L2：C++基本语法

亢孟军 武汉大学
mengjunk@whu.edu.cn

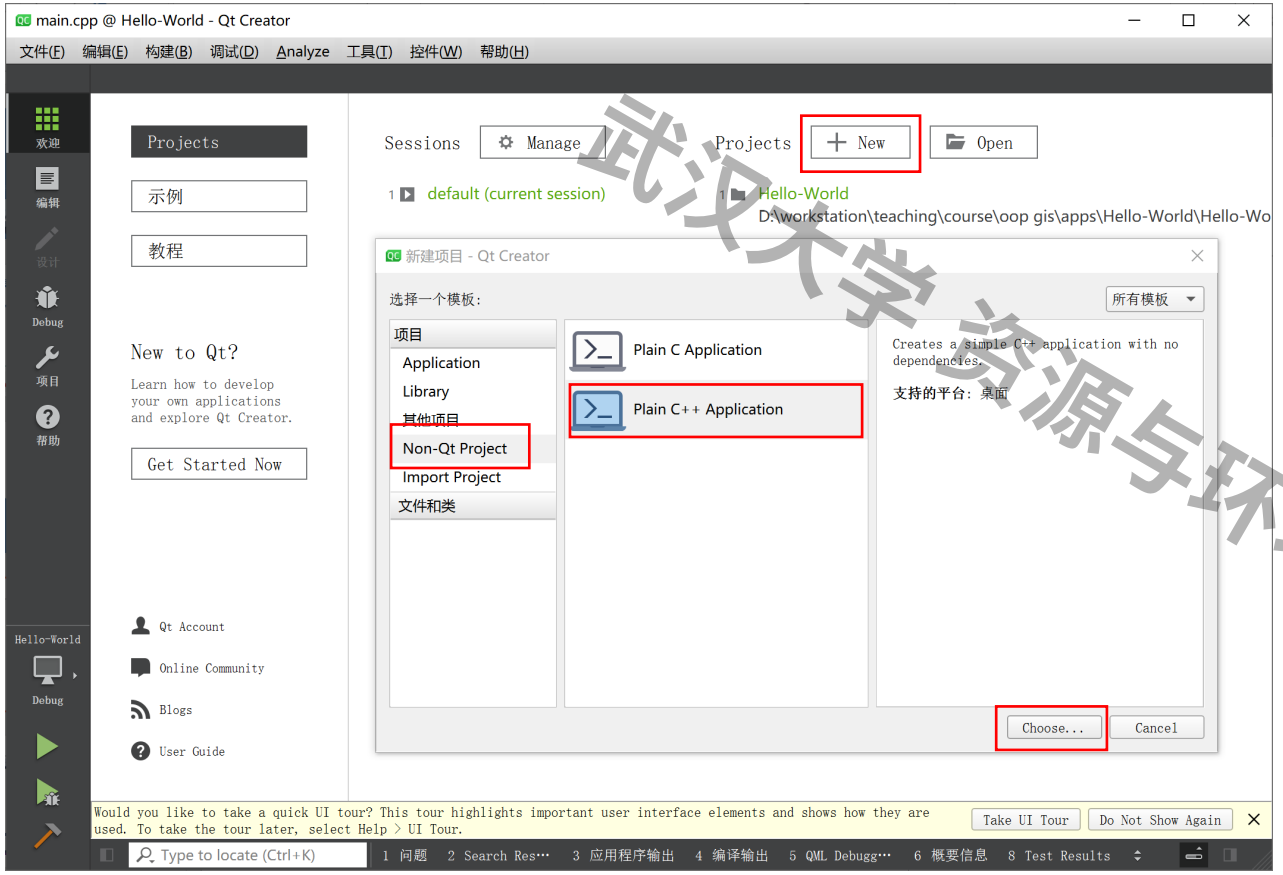
目录

本节课目的

介绍一个简单程序的组成，方便大家学习的同时，能够写测试程序

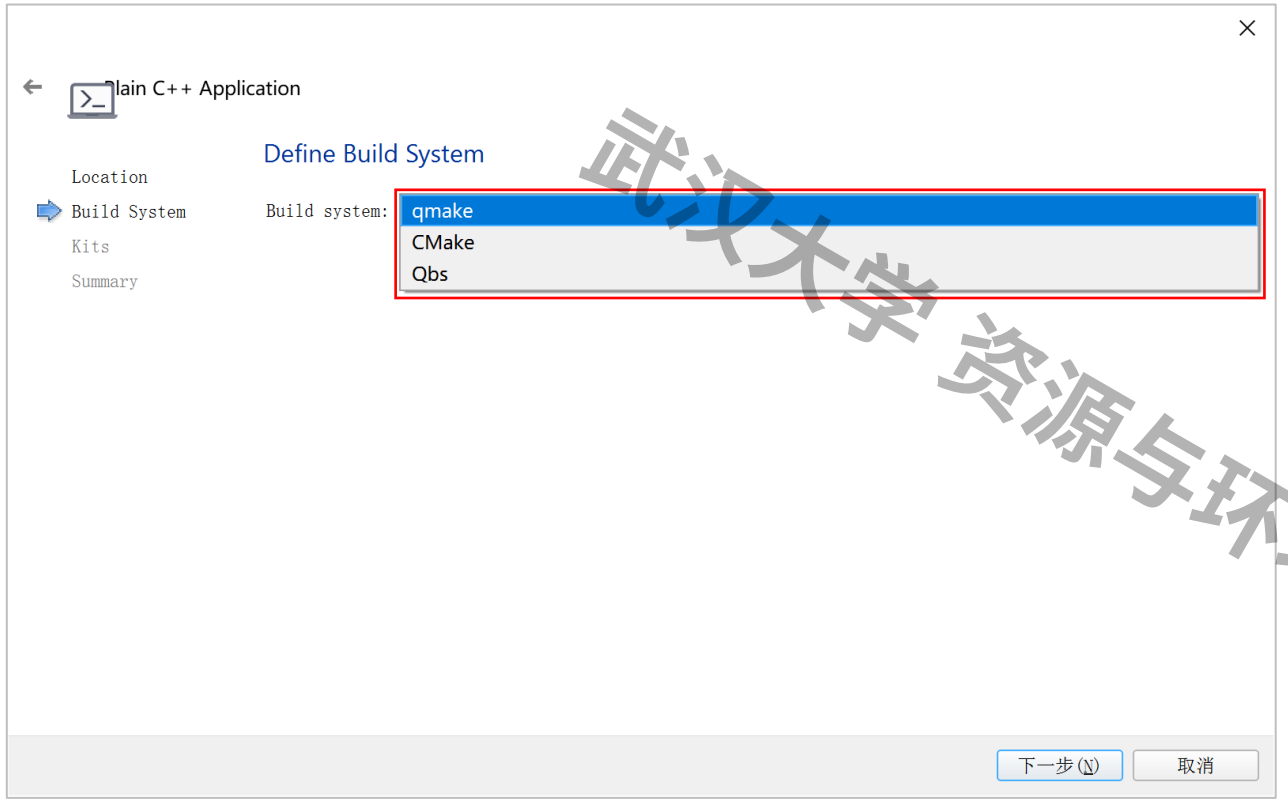
编写第一个C++程序

新建一个Non-QT项目



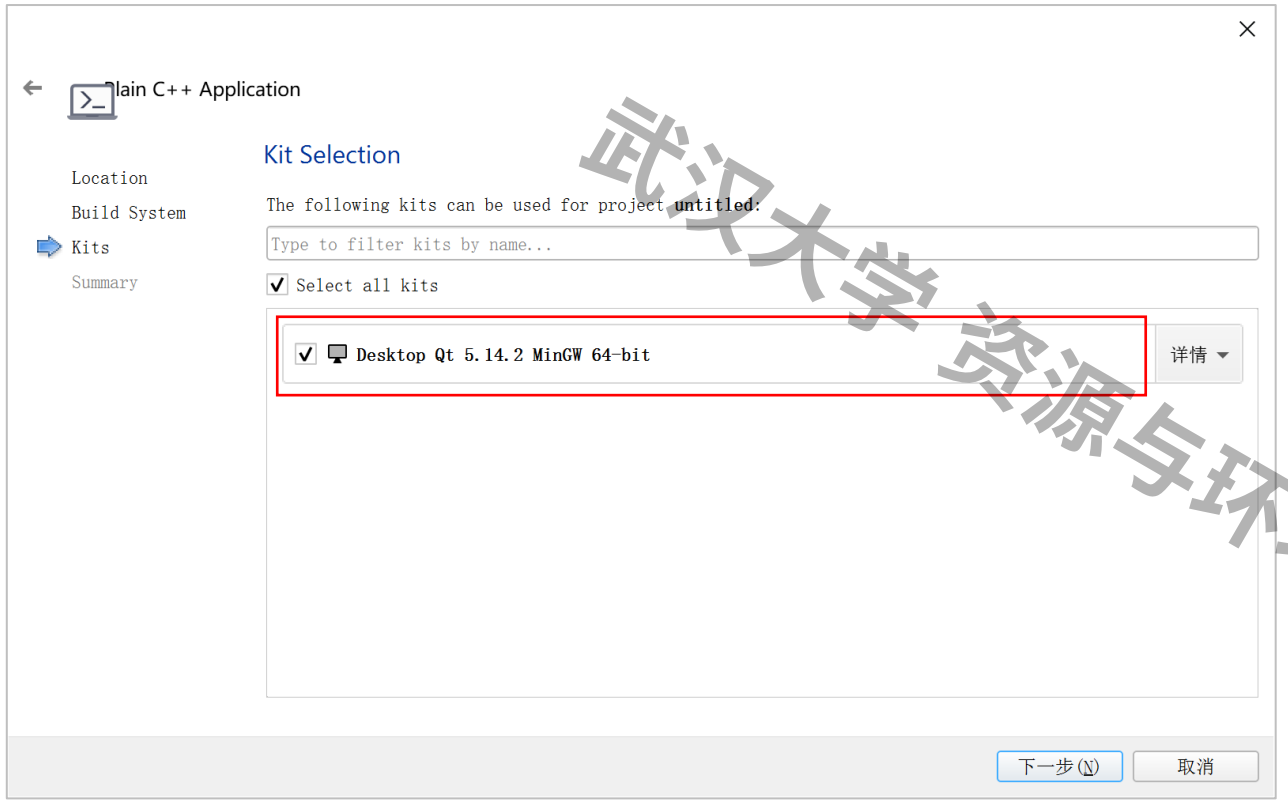
- **Non-QT项目**：项目中不包含任何QT相关库文件
