7 函数-C++的编程模块

函数参数和按值传递

- 函数被调用时，该函数将创建一个新的名为x的double变量，并将其初始化为5
  - 这样， cube()执行的操作将不会影响main()中的数据，因为cube()使用的是side的副本，而不是原来的数据
- 在函数中声明的变量（包括参数）为该函数私有
  - 在函数被调用时，计算机将为这些变量分配内存；在函数结束时，计算机将释放这些变量使用的内存（有些文献将分配和释放内存称为创建和销毁变量）。这样的变量被称为局部变量，因为它们被限制在函数中。

```
 ...
double cube(double x);
int main()
{

   ...
   double side = 5;
   double volume = cube(side);
   ...
}

double cube(double x)
{
 return x * x * x;
}
```

creates variable called side and assigns it the value 5 → 5 (side) — original value

passes the value 5 to the cube( ) function

creates variable called x and assigns it passed value 5 → 5 (x) — copied value

```
 ...
void cheers(int n);
int main()
{
   int n = 20;
   int i = 1000;
   int y = 10;
   ...
   cheers(y);
   ...
}

void cheers(int n)
{
   for (int i = 0; i <n; i++)
      cout << "Cheers!";
   cout << "\n";
}
```

Each function has its own variables with their own values.

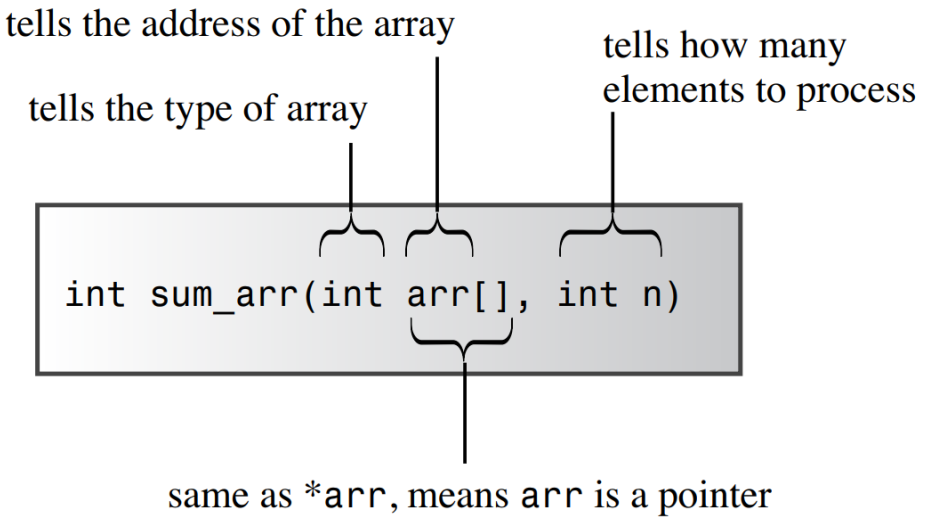
20 1000 10 — n i y — variables in main()

10 0 — n i — variables in cheers()

3 函数和数组

P7.5 arrfun1.cpp

- 定义
  - `int sum_arr(int arr[], int n);`
  - `int sum_arr(int *arr, int n);`
- C++将数组名解释为其第一个元素的地址
  - 首先，数组声明使用数组名来标记存储位置
  - 其次，对数组名使用s'izeof 将得到整个数组的长度（以字节为单位）
  - 最后，将地址运算符&用于数组名时，将返回整个数组的地址
- 当（且仅当）用于函数头或函数原型中，`int *arr`和`int arr[ ]`的含义才是相同的



函数和数组

- 将数组地址作为参数可以节省复制整个数组所需的时间和内存(P7.7 arrfun3.cpp)
  - 如果数组很大，使用拷贝的系统开销将非常大；程序不仅需要更多的计算机内存，还需要花费时间来复制大块的数据

- 另一方面，使用原始数据增加了破坏数据的风险
  - ANSIC 和C++中的const 限定符提供了解决这种问题的办法

3.5 指针和const

- 代码
  - `int age = 39;`
  - `const int *pt = &age;`
- 意味着
  - 不能通过pt修改age!

- 如形参数组的值不能被改变，需要
  - `void show_array(const double arr[], int n);`
  - `void show_array(const double *arr, int);`

```
int gorp = 16;
int chips = 12;
const int * p_snack = &gorp;

*p_snack = 20;          p_snack = &chips;
```

NO　　OK

disallows changing value to which p_snack points

p_snack can point to another variable

```
int gorp = 16;
int chips = 12;
int * const  p_snack = &gorp;

*p_snack = 20;          p_snack = &chips;
```

OK　　NO

p_snack can be used to change value

disallows changing variable to which p_snack points

4 函数和二维数组

```
1. int sum(int ar2[][4], int size)
2. {
3.     int total = 0;
4.     for (int r = 0; r < size; r++)
5.         for (int c = 0; c < 4; c++)
6.             total += ar2[r][c];
7.     return total;
8. }
```

5 函数和C-风格字符串

- `unsigned int c_in_str(const char * str, char ch);`
  - 不用传长度，通过遍历指针和'\0'比较
  - P7.9 strgfun.cpp
- `char * buildstr(char c, int n);`
  - 在函数中new
  - P7.10 strgback.cpp

6 函数和结构

- `travel_time sum(travel_time t1, travel_time t2)`
  - 不修改实参的值
  - P7.11 travel.cpp
- `void rect_to_polar(const rect * pxy, polar * pda)`
  - const *, 不修改;
  - *可修改
  - P7.13 strctptr.cpp
- `void TestArry(rect *rts, int n); //可修改rts`
  - 数组方式
  - `void TestArry(const rect *rts, int n); //不修改`

7 函数和string对象

- 可以直接当成一个独立的数据结构用

P7.14 tipfive.cpp

```
1.  #include <string>
2.  using namespace std;
3.  const int SIZE = 5;
4.  void display(const string sa[], int n);
5.  int main(){
6.      string list[SIZE];
7.      for (int i = 0; i < SIZE; i++){
8.          cout << i + 1 << ": ";
9.          getline(cin,list[i]);
10.     }
11.     display(list, SIZE);
        return 0;
12. }

13. void display(const string sa[], int n){
14.     for (int i = 0; i < n; i++)
15.         cout << i + 1 << ": " << sa[i] << endl;
16. }
```

10 函数指针

- 函数名就是指向函数的指针

```
process(think);   // passes address of think() to process()
thought(think()); // passes return value of think() to thought()
```

- 声明函数指针
  - 原型：double pam(int);
  - 指针： double (*pf)(int);
  - 赋值：pf = pam;
  - 使用：(*pf)(5)或者pf(5)
- P7.18 fun_ptr.cpp

```
typedef double (p_fun)(double); // p_fun now a type name
p_fun p1 = f1;                  // p1 points to the f1() function
```