使用指针做函数返回值

- 当使用指针做为函数的返回值时，主函数处的**char *p**将获得调用函数**char *pf;**的值，即一个地址值。此时需要我们注意的是该地址值所指向的空间是否存在**(**即已向操作系统声明注册，不会被释放，即可能被其他操作修改**)**
- 使用栈内存返回指针是明显错误的，因为栈内存将在调用结束后自动释放，从而主函数使用该地址空间将很危险。

```
char* GetMemory(){
char p[] = "hello world";
return p;
}
void main(){
//出错！得到一块已释放的内存
char *str = GetMemory();
printf(str);
}
```

- 使用堆内存返回指针是正确的，但是注意可能产生内存泄露问题，在使用完毕后主函数中释放该段内存。

```
char* GetMemory(){
 char *p = new char[100];
 char *p = (char*)malloc(sizeof(char)*100);
 return p;
}
void main(){
 char *str = GetMemory();
 delete [] str; //防止内存泄露!
 free(str);
}
```

- 有时候函数的返回值是一个纯虚类的指针
- 按照上一个条堆内存返回指针的要求，需要创建一个纯虚类对象，显然是错误的
- 此时需要使用二级指针：指向纯虚类指针的指针

```
Element* ParseCode::parseSentence(QString sentence){
 Element** res=new Element*;
 *res=createNoShapeElement(sentence);
 return *res;
}
```

使用指针做函数参数

有的情况下我们可能需要需要在调用函数中分配内存，而在主函数中使用，而针对的指针此时为函数的参数

- 直接使用形参分配内存的方式显然是错误的，因为实参的值并不会改变，如下则实参一直为**NULL:**

```
void GetMemory(char* p){
 char *p = new char[100];
}
void main() {
 char *str;
 GetMemory(str);
 strcpy(str, "hi"); // str = NULL
}
```

- 由于通过指针是可以传值的，通过指针传递改变了实参的值,采用指向指针的指针来进行在调用函数中改变指针

```
  void GetMemory(char** p) {
 char *p = new char[100];
 }

 void main() {
 char a = 'a';
 char* str = &a;
 //定义一个指向指针的指针
 char** pstr = &str;
  GetMemory(pstr);
  strcpy(str, "hi");
  }
```