- **简洁**、适合作为入门学习

新建一个Non-QT项目



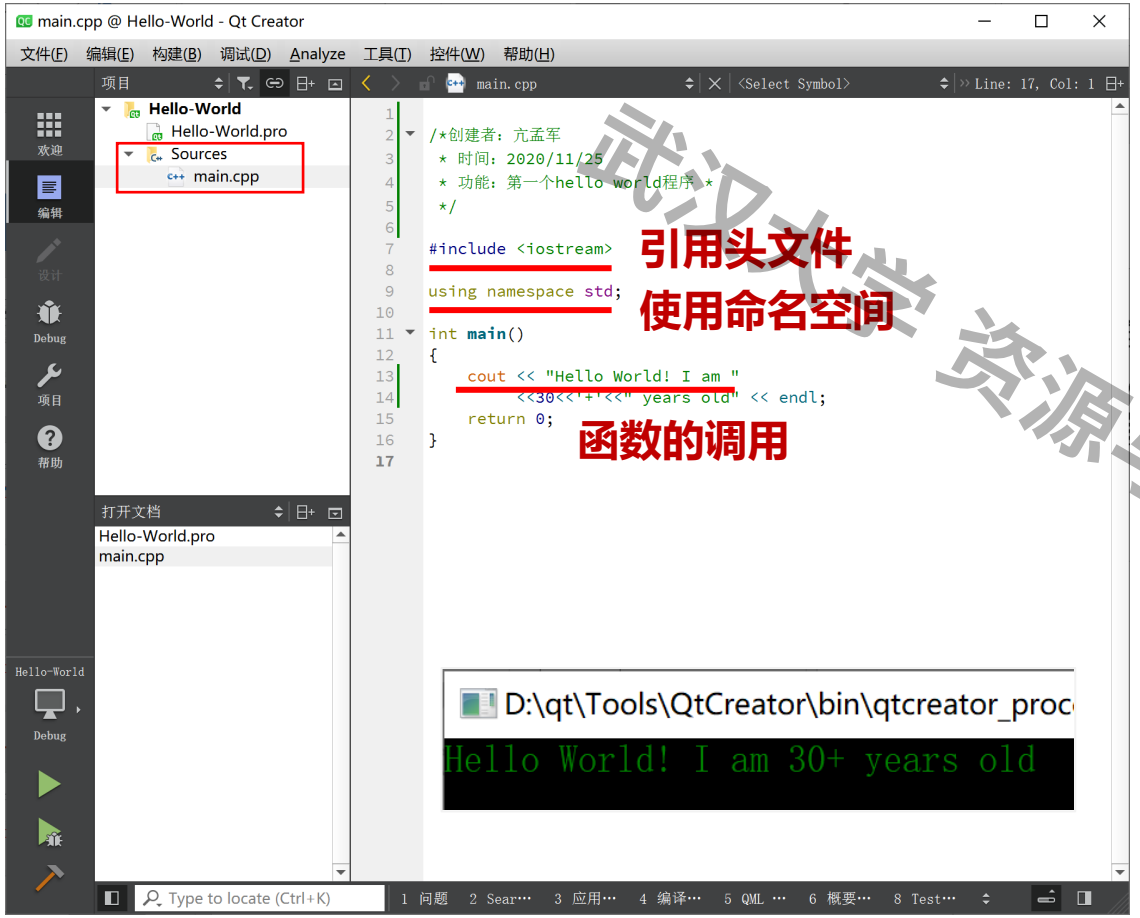
- **qmake, Cmake & Qbs:** 是三种不同的项目管理工具
- 一个项目往往包含多个文件
- 编译器一次编译一个源码文件
- 项目管理工具负责组织多个源码文件，并构建编译逻辑
- qmake为QT Creator量身打造
- **QT Creator默认使用qmake**
- qmake通过**.pro**文件管理项目

