  int defense_power;
  const Hero & higherHP(const Hero & other) {
    return this->HP > other.HP ? *this : other;
  }
};
```

继承

- 对有同样属性的对象，我们可以使用继承来避免编写大量重复的代码。

```
class Character {
public:
  Character(int HP, int AP, int DP);
  int HP, attack_power, defense_power;
  void attack(Character enemy);
}

class Hero : public Character {
...
}

class Dragon : public Character {
...
}
```

运算符重载

- C++可以对运算符进行重载，实现class间的加减乘除等操作

```
class Vector {
public:
  Vector(int x, int y) : x(x), y(y) {}
  Vector operator+ (const Vector & v2) {
    return Vector(x + v2.x,  y + v2.y);
  }
  int x, y;
}

Vector v1 = Vector(0,1);
Vector v2 = Vector(1,0);
Vector v3 = v1 + v2;
std::cout << v3.x << “ “<<v3.y<<”\n”;
```

练习

Leetcode 2069. 模拟行走机器人

（https://leetcode.cn/problems/walking-robot-simulation-ii/）