Module03-02
C++ 语言基础：数组、指针、结构

语言基础 - 数组、指针、结构

- 数组、指针、结构
  - 数组 (Arrays)
  - 指针 (Pointers)
  - 字符串文字常量 (String Literals)
  - 指针与数组的关系
  - 引用 (References)
  - void*
  - 结构 (Structures)

- 数组 (Array)
  - 数组是一种容纳相同类型对象的容器
  - 可以通过 2 种方式访问数组中的元素：
    - 下标操作符 [ ]：如 a[0]，表示数组 a 的第一个元素。注意：假设数组 a 的元素个数为 n，则数组有效下标（index）从 0 开始，到 n-1 结束
    - 数组名作为指针：数组的名可以隐式的转化为指针，指向数组的第一个元素（数组名所代表的指针为常量指针： T* const ）

```
int a[] = { 1, 2, 3, 5, 8 };      // 5个元素
cout << a[2] << endl;     //通过下标操作，输出第3个元素的值
cout << *(a+1) << endl; //通过数组名（指针），输出第2个元素的值
```

  - 多维数组

```
double da[2][3];      // 一个拥有2个数组、每个数组中有3个元素的数组
```

数组 - 数组的初始化

- 数组 (Array)
  - 数组的维数必须在编译期指定

```
int na[] = { 1, 2, 4, 8 }; // 4个元素
int nb[5] = { 1, 2, 3 }; // 余下的2个元素被初始化为0: 1,2,3,0,0
int nc[3] = { 1, 2, 3, 4 }; // 错误，nc最多只有3个元素

int nd[3];
nd[3] = { 1, 2, 3 };    // 语法错误，不存在这种格式的赋值
nd = { 1, 2, 3 };       // 语法错误，不存在这种格式的赋值

int ma[][2] = { { 1, 2 }, { 3, 4 } }; // 2x2多维数组
int mc[][] = { { 1, 2 }, { 3, 4 } }; // 错误，多维数组只有第一个维数可以为空
```

数组、指针、结构 - 指针

- 指针 (Pointer)
  - 指针是一种保存其它对象地址的对象，通过指针可以间接操纵所指向的对象
  - 指针对象的产生
    - 通过取址操作符 (&) 获取其它对象的地址
    - 通过 new 操作符获取动态分配的对象地址
    - 通过其它指针对象赋值
    - 指向一个数组
  - 获取指针指向的对象
    - 通过“去引用”操作: *p
  - 关于空指针 (null pointer)
    - int* p = 0; // 表示 p 不指向任何对象

指针 - 示例

- 指针 (Pointer)
  - 示例：

```
int n = 18;
int* p1 = &n;          // p1指向n
*p1 += 5;              // n += 5;
cout << n << endl;     // 23
int* p2 = p1;          // 现在p2,p1同时指向n
cout << "&n: " << &n << ", p1: " << p1
     << ", p2: " << p2 << endl;
*p2 += 8;              // n += 8;
cout << n << endl;     // 31
int m = 0;
p1 = &m;               // 现在p1指向m，不再指向n
double da[] = { 1., 2.3, 89.02 };
double* dp = da;       // 指向数组da的第一个元素
while (dp < da + 3)
    cout << *dp++ << ' ';
cout << endl;
Employee* emp = new Employee; // emp指向堆空间一个Employee对象
emp->setSalary(8200.0); //通过指针访问堆空间中的Employee对象
```

- 字符串文字常量
  - 字符串是一种以 '\0' 结尾的文字常量 (Literals)
  - 可以用字符串初始化 char[] 或 char*
  - 示例：

```
char ca[] = "Tiger"; // 数组ca元素个数6，5个字符和字符串结尾的'\0'
ca[2] = 'm'; // OK，现在ca的内容"Timer\0"

cout << sizeof("tiger") << endl; // 6

char* cb = "Tiger"; // 不推荐的方式
cb[2] = 'm'; // 运行期错误
const char* cc = "Tiger"; // 推荐的方式
cc[2] = 'm'; // 编译期错误

const char* cat = "tiger's"
    " home";    // OK, 最终被编译器合并为"tiger's home"
```