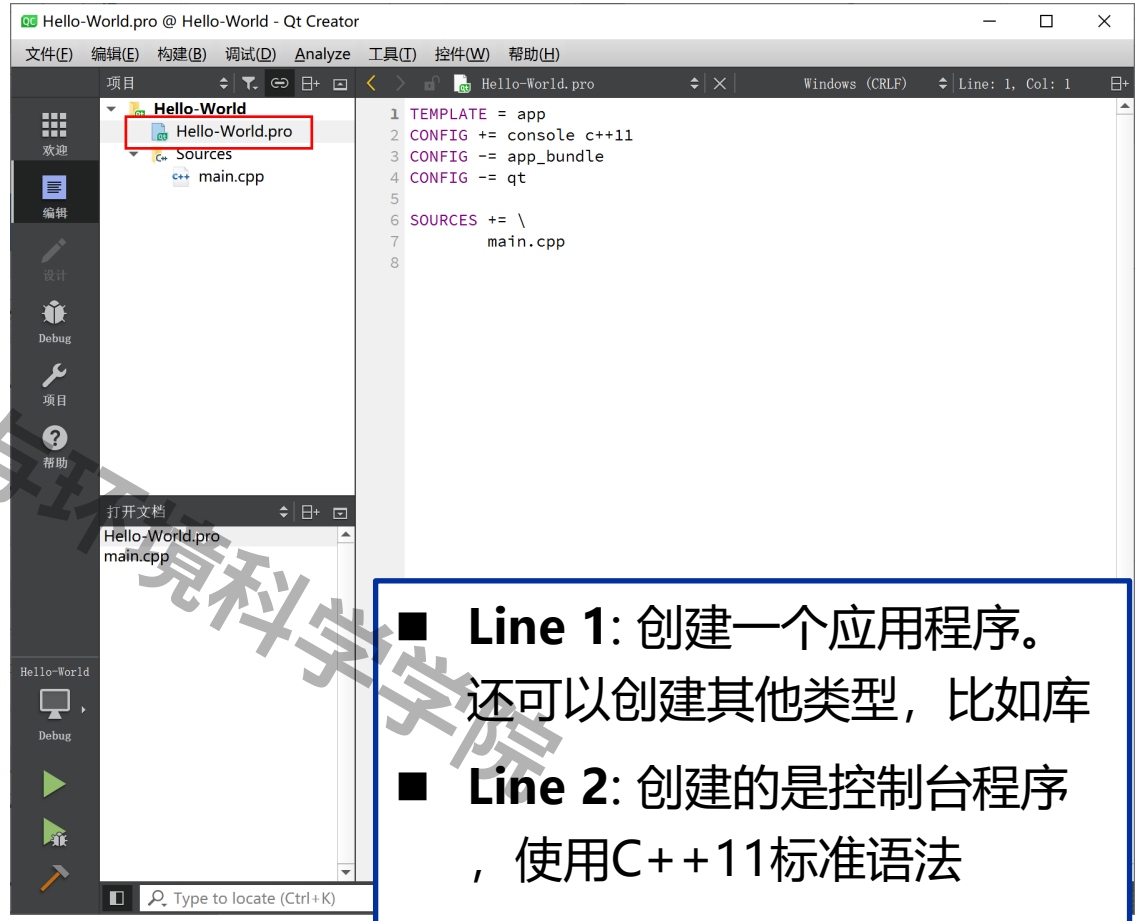
新建一个Non-QT项目



- 自动识别到本机已安装的编译器MinGW
- **自己查阅什么是MinGW?**

新建一个Non-QT项目



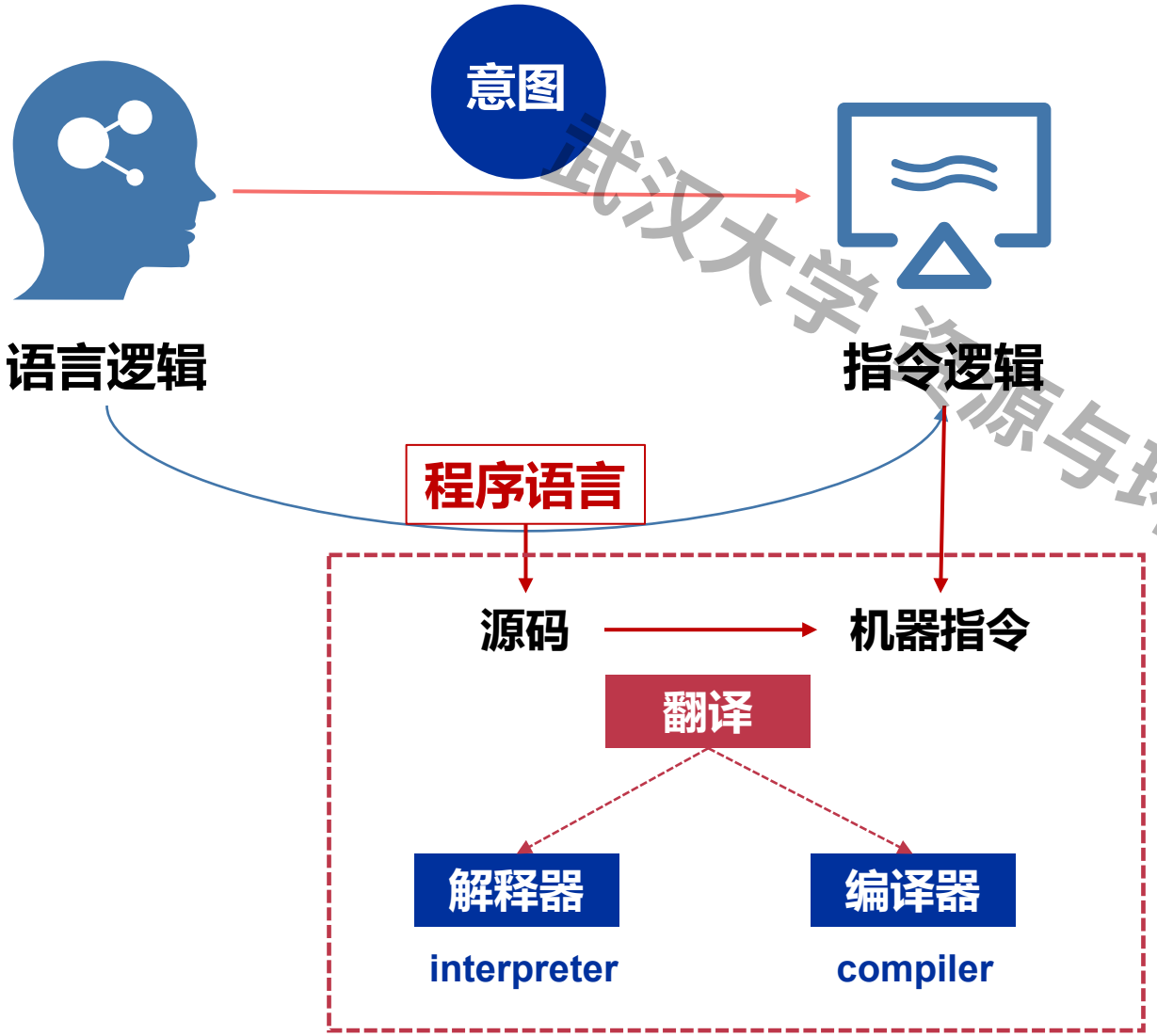


- **Line 1**: 创建一个应用程序。还可以创建其他类型，比如库
- **Line 2**: 创建的是控制台程序，使用C++11标准语法
- **Line 3-4**: 去掉或不构建
- **Line 5**: 包含的源文件

