Module03-01
C++ 语言基础：类型与声明

- 楔子
- → 语言基础
- 面向对象编程
- 泛型编程

语言基础

- 这部分的课程我们将熟悉 C++ 语言的基础部分：
  - 类型与声明
  - 指针、数组、结构
  - 表达式
  - 语句
  - 函数
  - 名字空间
  - 源文件与程序

语言基础 - 类型

- C++ 数据类型
  - 基本类型
    - Booleans：如 `bool`
    - Characters：如 `char`
    - Integers：如 `int`、 `short`、 `long`
    - Floats：如 `double`、 `float`
  - 枚举类型 (Enumerations)
  - void 类型
  - 由基本类型或枚举类型组成的类型
    - Arrays，如 `double[]`
    - Pointers，如 `int*`
    - References：如 `Employee&` 、 `int&`
    - Structures、 Classes

类型 - Booleans

- Boolean
  - 类型： bool
  - 一个 bool 类型变量的值为： true 或 false
  - bool 类型通常用于表示逻辑运算的结果
  - 示例：

```
bool b = true;

bool result = (a > b);

bool isEquals(const int& n, const int& m) {
    return n == m;
}
```

类型 - Characters

- Character
  - 类型：
    - char(signed char)：单字节有符号字符
    - unsigned char：单字节无符号字符
    - wchar_t：双字节字符
  - 示例：

```
char c = 'V';
c = '\n';
wchar_t wc = L'W';
```

类型 - Integers

- Integer
  - 类型：
    - 有符号整型
      - int(signed int)：int
      - short int(signed short int)：short
      - long int(signed long int)：long
    - 无符号整型
      - unsigned int：unsigned
      - unsigned short int：unsigned short
      - unsigned long int：unsigned long
  - 整型变量在内存中的所占的字节数（bytes）由具体的实现（编译器）而定

类型 - Floats

- Float
  - 类型：
    - `float`：单精度浮点数
    - `double`：双精度浮点数
    - `long double`：扩展精度浮点数
  - 浮点数极值范围具体由不同的实现（编译器）而定

类型 - 基本类型的文字常量

- 文字常量（ Literals ）
  - 示例：

```
1,Booleans
true  false

2,Characters
'G'   'k'
L'F'     // 宽字符, wchar_t
'\t'     // 转义字符, 表示水平制表符

3,Integers
25  0x19  031 // int, 分别以十进制、十六进制、八进制表示整数25
25U  25u      // unsigned int, 无符号整数25
25L  25l      // long int, 尽量使用大写L, 而不是小写l, 易跟数字1混淆
25UL 25ul 25Lu  // unsigned long int, 25

4,Floats
16.  2.08  .23  3.0d  8d  5.26E7 3.298e-3  // double
12f  12.F  3.48e6f  4.9087e-2f  //float
1.2L  23.098e-8L     // long double
```

类型 - 基本类型的文字常量

- 文字常量（ Literals ）（续）
  - 文字常量的默认类型（未指定 U 、 L 、 F 等后缀）
    - 整型文字常量默认类型为 int ， 如： 12 表示 int 型的 12
    - 浮点型文字常量默认类型为 double ， 如： 1.2 为 double 型
  - 常用的转义字符：

描述	字符
newline(换行符)	\n
horizontal tab(水平制表键)	\t
vertical tab(垂直制表键)	\v
backspace(退格键)	\b
carriage return （回车键）	\r
formfeed （进纸键）	\f
alert (beel) （响铃符）	\a
backslash （反斜杠键）	\\
question mark （问号）	\?
single quote （单引号）	\'
double quote （双引号）	\"
null character （ 0 字符 ）	\0

类型 - 基本类型的 size

- 类型的 size
  - C++ 语言中各种类型在内存中所占的 byte 数由具体实现（编译器）决定
  - sizeof 表达式：可以通过 sizeof 表达式查询（任意）类型或对象的 size
  - 示例：

```
int n = 0;
cout << sizeof(n) << endl;        // 4 in my 32-bits system
cout << sizeof(long) << endl;     // 4 in my 32-bits system
cout << sizeof(double) << endl; // 8 in my 32-bits system
```

