函数指针的声明

函数指针指向的是函数而非对象

而我们在声明函数指针时，只需要用指针替换函数名即可。例如:

```
bool lengthCompare(const string &, const string &);

bool (*pf)(const string &,const string &);
```

> 从我们 声明的名字开始观察，pf前面有个*，因此pf是指针；右侧是 形参列表，表示pf指向的是函数；再观察左侧，发现函数的返回类型是布尔值。因此pf就是一个指向函数的指针，其中该函数的参数是两个const string的引用，返回值是bool类型。

注意：*pf两边必须加上括号，否则pf会被认为是一个返回值为bool指针的函数

函数指针的使用

当我们把函数名作为一个值使用时，该函数自动地转换成指针。例如，按照如下形式，我们可以将lengthCompare的地址赋给pf：

```
pf=lengthCompare;  //pf指向lengthCompare函数
pf=&lengthCompare;  //与上述等价，因为函数名就是函数首地址
```

另外，我们还能直接使用指向函数的指针来调用对应的函数，并且无须提前解引用指针：

```
bool b1 = pf ("hello","goodbye");//调用lengthCompare函数
bool b2 = (*pf)("hello","goodbye");//一个等价的调用
bool b3 = lengthCompare("hello","goodbye");//另一个等价的调用
```

以上三种调用方式都是合法的。

对于函数的匹配，与普通函数是相同的。

```
string::size_type sumLength(const string &,const string);
bool cstrigCompare(const char*,const  char*);
pf=0;//正确：pf不指向任何函数
pf=sumLength;//错误 ： 返回类型不匹配
pf=cstringCompare;//错误：函数类型不匹配
pf=lengthCompare;//正确：函数和指针的类型精确匹配
```

重载函数的指针

当我们使用重载函数时，上下文必须清晰地界定到底应该选用哪个函数。但如果我们定义了指向重载函数的指针

```
void ff(int *);
void ff(unsigned int);
void (*pf1)(unsigned int)=ff;//pf1指向ff(unsigned)
```

那么编译器会通过指针类型来确定选用哪个函数，指针类型必须与重载函数中的某一个精确匹配

```
void (*pf2)(int)=ff;//错误：没有任何一个ff与该形参列表匹配
double (*pf3)(int *)=ff;//错误：ff和pf3的返回类型不匹配
```

函数指针形参

和数组类似，虽然不能定义函数类型的形参，但是形参可以是指向函数的指针。此时，形参看起来是函数类型，实际上确实当成指针使用：

```
//第三个形参是函数类型，他会自动地转换成指向函数的指针
void useBigger(const string &s1,const string &s2,bool pf(const string &,const string &));

//等价的声明：显式地将形参定义成指向 函数的指针
vovid useBigger(const string &s1,const string &s2,bool (*pf)(const string &,const string &));
```

而我们可以直接把函数作为实参使用，此时它会自动地转换成指针：

```
//自动将函数lengthCompare转换成指向该函数的指针
useBigger(s1,s2,lengthCompare);
```