源码 -> 执行

语言的*翻译*过程

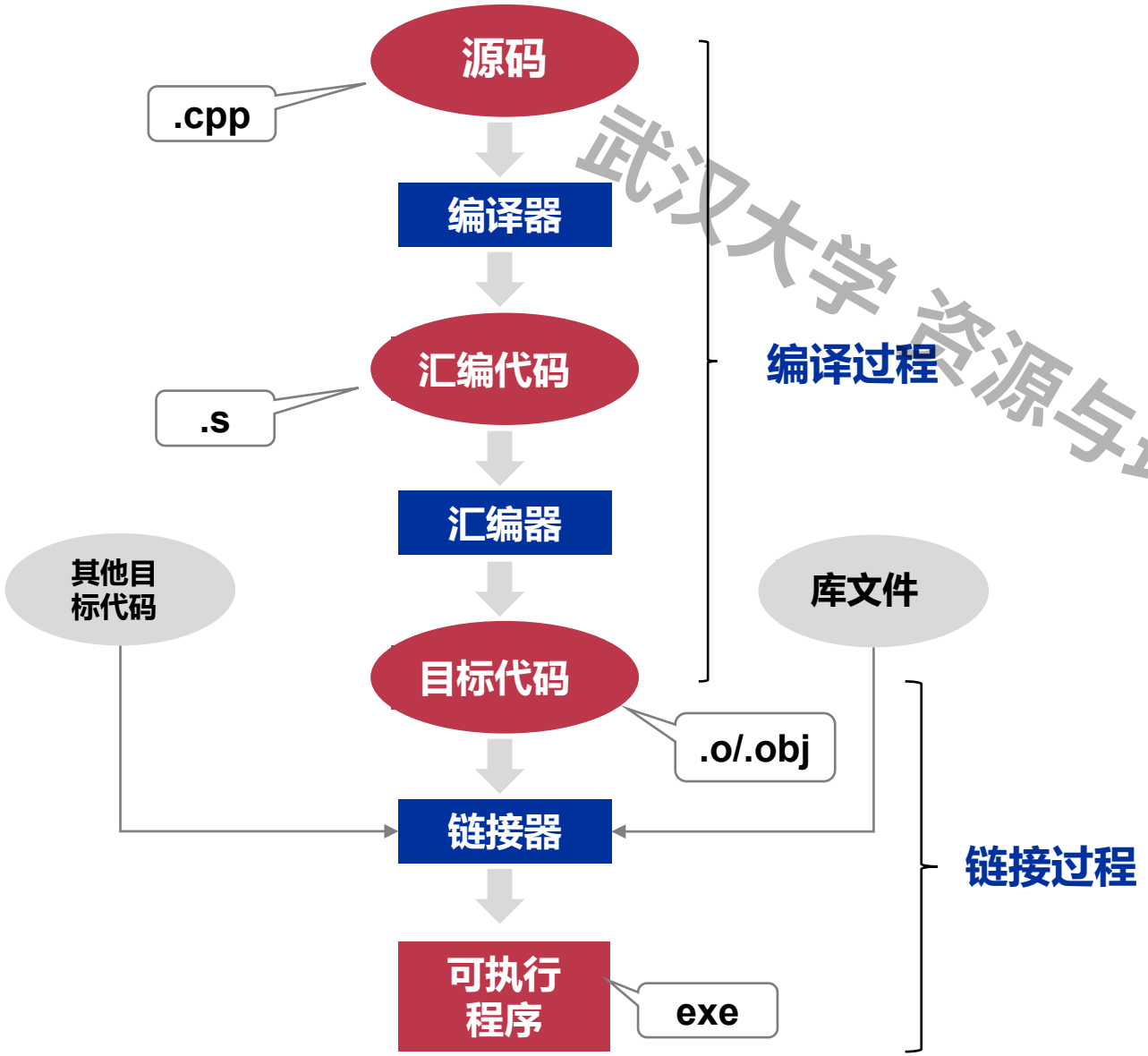
源码 -> 执行



- **解释器**：将源码转化为一些动作（可由多组机器指令组成）并立即执行这些动作，如basic, R
- **编译器**：直接把源码转化成汇编语言或机器指令，如C，C++

类别	优点	缺点
解释器	• 边编码边执行	• 不适合大型工程
编译器	• 可执行程序小 • 运行速度快 • 适合大型工程 • 易于调试	• 过程复杂

编译的过程



编译过程

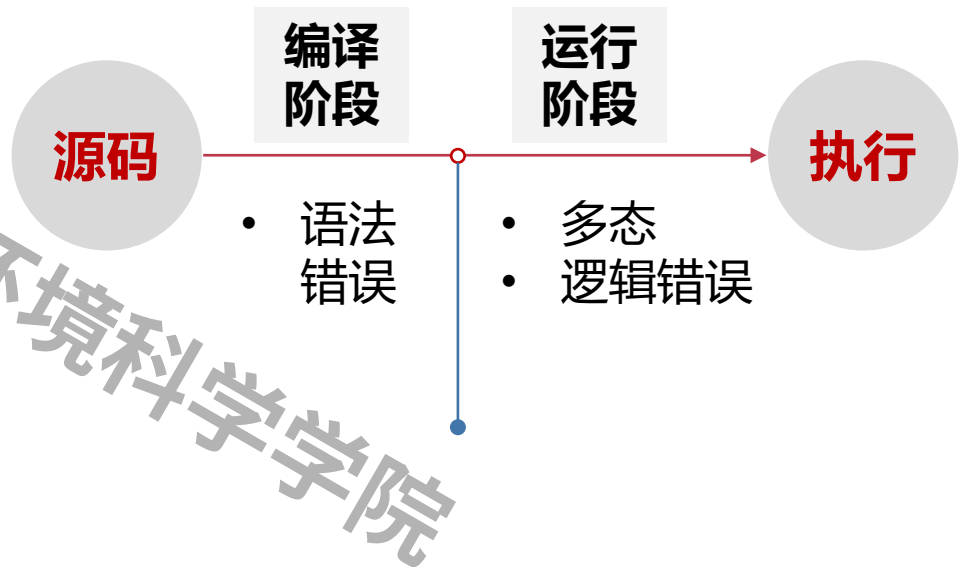
① 语法分析，静态类型检查（**发现语法错误**）

② 遍历语法树，生成汇编语言或机器代码

③ 链接器把目标快连接成可执行文件

构建项目 → 运行

编译和运行



C++是一套标准，经过了漫长的发展

也在持续更新

C++标准的发展历程

- Template
- STL containers & algorithms
- I/O Streams

- Move semantics
- Unified initialization
- Auto and decltype
- Lambda expressions constexpr
- Multithreading and the memory model
- Regular expressions
- Smart pointer
- Hash tables
- std::array

- Reader-writer locks
- Generic laxmbda functions

- Fold expressions
- constexpr if
- structured binding
- std::string_view
- Parallel algorithms of the STL
- Filesystem library
- std::any, std::optional and std::variant

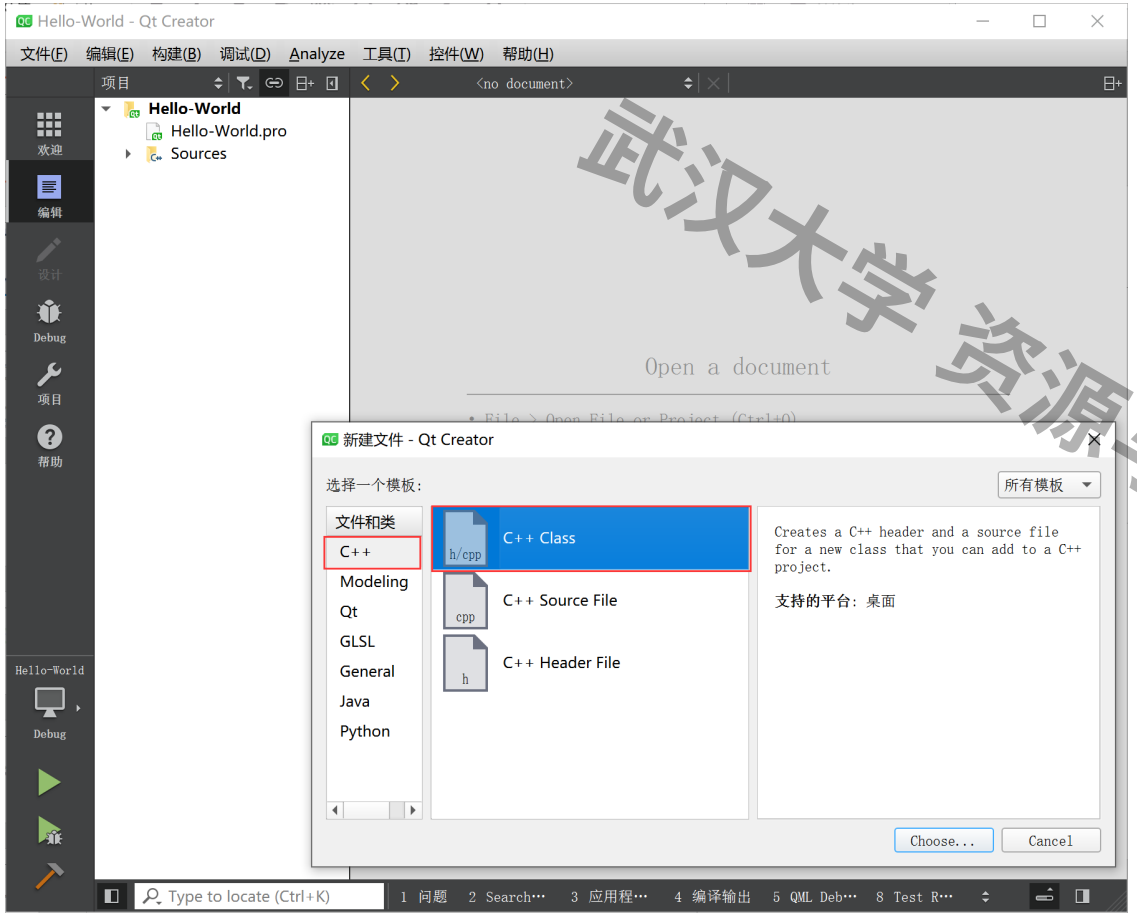
- Coroutines
- Modules
- Concepts
- Ranges library

