C++ Basic Course

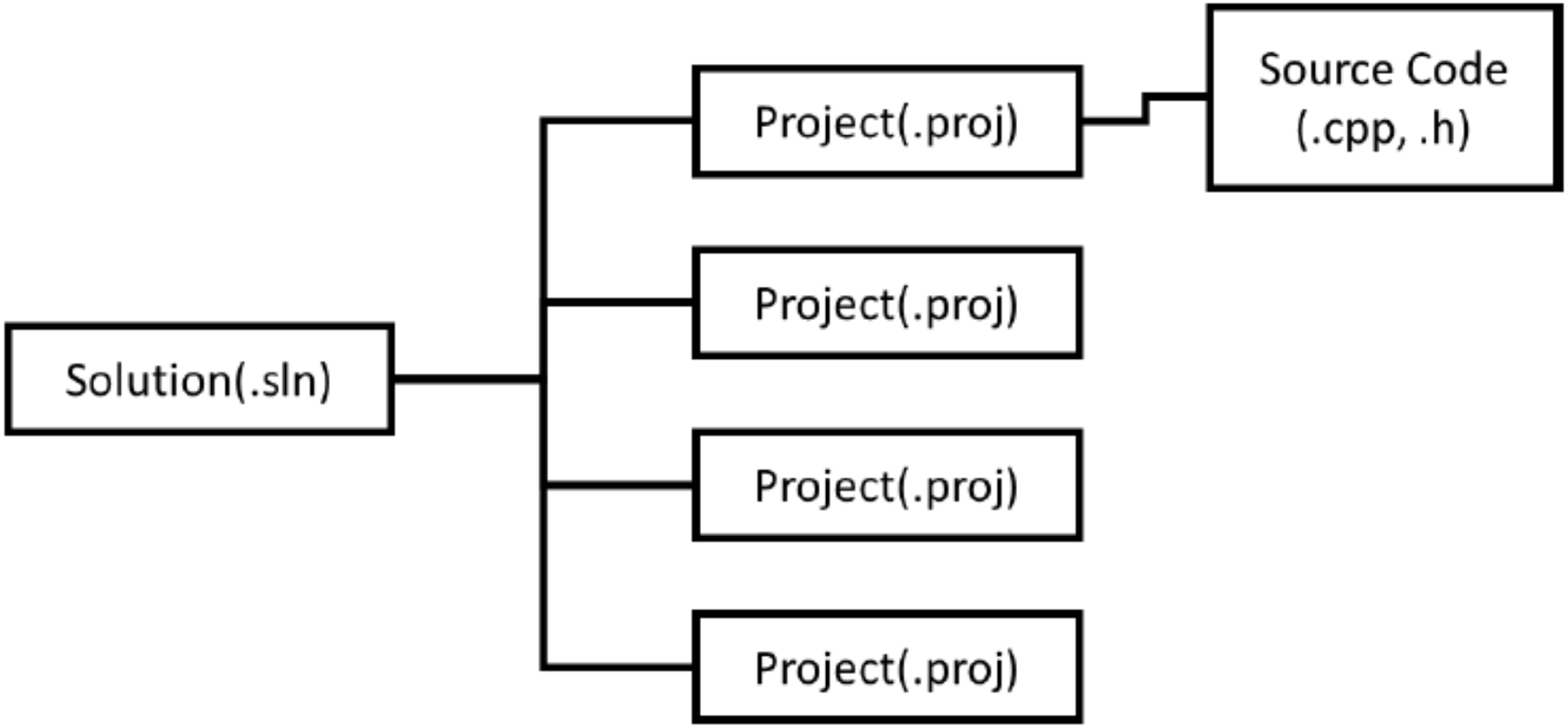
Part I: The Basics (Ch. 2 – Ch. 7)

群杜工作室

Last Time

- How to establish Win32 empty Project?
- Lab 1

Solution and Project in VS2017



See Program in Lab1

Lab 1

- 2nd sample run in Prob1.
- #include <random>
- Space

```
for (int celsius = floor; celsius != ceil; ++celsius)
{
    cout << celsius << "\t" << celsius * (9. / 5.) + 32 << endl;
}
```

See detail in Lab1

Part I: The Basics (Ch. 2 – Ch. 7)

- The basics of the C++ language: you will definitely need to master these features.
- We will do highlights/exercises to “jump” you through this part.
- You should read this part at least once and use chapter summary to check your understanding.