类型 - 基本类型的极值

- 基本类型的最大值和最小值
  - 查看最大值: numeric_limits<Type>::max()
  - 查看最小值: numeric_limits<Type>::min()
  - 示例:

```
#include <iostream>
#include <limits>   // 使用 numeric_limits 必须包含此头文件

using namespace std;

int main() {
    cout << "largest int: " << numeric_limits<int>::max()
         << endl;
}
```

如果超过某种类型的最大值（上溢）或小于最小值（下溢），会出现什么情况？

类型 - 浮点数二进制表示

- 浮点数的二进制表示：
  - S： 符号位， 0- 正数， 1- 负数
  - E： 指数位 (exponent)
  - F： 小数位或称尾数位 (fraction, mantissa)

```
# 单精度浮点数 (float)：指数 8 bits，尾数 23 bits
S EEEEEEEE FFFFFFFFFFFFFFFFFFFFFFF
0 1       8 9                     31

# 双精度浮点数 (double)：指数 11 bits，尾数 52 bits
S EEEEEEEEEEE FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
0 1          11 12                                                63
```

类型 - 浮点数二进制表示

- 浮点数的二进制表示（续）：

```
# 示例：单精度浮点数（float）二进制 --> 十进制换算，^ 表示幂
0 00000000 00000000000000000000000 = 0
1 00000000 00000000000000000000000 = -0

0 11111111 00000000000000000000000 = Infinity
1 11111111 00000000000000000000000 = -Infinity

0 11111111 00000100000000000000000 = NaN
1 11111111 00100010001001010101010 = NaN

0 10000000 00000000000000000000000 = +1 * 2^(128-127) * 1.0 = 2
0 10000001 10100000000000000000000 = +1 * 2^(129-127) * 1.101 = 6.5
1 10000001 10100000000000000000000 = -1 * 2^(129-127) * 1.101 = -6.5

0 00000001 00000000000000000000000 = +1 * 2^(1-127) * 1.0 = 2^(-126)
0 00000000 10000000000000000000000 = +1 * 2^(-126) * 0.1 = 2^(-127)
0 00000000 00000000000000000000001 = +1 * 2^(-126) *
                                     0.00000000000000000000001 =
                                     2^(-149) （一个极小的浮点数）
```

类型 - void

- void
  - 并没有 void 类型的对象， void 通常用于函数的返回值，表示无返回值

类型 - Enumerations

- Enumerations(枚举)
  - 关于枚举类型
    - 枚举是一种用户定义类型，用于列举一类相关的值，如我们可以用 red 、 green 、 blue 作为三元色的枚举
    - 枚举是一种具有整型值的常量
  - 枚举类型的定义：

```
// 匿名枚举类型
enum { A, B };
// 默认: RED==0, GREEN=1, BLUE=2
enum Color { RED, GREEN, BLUE };
// 指定枚举项的值
enum Level { LOW = -2, MIDDLE = 0, HIGH = 2 };

int main() {
    // 使用枚举类型
    Color color = BLUE;
    cout << A << endl;
}
```

类型 - Enumerations

- Enumerations(枚举) (续)
  - 枚举项的值:
    - 如果枚举项均未显式指定值，则第一个枚举项值为 0 ，后面依次加 1
    - 可以为枚举项显式指定值，且不一定须连续或唯一
  - 枚举类型的取值范围:
    - 最小枚举项值不小于 0 ，范围为 0 到最大枚举项的值接近的 2^n-1
    - 最小枚举项值小于 0 ，范围为 -(最大枚举项的值接近的 2^n) 到最大枚举项的值接近的 2^n-1

```
enum E1 { EO1 = 1, EO2 = 12 }; // range 0~15
enum E2 { H = 176, L = -4 }; // range -256~255
```

语言基础 - 声明

- 声明
  - 声明与定义的概念
  - 声明语句的结构
  - 对象的储存方式
  - 名字的规定
  - 名字的作用域
  - 对象的初始化
  - 对象的生命周期
  - 类型的别名 (typedef)