- Deducing this
- Modularized standard library
- print and println
- Flat associative containers
- std::expected
- Improved ranges
- std::mdspan
- std::generator

包含头文件

include

一般化的C++项目结构



- 通常，一个C++项目由**多个文件**组成，主要是类
- 使用**类向导**创建类更方便

声明和定义

```cpp
/*创建者：亢孟军
 * 时间：2020/11/25
 * 功能：第一个hello world程序 *
 */

#include <iostream>

extern int ga;
int sum(int,int);

using namespace std;
int main()
{
    cout << "Hello World! I am "
         <<ga<<'+'<<" years old" << endl;

    cout<<sum(10,20)<<endl;

    return 0;
}
```

变量&函数的**声明**

main.cpp

```cpp
int ga =10;

int sum(int m,int n)
{
    return m+n;
}
```

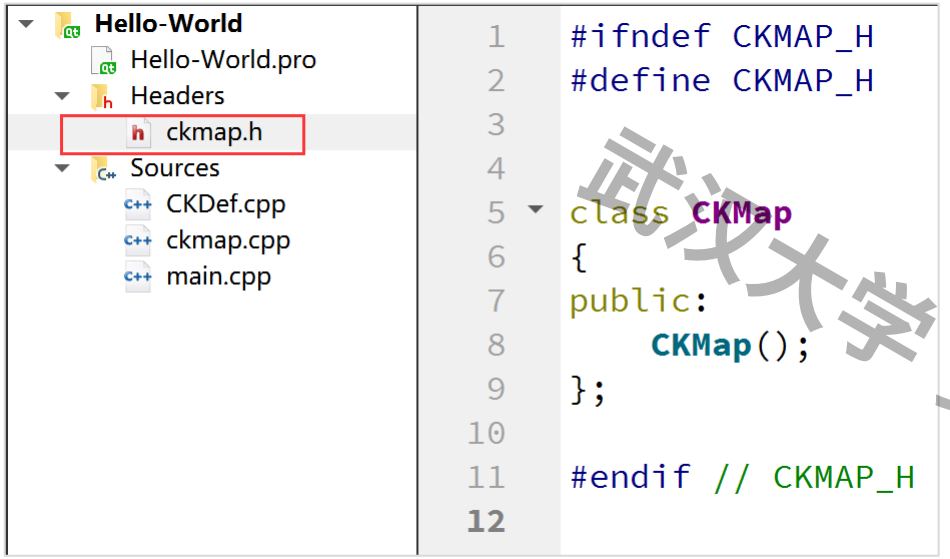
变量&函数的**定义**

CKDef.cpp

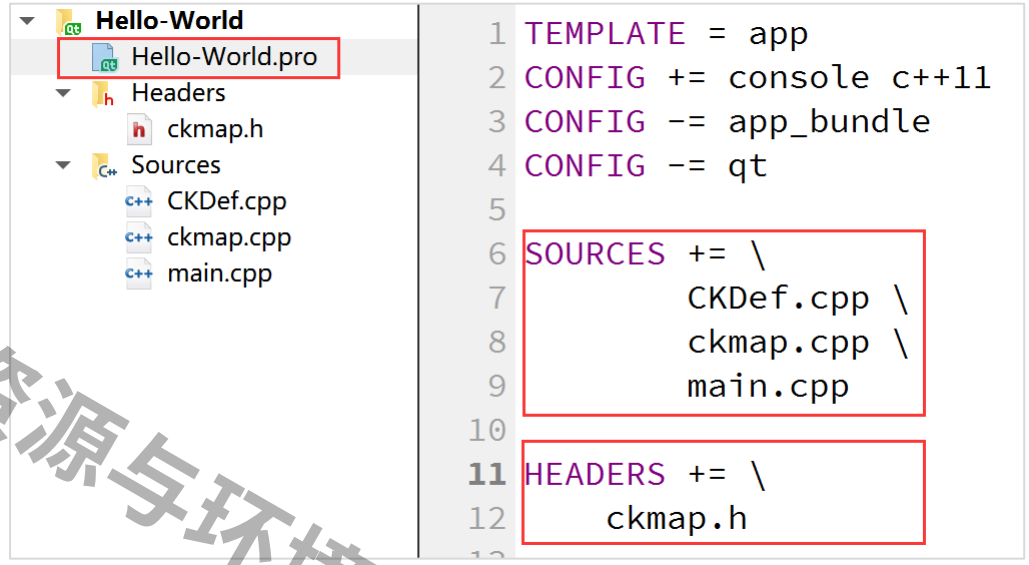
- 一个C++项目由**多个文件**组成，主要是类
- 使用**类向导**创建类更方便

- **声明**：declaration, 是向编译器介绍标识符号，其定义在某个地方。**可多次使用声明**
- **定义**：变量或函数的实现。**一次定义**

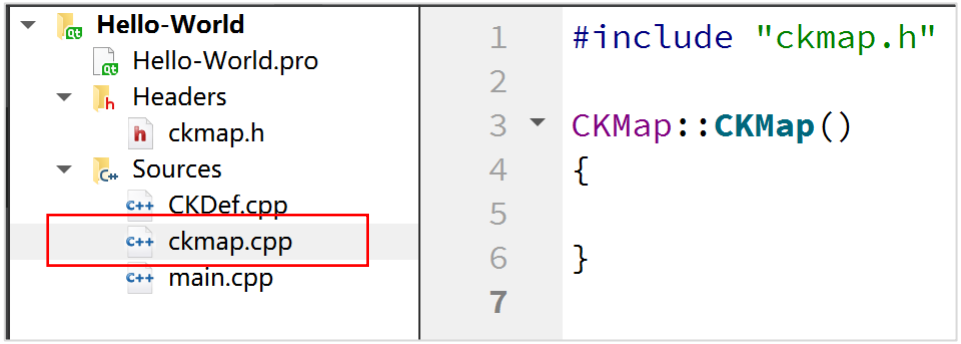
包含头文件




```
#ifndef CKMAP_H
#define CKMAP_H


class CKMap
{
public:
    CKMap();
};

#endif // CKMAP_H

```




```
TEMPLATE = app
CONFIG += console c++11
CONFIG -= app_bundle
CONFIG -= qt

SOURCES += \
        CKDef.cpp \
        ckmap.cpp \
        main.cpp

HEADERS += \
    ckmap.h
```




```
#include "ckmap.h"

CKMap::CKMap()
{

}

```

- 向导构建的类包含：.h & .cpp文件
- **.h**：包含声明
- **.cpp**：包含定义

include

WHY include

有些头文件有.h

有些没有?