Reader Guide: Book Icons

- You should read and understand these sections.

- You can skip them for the first read.

- Tricky part; it takes time to understand but is essential to language.

Chapter 2: Variables (Objects) and Basic Types

- Primitive Built-in Types
- Variables
- **Compound Types**
- Const Qualifier
- Dealing with Types
- Defining our Own Data Structures

Primitive Built-in Types: Arithmetic Types

- Integral Types
  - Integers: short, int, long, long long --- signed, unsigned
  - Characters: char --- signed, unsigned
  - Extended Characters: wchar_t, char16_t, char32_t
  - Boolean values: bool
  - The unsigned int can be abbreviated as unsigned
- Floating-Point Types
  - float, double, long double

Variables (Objects)

- A variable provides us with named storage that our programs can manipulate.
- Each variable in C++ has a type. The type determines
  - the size and layout of the variable's memory
  - the range of values that can be stored within that memory
  - the set of operations that can be applied to the variable.
- C++ programmers tend to refer to variables as "variables" or as "objects" interchangeably

Compound Types

- A compound type is a type defined in terms of another type. We cover reference and pointer here.
- Reference
  - A reference is an alternative name for an object (e.g., 孫文 and 孫中山). An object declared as a reference is merely a second name (alias) assigned to an existing object. No new object is created.
  - A reference is defined by preceding a variable name by the & symbol.

See Note

Compound Type: Pointer

- A **pointer** is a compound type that "points to" another type.
- Conceptually, pointers are simple: a pointer holds the address of another object.

```
string  s("hello world");
string *sp = &s;  //sp holds the address of s
```

define sp as a pointer to string

```
string *sp = &s;
```

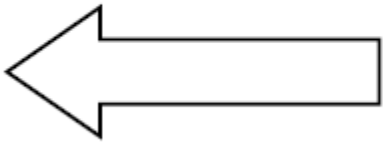
initialize sp to point to the string named s;

Defining and Initializing Pointer

- We use * operator symbol in a declaration to indicate that an identifier is a pointer.
- 

When attempting to understand pointer declarations, read them from right to left.

```
string *pstring;
```



? read them from right to left.

Reading from right to left: pstring is a pointer that can point to string objects.

Pointer Operation

- We use * operator (the dereference operator) to access the object to which the pointer points.

```
string s("hello world");
string *sp = &s; //sp holds the address of s
cout << *sp;
```

```
int ival = 1024;
int *pi = &ival; // pi points to an int
int **ppi = &pi; // ppi points to a pointer to int
cout << "The value of ival\n"
     << "direct value: " << ival << "\n"
     << "indirect value: " << *pi << "\n"
     << "doubly indirect value: " << **ppi
     << endl;
```

See Note

const Qualifier

- The const qualifier provides a way to transform an object into a constant.
  - Avoid magic number in the code.
  - Define constants such as PI.
- We must initialize it when it is defined (why?).

```
const double PI = 3.14159265358979323846264338327 95;
```

const Qualifier

- Objects declared with const-qualified types may be placed in read-only memory by the compiler, and if the address of a const object is never taken in a program, it may not be stored at all.

```
const int n = 1; // object of const type
n = 2; // error: the type of n is const-qualified
int x = 2; // object of unqualified type
const int* p = &x; *p = 3; // error: the type of the lvalue *p is const-qualified
```

auto Type Specifier

C++ 11

- We can let the compiler figure out the type for us by using the auto type specifier.
- Unlike typical type specifiers, such as double, that name a specific type, auto tells the compiler to deduce the type from the initializer.

```
auto i = 0;

vector<int> v;
vector<int>::iterator p = v.begin();
auto p = v.begin();
```

See Note

decltype Type Specifier

- auto tells the compiler to deduce the type from the initializer.
- decltype tells the compiler to deduce type from an expression. The compiler analyzes the expression to determine its type but does not evaluate the expression.

See Note

struct and class: the way to define your own type

(the basic of struct, see note)

Header File (.h)

- We use #include (a C++ preprocessor) when using a header.

```
#include <iostream>
#include "Sales_item.h"
```

(what is difference between <> and “”?).

- When we write our own header, we do header guards (see note).

```
#ifndef SALESITEM_H
#define SALESITEM_H
...
#endif
```

Until Next Time

- Lab 2
- HW1
- [Reading] Chapter 3 (Your first exposure to the powerful library type).