数组、指针、结构 - 指针与数组的关系

- 指针与数组的关系
  - 一个数组的名字可以隐式的转化为指针，指向其第一个元素
  - 不能将一个指针赋值给一个数组名

数组、指针、结构 - 引用

- 引用 (Reference)
  - 引用就是一个对象的别名 (alias)，通过引用可以间接的操纵其引用的对象（这点很像指针的行为）
  - 引用多用于函数的参数传递（后续的课程中介绍，引用对于 C++ 程序的性能有非常大的影响）
  - 与指针的区别：
    - 定义引用必须同时初始化，而指针可以先声明，再赋值
    - 引用一旦定义就不可改变所引用的对象，指针可以重新指向别的对象

引用 - 示例

- 引用 (Reference)
  - 示例：

```
int n = 9;
int& rn = n; // n的引用
rn += 5; //n += 5;
int m = 0;
rn& = m;     // 语法错误，不存在引用转向
rn = m; // 可以，n = m;

int& ra = 1; // 错误，不存在对临时对象的引用
const int& rb = 1; //OK，const引用
```

数组、指针、结构 - void*

- void* ：万能指针
  - 任意类型的指针都可以赋值给 void*
  - void* 不能直接进行“去引用”操作
  - 对 void* 操作：
    - 将任意类型的指针赋值给 void*
    - 将一个 void* 对象赋值给另一个 void* 对象
    - 显式的将 void* 转换成其它类型的指针
    - 比较两个 void* 对象是否相等
    - 除上述操作之外的操作均不安全或不允许

void* - 示例

- void*：万能指针
  - 示例：

```
int n = 0;
int* p1 = &n;
void* vp1 = p1;
*vp1 += 5; // 不能对void*执行"去引用"操作
++vp1;  // 操作不允许

int* p2 = static_cast<int*>(vp1); // 显式转换成int*
*p2 += 5;
```

数组、指针、结构 - 结构

- 结构 (Structures)
  - 结构是（几乎是）任意类型的元素的聚集
  - C++ 中的 struct 是 class 的另一种称谓
  - 结构类型的定义，如下面的语句块定义了一种新类型，名为 Employee

```
struct Employee {
    string employeeId;
    string name;
    string department;
    string email;
    int salary;
};
```

结构 - 对象初始化、成员的访问

- 结构对象的初始化
  - 成员的初始化顺序严格按照其在定义时出现的顺序；
  - 类似于数组的初始化

```
Employee emp1;  // 创建一个Employee对象，未初始化
Employee emp2 = { "DN0012", "Tiger", "RND",
                  "sample@mmm.com", 8900 };    // 创建对象并初始化
```

- 结构内元素（成员）的访问

```
// 直接访问成员
cout << emp1.name << endl;
emp1.salary = 12000;

// 对象指针访问成员
Employee* emp3 = &emp2;
emp3->salary = 12800;  // 相当于(*emp3).salary
cout << emp3->department << endl;
```

结构 - 自定义类型的 size

- 用 struct 定义的类型的 size
  - 所有成员的 size 之和，还有：
  - 内存空间中对齐的问题（一般是自然机器字 (word) 的倍数）

结构 - 关于 struct 前缀

- 关于 struct 前缀
  - 在 C 中，声明 struct 对象需加 struct 前缀，而 C++ 中则无必要，如：

```
struct Employee {
    string employeeId;
    string name;
    string department;
    string email;
    int salary;
};
// 下列情况在C++中已不必要，加上前缀也无影响
void print(struct Employee*);
struct Empoyee emp;

// 更多时候是用于避免名字混淆，这点 class、enum等与struct规则一致
struct aname {/* */};
void aname(int*); // 这里aname是函数
// 下面的struct前缀是必要的，可以避免名字混淆
struct aname obj;  // 创建一个aname类型的对象，跟函数aname无关
```

指针、数组、结构 - Bjarne's Advices

- Bjarne's Advices
  - 注意不要越界对数组进行写操作
  - 使用 0 初始化空指针，而不是 NULL(一个宏)
  - 尽量使用 vector 或 valarray ， 而不是数组
  - 尽量使用 C++ 的 string ， 而不是以 '\0' 结尾的字符数组
  - 尽量少使用 T& 类型的函数参数（使用 const T& ）
  - 除在低阶代码中使用外，避免使用 void*
  - 将文字常量定义成 const 对象来使用