- 旧的 C++ 头文件，如 iostream.h、fstream.h 继续被支持
  - ✓ **这些头文件在全局作用域中，+.h**
- 新的 C++ 头文件，如 iostream、fstream 等包含的基本功能和对应的旧版头文件相似
  - ✓ **头文件的内容在命名空间 std 中，include without .h**
- 标准C头文件如 stdio.h、stdlib.h 等继续被支持
  - ✓ **这些头文件在全局作用域中，+.h**
- 新C++头文件具有如 cstdio、cstdlib，**include without .h**

命名空间是一个更大的作用域，使用其中的标识需include first

/05

命名空间

命名空间

- 一个软件往往由多名程序员共同开发，会使用大量的变量和函数，不可避免地会出现变量或函数的命名冲突
- **C++ 引入了命名空间（Namespace）解决命名冲突**

```
01. namespace Li{  //小李的变量定义
02.     FILE fp = NULL;
03. }
04. namespace Han{  //小韩的变量定义
05.     FILE fp = NULL
06. }
```

- **示例**

小李和小韩都参与了一个文件管理系统的开发，它们都定义了一个全局变量 fp，用来指明当前打开的文件，将他们的代码整合在一起编译时，很明显编译器会提示 fp 重复定义（Redefinition）错误

命名空间

- 关键字namespace，用来定义一个命名空间
- 其语法格式为：

```
namespace name{
   //variables, functions, classes
}
```

- Name是命名空间的名字，包含变量、函数、类、typedef、#define等，最后由{}包围
- 使用变量、函数时要指明它们所在的命名空间

```
01.  Li::fp = fopen("one.txt", "r");  //使用小李定义的变量 fp
02.  Han::fp = fopen("two.txt", "rb+");  //使用小韩定义的变量 fp
```

- :: 是域解析操作符，用来指明要使用的命名空间

命名空间

- 直接使用**域解析操作符**，或 using 关键字声明，例如：

```
01.  using Li::fp;
02.  fp = fopen("one.txt", "r");  //使用小李定义的变量 fp
03.  Han :: fp = fopen("two.txt", "rb+");  //使用小韩定义的变量 fp
```

- 在代码的开头用using声明了 Li::fp，它的意思是，using 声明以后的程序中如果出现了未指明命名空间的 fp，就使用 Li::fp；但是若要使用小韩定义的 fp，仍然需要 Han::fp
- **using 声明不仅可以针对命名空间中的一个变量，也可以用于声明整个命名空间**

```
01.  using namespace Li;
02.  fp = fopen("one.txt", "r");   //使用小李定义的变量 fp
03.  Han::fp = fopen("two.txt", "rb+");   //使用小韩定义的变量 fp
```

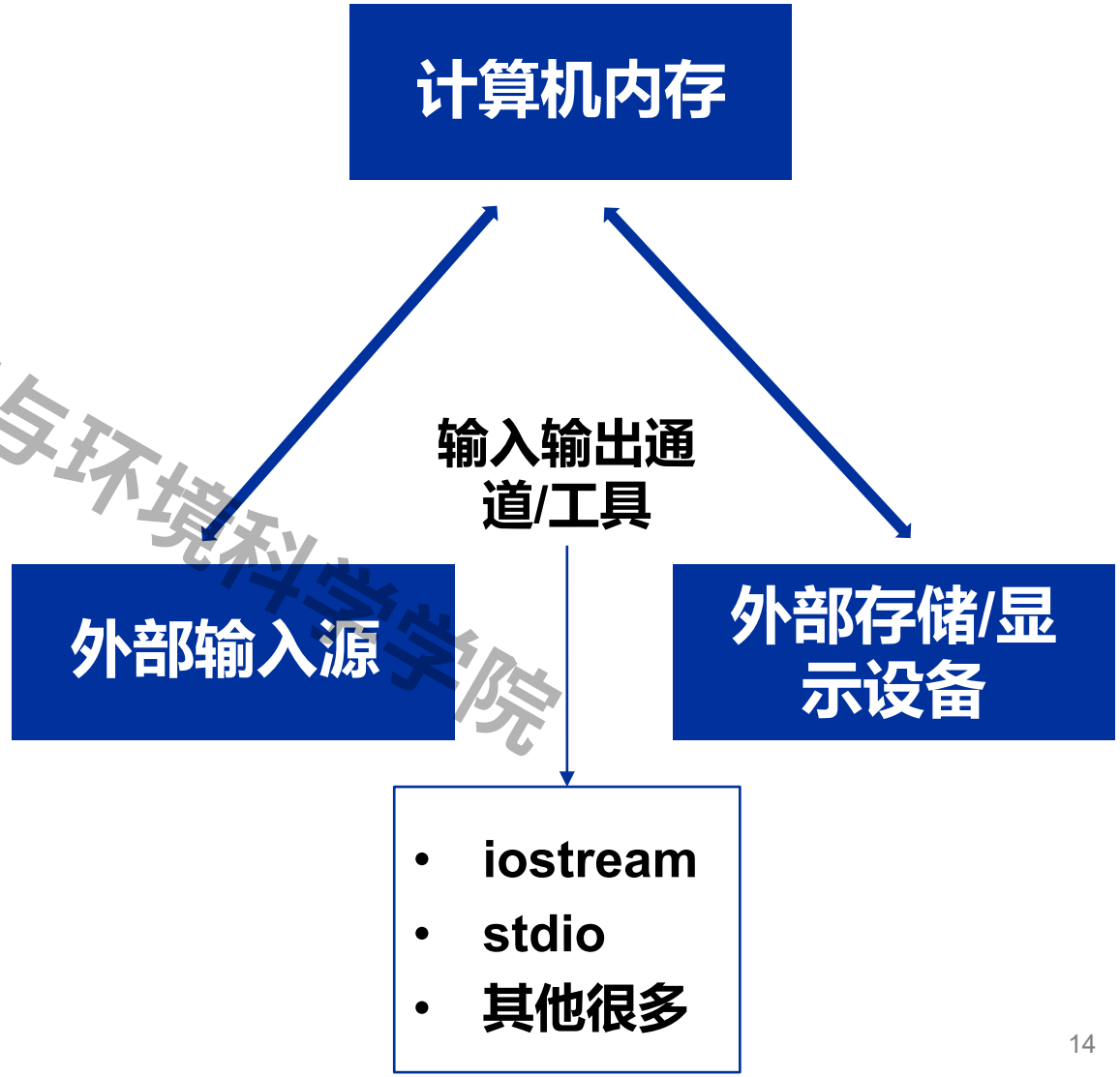
输入输出流

- **输入**: 解决外部数据进入到计算机内存
- **输出**：内存数据到外部设备，包括屏幕、硬盘、打印机、投影等

输入输出流

- C使用 **scanf** 和 **printf** 来对数据进行输入输出操作
- **C++使用iostream实现**

```
01. #include<iostream>
02. using namespace std;
03. int main(){
04.     int x;
05.     float y;
06.     cout<<"Please input an int number:"<<endl;
07.     cin>>x;
08.     cout<<"The int number is x= "<<x<<endl;
09.     cout<<"Please input a float number:"<<endl;
10.     cin>>y;
11.     cout<<"The float number is y= "<<y<<endl;
12.     return 0;
13. }
```



