7.12 复习题

1. 使用函数的3个步骤

- 定义函数
- 调用函数
- 声明函数/函数原型

2.

2．请创建与下面的描述匹配的函数原型。
a．igor( )没有参数，且没有返回值。
b．tofu( )接受一个 int 参数，并返回一个 float。
c．mpg( )接受两个 double 参数，并返回一个 double。
d．summation( )将 long 数组名和数组长度作为参数，并返回一个 long 值。
e．doctor( )接受一个字符串参数（不能修改该字符串），并返回一个 double 值。
f．ofcourse( )将 boss 结构作为参数，不返回值。
g．plot( )将 map 结构的指针作为参数，并返回一个字符串。

```
// a
void igor(void);

// b
float tofu(int);

// c
double ,pg(double, double);

// d
long summation(long[], int);

// e
double doctor(const char *str);

// f
struct boss {};
void ofcourse(boss);

// g
struct map {};
char* plot(map *pt ); // map结构体的指针作为参数
```

3.

3．编写一个接受 3 个参数的函数：int 数组名、数组长度和一个 int 值，并将数组的所有元素都设置为该 int 值。

```
int tempFunc(int arr[], int size, int n)
{
    for (int i = 0; i < size; i++){
        arr[i] = n;
    }
}
```

题目4-10

4．编写一个接受 3 个参数的函数：指向数组区间中第一个元素的指针、指向数组区间最后一个元素后面的指针以及一个 int 值，并将数组中的每个元素都设置为该 int 值。

5．编写将 double 数组名和数组长度作为参数，并返回该数组中最大值的函数。该函数不应修改数组的内容。

6．为什么不对类型为基本类型的函数参数使用 const 限定符？

7．C++程序可使用哪 3 种 C-风格字符串格式？

8．编写一个函数，其原型如下：

```
int replace(char * str, char c1, char c2);
```

该函数将字符串中所有的 c1 都替换为 c2，并返回替换次数。

9．表达式*"pizza"的含义是什么？"taco" [2]呢？

10．C++允许按值传递结构，也允许传递结构的地址。如果 glitz 是一个结构变量，如何按值传递它？如何传递它的地址？这两种方法有何利弊？

4.

```
void set_arr(int *begin, int *end, int value)
{
    for (int *pt = begin; pt != end; pt++){
        (*pt) = value;
    }
}
```

5.

```
double findMax(const double *arr, int size)
{
    double max_val = -100000;
    for (int i = 0; i < size; i++){
        if (*(arr+i) > max_val){
            max_val = *(arr+i);
        }
    }

    return max_val;
}
```

6.

因为基本数据类型在函数调用的时候是按值传递的, 此时函数里使用的是参数的副本, 因此加不加`const`都不会影响原来的参数, 所以就可以不使用const关键字.

7.

- C++ 中, C风格的字符串可以是:
  - char 字符数组 `char str[] = "Hello World"`,
  - 直接使用双引号表示字符串: `"Hello World"`.
  - 首字符的指针: `char *pt = "Hello World"`

8.

```
int replace(char * str, char c1, char c2)
{
    int count = 0;
    char *pt = str;

    while ( *pt != '\0'){            // 遍历到最后一个元素为止
        if ( *pt == c1){
            *pt = c2;
            count++;
        }
        pt++;                        // 别忘记移动指针
    }

    return count;
}
```

9.

- `*"Pizza"`: "Pizza" 是一个字符串, 它本身也代表着**第一个元素** "P" 的地址, 所以*"Pizza" 表达式会返回字符 'P'.
- `"taco"[2]`: 同理, "taco" 是一个字符串, 它本身也是这个字符串首字符的地址, 后面索引[2]表示的是第三个元素, 所以这个表达式会返回字符 'c'.

10.

- 传递结构的值:

```
struct glitz{};                  // 假设有一个结构体 glitz
void tempFunc(glitz temp);       // 按值传递
tempFunc(glitz);                 // 调用函数的时候, 传递结构体的值
```

- 按地址传递:

```
struct glitz{};                     // 假设有一个结构体 glitz
void tempFunc(glitz *temp);         // 按地址传递
tempFunc(&glitz);                   // 调用函数的时候，传递结构体的地址
```

它们的差异是:

- 按值传递的话, 在函数内部对结构体的修改不会影响到原来的结构体,
  - 但是会执行一次拷贝的过程, 如果结构体的值占用较大的内存空间, 比如图片等数据, 此时就会产生较大的资源开销.
- 传地址不会产生拷贝, 但是有可能会修改原来的结构体, 所以需要注意是否需要使用const关键字做保护.

11.

11. 函数 judge( )的返回类型为 int，它将这样一个函数的地址作为参数：将 const char 指针作为参数，

并返回一个 int 值。请编写 judge( )函数的原型。

- 将函数地址作为参数, 说明judge传入的是一个函数指针.

```
int judge(int (*pt)(const char *pt));       // 函数指针是 pt
```

12.

12. 假设有如下结构声明：

```
struct applicant {
    char name[30];
    int credit_ratings[3];
};
```

a. 编写一个函数，它将 application 结构作为参数，并显示该结构的内容。
b. 编写一个函数，它将 application 结构的地址作为参数，并显示该参数指向的结构的内容。

```
// 写一下结构体
struct applicant{
    char name[30];
    int credit_ratings[3];
}

// a. 结构体作为参数传入函数中
void func_a(application temp);      // 函数原型
```

```
void func_a(application temp)
{
    using namespace std;
    cout << temp.name << endl;
    cout << temp.credit_ratings[0] << endl;
    cout << temp.credit_ratings[1] << endl;
    cout << temp.credit_ratings[2] << endl;
}

// b. 结构体的地址作为参数传入函数中
void func_b(application *temp);      // 函数原型
void func_b(application *temp)
{
    using namespace std;
    cout << temp->name << endl;
    cout << temp->credit_ratings[0] << endl;
    cout << temp->credit_ratings[1] << endl;
    cout << temp->credit_ratings[2] << endl;
}
```

13.

13．假设函数 f1()和 f2()的原型如下：

```
void f1(applicant * a);
const char * f2(const applicant * a1, const applicant * a2);
```

请将 p1 和 p2 分别声明为指向 f1 和 f2 的指针；将 ap 声明为一个数组，它包含 5 个类型与 p1 相同的指针；将 pa 声明为一个指针，它指向的数组包含 10 个类型与 p2 相同的指针。使用 typedef 来帮助完成这项工作。

```
void f1(applicant * a);
const char * f2(const applicant *a1, const applicant *a2);

// part 1: p1 和 p2 都是函数指针
// 借助 typedef 写一个函数指针类型的别名
typedef  void (*type_1)(applicant a);
typedef const char *(*type_2) (const applicant *a1, const applicant
*a2);
tpye_1 p1 = f1;      // 得到 f1 的函数指针 p1
type_2 p2 = f2;      // 得到 f2 的函数指针 p2

// part 2: 声明函数指针数组
tpye_1 ap[5];       // 5 个函数指针的数组
tpye_2 (*pa)[10];     // 10 个函数指针的数组 的指针, 千万别漏了括号!!!!
```