声明与定义 - 声明与定义的概念

- 声明 (Declarations)
  - 声明就是告诉编译器一个名字（标识符）（name、identifier）所关联的实体是什么，如 n 是一个 int 型对象？一个类名？一个类对象？一个函数名？...
  - 一个名字在使用之前，必须先声明它
  - 一个名字可以重复声明
- 定义 (Definitions)
  - 为声明的名字定义如类体、函数体、对象的存储方式、值等等
  - 在一个程序（或库）中，相同作用域下的一个名字（name、identifier）只能被定义一次，即一次定义原则（ODR）
  - 对于任意类型的对象而言，声明即定义

声明 - 声明语句的结构

- 声明语句的组成
  - 可选的 限定符 (specifier)：如 extern 、 static 等
  - 必须的 类型：如 int 、 Employee 等
  - 必须的 名字 (name 或 identifier)： age 等
  - 可选的 初始化表达式：如 int a = 0;

```
extern long seconds;
class Employee;
void func(const Color&);
bool b = true;
static double d = 1.414;
```

声明 - 对象的存储方式

- 五种存储方式（由以下 5 个 specifier 指定）
  - auto：自动对象，如局部变量、函数实参，其生命周期仅限于它所声明的作用域，该限定符一般不需指定
  - extern：表示该对象不再当前文件中定义
  - mutable：表示该对象即使是某个 const 对象的成员变量，也可以被改变
  - register：提示编译器将该对象存放在寄存器中（一般会被编译器忽略）
  - static：静态对象，生命周期延续至析构函数被调用或程序结束

```
extern long seconds;
static Employee emp;
mutable int n;
```

声明 - 名字的规定

- 声明的名字
  - 合法的名字（ name 或 identifier ）
    - 组成名字的字符：[a-zA-Z0-9_]，也即 [[:word:]]
    - 名字首字必须为字母或下划线：[a-zA-Z_]
    - 不可与 C++ 关键字相同
    - 双下划线（__）开头的名字通常是 C++ 实现（编译器）使用名字，尽量避免这种方式的命名

声明 - C++ 关键字

- C++ 关键字 (74 个)

基本类型相关	bool char wchar_t int long short float double signed unsigned void true false
自定义类型相关	enum struct union class this virtual private protected public explicit friend typeid operator
对象存储方式	auto extern mutable register static
对象修饰符	const volatile
类型转换	const_cast dynamic_cast reinterpret_cast static_cast
语句相关	if else switch case do while for break continue default goto try catch throw return
函数	inline
操作符	and and_eq or or_eq xor xor_eq not not_eq bitand bitor compl(~, bitwise complement) sizeof new delete
模板	template typename export
其它	using namespace typedef asm

声明 - 名字的作用域

- 作用域 (scope)
  - 如果一个名字声明在所有函数、名字空间、类外，则是一个全局 (global) 名字，否则为局部 (local) 名字
  - 内部名字可以覆盖其作用域外的名字
  - 通过域操作符 :: 可以取得全局对象

```
int n;  // #1, global
int main() {
    int n = 9; // #2, main() scope
    {
        int n = 8;  // #3, smaller scope
        cout << ::n << endl; // get #1
        cout << n << endl;  // get #3
    }
    cout << n << endl;  // get #2
}
```