输入输出流

- C++ 中的输入与输出可以看做是一连串的数据流，输入即可视为从文件或键盘中输入程序中的一串数据流，而输出则可以视为从程序中输出一连串的数据流到显示屏或文件中。
- 在编写 C++ 程序时，如果需要使用输入输出时，则需要包含头文件iostream，它包含了用于输入输出的对象，例如常见的cin表示标准输入、cout表示标准输出、cerr表示标准错误

> iostream 是 Input Output Stream 的缩写，意思是“输入输出流”。

- **cout 和 cin 都是 C++ 的内置对象**，而不是关键字
- 使用 cout 进行输出时需要紧跟<<运算符，使用 cin 进行输入时需要紧跟>>运算符，这两个运算符可以自行分析所处理的数据类型

字符串

- **文本**是现实世界的一种叙述性描述
- 有别于**数值型**
- 字符串是文本数据的结构化形式
- 字符组成了字符串
- 字符的本质是整数，因整数是计算机的计算基础

字符串

- C++ 的字符处理：char, char*, string类
- **STL**内置的 string 类，具有完备的字符串处理功能
- 使用 string 类需要包含头文件<string>
- **<string>是在std中的**

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1;
07.     string s2 = "c plus plus";
08.     string s3 = s2;
09.     string s4 (5, 's');
10.     return 0;
11. }
```

变量 s1 只是定义但没有初始化，编译器会将默认值赋给 s1，默认值是""，即空字符串。

变量 s2 在定义的同时被初始化为"c plus plus"。与C风格的字符串不同，string 的结尾没有结束标志'\0'。

变量 s3 在定义的时候直接用 s2 进行初始化，因此 s3 的内容也//是"c plus plus"。

变量 s4 被初始化为由 5 个's'字符组成的字符串，也就是"sssss"。

字符串

- 调用 string 类提供的 length() 函数获取字符串长度

```
01.  string s = "http://c.biancheng.net";
02.  int len = s.length();
03.  cout<<len<<endl;
```

- string 的末尾没有'\0'字符，所以 length() 返回的是字符串的真实长度，而不是长度 +1

字符串——转为C风格的字符串

- 虽然 C++ 提供了 string 类来替代C语言中的字符串，但是在实际编程中，有时候必须要使用C风格的字符串（例如打开文件时的路径），为此，string 类为我们提供了一个转换函数 c_str()，该函数能够将 string 字符串转换为C风格的字符串，并返回该字符串的 const 指针（const char*）。请看下面的代码：

```
01.  string path = "D:\\demo.txt";
02.  FILE *fp = fopen(path.c_str(), "rt");
```

- 为了使用C语言中的 fopen() 函数打开文件，必须将 string 字符串转换为C风格的字符串

字符串——字符串的输入输出

- string 类重载了输入输出运算符，可以像对待普通变量那样对待 string 变量，也就是用>>进行输入，用<<进行输出。请看下面的代码：

```
01. #include <iostream>
02. #include <string>
03.
04. using namespace std;
05.
06. int main(){
07.     string s;
08.     cin>>s;  //输入字符串
09.     cout<<s<<endl;  //输出字符串
10.     return 0;
11. }
```

代码示例

字符串——访问字符串中的字符

- string 字符串也可以像C风格的字符串一样按照下标来访问其中的每一个字符。string 字符串的起始下标仍是从 0 开始。

```cpp
#include <iostream>
#include <string>
using namespace std;

int main(){
    string s = "1234567890";
    for(int i=0,len=s.length(); i<len; i++){
        cout<<s[i]<<" ";
    }
    cout<<endl;
    s[5] = '5';
    cout<<s<<endl;
    return 0;
}
```

代码示例

字符串——字符串的拼接

- 有了 string 类，可以使用+或+=运算符来直接拼接字符串，不需要再使用C语言中的 strcat()、strcpy()、malloc() 等函数来拼接字符串了，也不用担心空间不够溢出了。
- 用+来拼接字符串时，运算符的两边可以都是 string 字符串，也可以是一个 string 字符串和一个C风格的字符串，还可以是一个 string 字符串和一个字符数组，或者是一个 string 字符串和一个单独的字符。

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1 = "first ";
07.     string s2 = "second ";
08.     char *s3 = "third ";
09.     char s4[] = "fourth ";
10.     char ch = '@';
11.
12.     string s5 = s1 + s2;
13.     string s6 = s1 + s3;
14.     string s7 = s1 + s4;
15.     string s8 = s1 + ch;
16.
17.     cout<<s5<<endl<<s6<<endl<<s7<<endl<<s8<<endl;
18.
19.     return 0;
20. }
```

代码示例

```
first second
first third
first fourth
first @
```

运行结果

字符串——字符串的增删改查

插入字符串

- insert() 函数可以在 string 字符串中指定的位置插入另一个字符串，它的一种原型为：

```
string& insert (size_t pos, const string& str);
```

- pos 表示要插入的位置，也就是下标；str 表示要插入的字符串，它可以是 string 字符串，也可以是C风格的字符串

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1, s2, s3;
07.     s1 = s2 = "1234567890";
08.     s3 = "aaa";
09.     s1.insert(5, s3);
10.     cout<< s1 <<endl;
11.     s2.insert(5, "bbb");
12.     cout<< s2 <<endl;
13.     return 0;
14. }
```

代码示例

```
12345aaa67890
12345bbb67890
```

运算结果

字符串——字符串的增删改查

删除字符串

- erase() 函数可以删除 string 中的一个子字符串。它的一种原型为：

```
string& erase (size_t pos = 0, size_t len = npos);
```

- pos 表示要删除的子字符串的起始下标，len 表示要删除子字符串的长度。如果不指明 len 的话，那么直接删除从 pos 到字符串结束处的所有字符（此时 len = str.length - pos）

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1, s2, s3;
07.     s1 = s2 = s3 = "1234567890";
08.     s2.erase(5);
09.     s3.erase(5, 3);
10.     cout<< s1 <<endl;
11.     cout<< s2 <<endl;
12.     cout<< s3 <<endl;
13.     return 0;
14. }
```

代码示例

```
1234567890
12345
1234590
```

运算结果

字符串——字符串的增删改查

提取子字符串

- substr() 函数用于从 string 字符串中提取子字符串，它的原型为：

```
string substr (size_t pos = 0, size_t len = npos) const;
```

- pos 表示要插入的位置，也就是下标；str 表示要插入的字符串，它可以是 string 字符串，也可以是C风格的字符串

```
01. #include <iostream>
02. #include <string>
03. using namespace std;
04.
05. int main(){
06.     string s1 = "first second third";
07.     string s2;
08.     s2 = s1.substr(6, 6);
09.     cout<< s1 <<endl;
10.     cout<< s2 <<endl;
11.     return 0;
12. }
```

代码示例

```
first second third
second
```

运算结果

字符串——字符串的增删改查

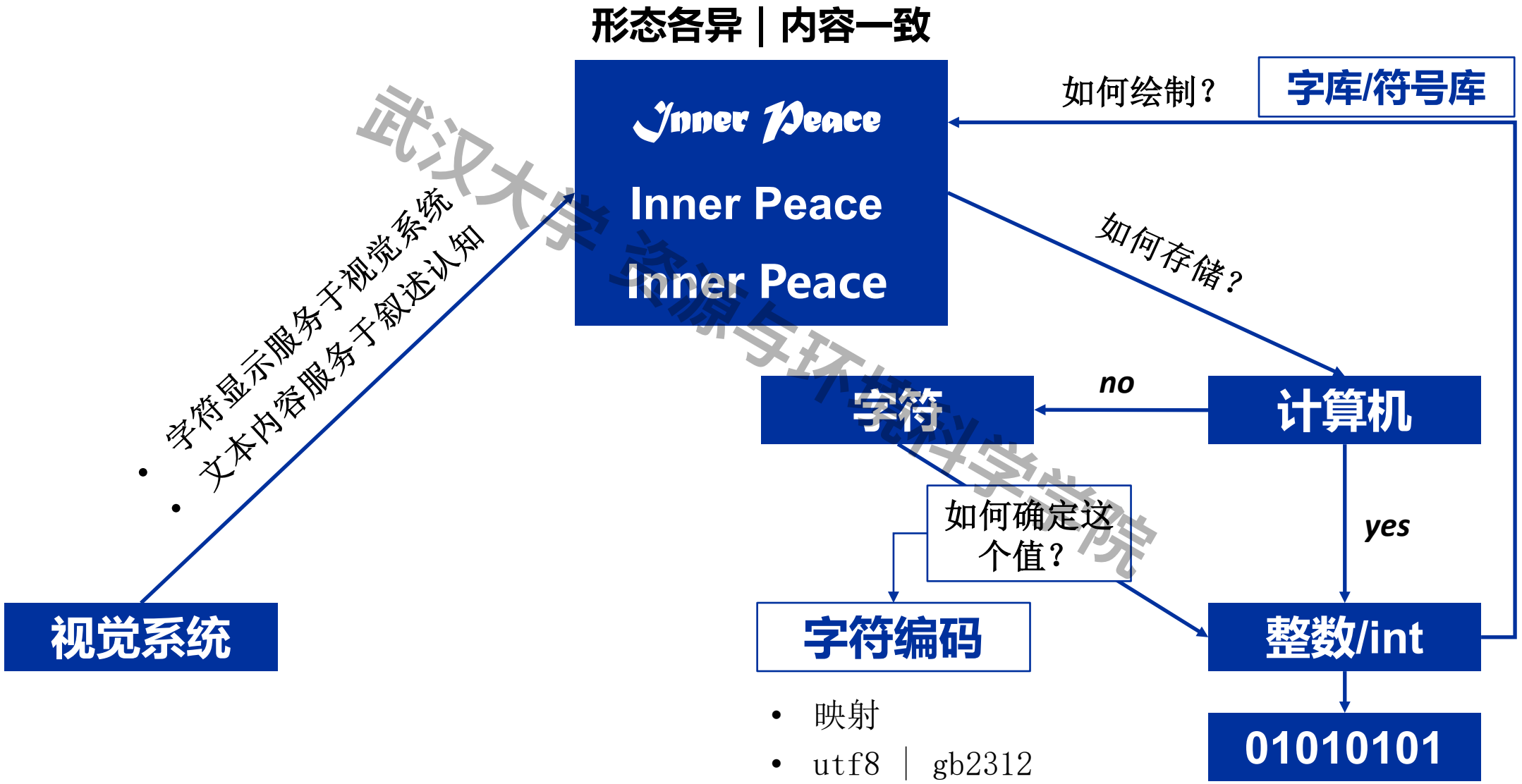
查找字符串

- string 类提供了几个与字符串查找有关的函数： find() 函数、 rfind() 函数、 find_first_of() 函数
- find() 函数用于在 string 字符串中查找子字符串出现的位置，它其中的两种原型为：

```
size_t find (const string& str, size_t pos = 0) const;
size_t find (const char* s, size_t pos = 0) const;
```

- rfind() 和 find() 很类似，同样是在字符串中查找子字符串，不同的是 find() 函数从第二个参数开始往后查找，而 rfind() 函数则最多查找到第二个参数处，如果到了第二个参数所指定的下标还没有找到子字符串，则返回一个无穷大值4294967295。
- find_first_of() 函数用于查找子字符串和字符串共同具有的字符在字符串中首次出现的位置。请看下面的代码：

再回顾下字符串的知识点



vector

ISO C++ standard is officially known as
ISO *International Standard ISO/IEC 14882:2020(E)*
– Programming Language C++

C++语法

C++ Standard Library

- classes
- functions

包含

Standard Template Library

- data structure
- algorithm
- container
- iterator

vector

- 它与一维数组类似，**更灵活的存储、更全面的功能、更安全的操作**
- 这些数据类型和算法不是C++语言的一部分，但是对内置数据类型的有益补充
- **using namespace std**
- 在STL中定义的数据类型通常称为容器（Container）。它们之所以称为容器，是因为他们存储和组织数据
- STL中有两种类型的容器：顺序容器和关联容器
- 顺序容器以序列方式组织数据，类似于数组
- 关联容器使用关键字组织数据，它允许对存储在容器中的元素进行快速随机访问
- **vector数据类型是一个顺序容器**

vector

- vector数据类型是一个序列式容器，它很像一个一维数组，其表现和数组相似的地方如下：

（1）矢量包括一系列值或者元素。

（2）矢量将其元素存储在连续的内存位置中。

（3）可以使用数组下标运算符[]访问矢量中的各个元素。

（4）但是，矢量相对于数组还有几个优点：

（5）不必声明矢量将具有的元素的数量。

（6）如果向已满的矢量添加值，则矢量将自动增加其大小以适应新值。

（7）矢量可以报告它们包含的元素的数量。

vector——定义和初始化

- 要在程序中使用矢量，则必须使用以下语句包含vector头文件： #include <vector>
- 创建矢量对象的语法与定义一个常规变量或数组所用的语法有所不同。以下是一个例子

```
vector<int> numbers;
```

- 这个语句将numbers定义为一个int的矢量。请注意，数据类型包含在尖括号内，紧跟在vector之后。由于矢量的大小随着添加值而扩大，因此不需要声明大小。
- 为矢量指定开始大小和初始化值。初始化值被复制到每个元素。示例如下：

```
vector<int> numbers(10,2);
```

- 也可以用另一个矢量中的值初始化一个矢量。例如，如果set1是已经有值的int矢量，则以下语句将创建一个名为set2的新矢量，它是set1的精确副本。

```
vector<int> set2(set1);
```

vector——成员函数

成员函数	描述	成员函数	描述
at(position)	返回vector中位于position位置的元素的值。	push_back(value)	在矢量的最后一个元素中存储一个值。如果矢量已满或为空，则会创建一个新元素。
	示例：x=vect.at(5);//将vect[5]的赋值给x		示例：vect.push_back(7); //在vect的最后一个元素中存储7.
capacity()	返回在不重新分配额外内存的情况下，可能存储在矢量中的最大元素数量（它和由size成员函数返回的值并不是一回事）。	reverse()	反转矢量中元素的顺序（最后一个元素变成第一个元素，第一个元素成为最后一个元素）。
	示例：x=vect.capacity();//将vect的容量赋值给x。		示例：vect.reverse(); //反转vect中元素的顺序
clear()	清楚矢量的所有元素。	resize(n)、resize(n,value)	调整矢量的大小以使其具有n个元素，其中n大于矢量当前大小。如果包含可选的参数value，则每个新元素都将使用该value值进行初始化。如果不指定value的话，所有没有元素的位置上都被初始化为0，
	示例：vect.clear();//从vect中移除所有元素		vect当前已经有4个元素情况下的示例：vect.resize(6,99);//将两个元素添加到矢量的末尾，每个元素的初始化为99
empty()	如果vector是空的，则返回true。否则，它返回false。	size()	返回矢量中元素的数量。示例：
	示例：if(vect.empty())//如果该矢量是空的 cout<<"The vector is empty."; //显示该消息		示例：numElements=vect.size();
pop_back()	从矢量中删除最后一个元素。	swap(vector2)	将矢量的内容与vector2的内容交换。
	示例：vect.pop_back();//移除vect的最后一个元素，使其大小减去1		示例：vect1.swap(vect2);//交换vect1和vect2的内容

谢谢

亢孟军 武汉大学
mengjunk@whu.edu.cn