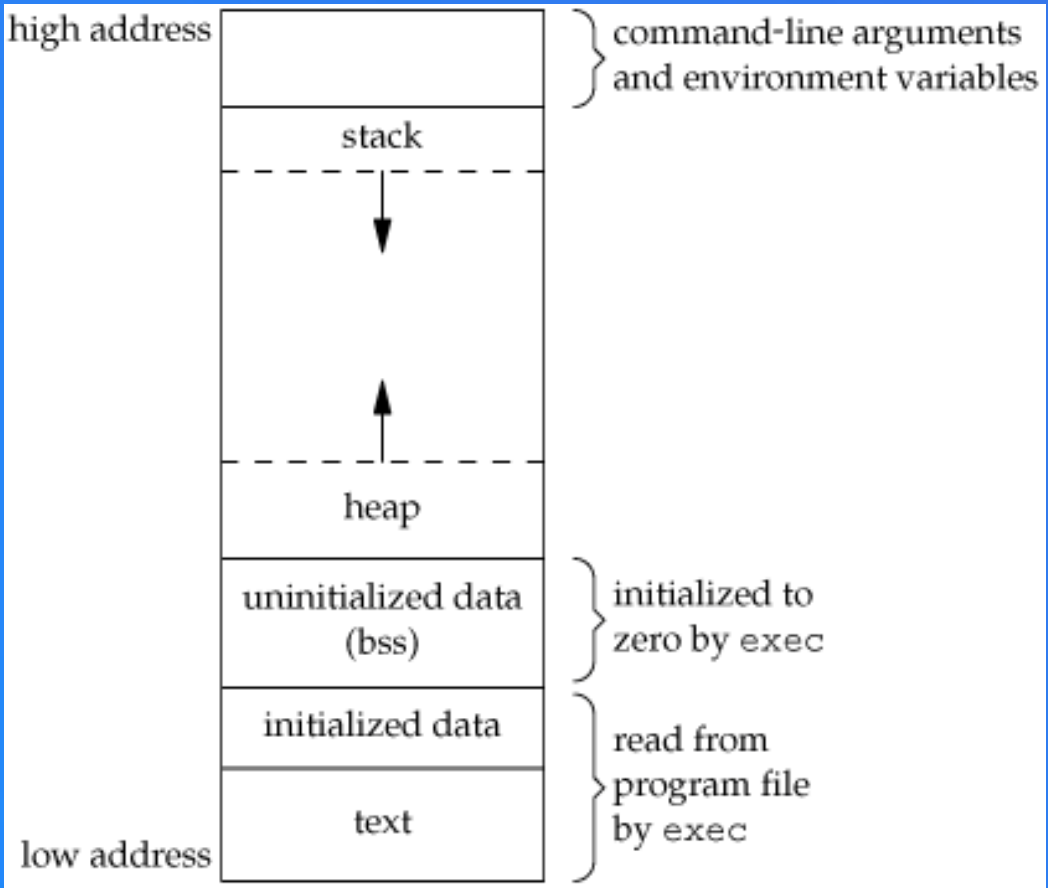
声明 - 对象的初始化

- 对象的初始化
  - 如果没有明确指定初始式，属于全局、名字空间、静态的对象初始化其类型对应的 0 值
  - 2 种初始化形式：
    - 赋值形式的初始化：通常为复制性初始化，如： Type t = ot;
    - 函数形式的初始化：通常为直接初始化，如： Type t(7);
  - 示例：

```
int* p; // int* p = 0;
double d; // double d = 0.0;
int main() {
    int x = { 42 }; // 标量(单个)对象(scalar object)初始化
    int m; // m没有被初始化
    bool b = true; // 赋值形式的初始化
    int n(8); // 函数形式的初始化
    Employee e(32, "Hill"); // 函数形式的初始化（构造函数）
}
```

声明 - 对象的生命周期

- 对象的生命周期
  - Linux C++ 程序可能的内存布局：
    text 、 data 、 bss 、 stack 、 heap



声明 - 对象的生命周期

- 对象的生命周期（续）
  - 对象的生命周期由分配储存空间、初始化开始，到析构函数被成功调用或其拥有的内存被释放结束
  - 三种类别的生命周期：
    - 静态对象：局部静态对象是在程序运行到该对象声明处初始化、全局静态对象是程序运行后在 main 函数执行前被初始化，两者都是在其析构函数被正确调用或程序结束而销毁
    - 自动对象：也叫栈对象，如局部变量、函数实参，于程序运行到该对象声明处初始化，离开其所在的作用域自动销毁
    - 动态对象：也叫堆对象，由 new 表达式创建，由 delete 表达式删除

声明 - typedef

- 类型别名 typedef
  - typedef 不会创建一种新类型，只是为一种类型加个别名

```
typedef unsigned long int ulong;
typedef char byte;
typedef double (*getRate) ();
```

类型与声明 - Bjarne's Advices

- Bjarne's Advices
  - 保持小的作用域（一个名字只有在使用时、使用处才引入）
  - 不要在一个作用域以及其内的作用域中声明相同的名字
  - 一个声明语句声明一个名字
  - 声明一个对象的同时，作恰当的初始化 (Tiger's Advices)
  - 常用的、局部的名字尽量短小，非局部、不常用的名字尽量完整
  - 避免相似的命名
  - 保持一致的命名风格
  - 尽量使用 int，而不是 short 或 long
  - 尽量使用 char，而不是 signed char、 unsigned char
  - 尽量使用 double，而不是 float、 long double