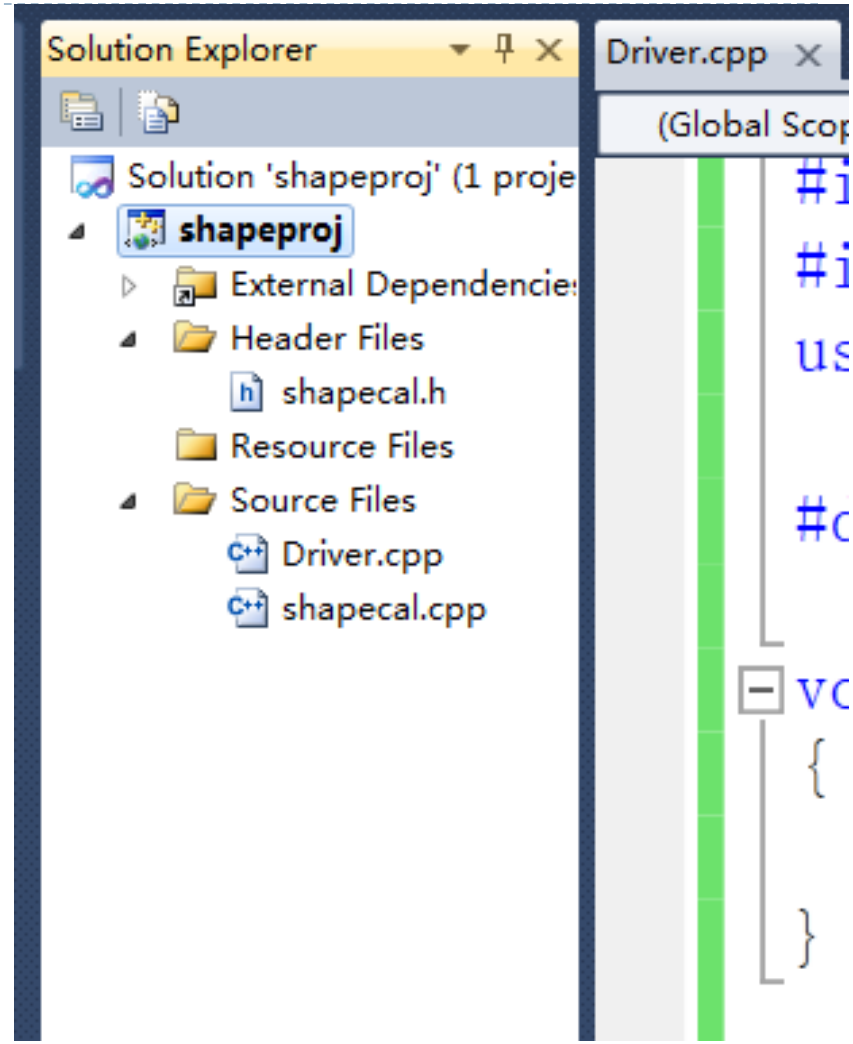
第6章　程序结构

目录

- 外部存储类型和静态存储类型
- 作用域与生命期
- 预编译
- 头文件
- 多文件结构

工程实例

- 多文件工程
  - 对软件按功能进行划分
  - 每个功能模块对应一对文件(.h/.cpp: 头文件/源程序文件)



工程实例

```
/*============================
 *shapcal.h
 *function:计算图形面积、周长
等函数的声明头文件
*============================*/
#ifndef  _SHAPCAL_H
#define _SHAPCAL_H

extern double const PI;
typedef struct _point
{
double x;
double y;
}TagPoint;

//计算三角形周长函数
double GetShapeCircum(TagPoint,
TagPoint,TagPoint);
//计算矩形周长函数
double
GetShapeCircum(TagPoint,TagPoint);
//计算圆周长函数
double GetShapeCircum(double);
//计算三角形面积函数
double GetShapeArea(TagPoint,
TagPoint,TagPoint);
//计算矩形面积函数
double GetShapeArea(TagPoint,TagPoint);
//计算圆面积函数
double GetShapeArea(double);

#endif
```

头文件卫士

外部变量声明

工程实例

```
/*==================================
 *shapcal.cpp
 *function:计算图形面积、周长等
 *==================================*/
#include <cmath>
#include "shapecal.h"

const double PI=3.14159;

static double GetLenOfLine(TagPoint
point1, TagPoint point2)
{
    double len=std::sqrt((point1.x-
      point2.x)*(point1.x-point2.x)+
      (point1.y-    point2.y)
       *(point1.y-point2.y));
return len;
}
```

全局数据定义

静态函数

```
//计算三角形周长函数
double GetShapeCircum(TagPoint point1,
TagPoint point2,TagPoint point3){
    double len1, len2, len3;
    len1 = GetLenOfLine(point1,point2);
    len2 = GetLenOfLine(point2,point3);
    len3 = GetLenOfLine(point3,point1);
    return len1+len2+len3;
}
//计算矩形周长函数
double GetShapeCircum(TagPoint
point1,TagPoint point2){
  double len1,len2;
  len1 = std::fabs(point2.x-point1.x);
  len2 = std::fabs(point2.y-point2.y);
  return (len1+len2)*2;
}
//计算圆周长函数
double GetShapeCircum(double r){
    return 2*PI*r;
}
```

工程实例

```
//计算三角形面积函数
double GetShapeArea(TagPoint point1, TagPoint point2,TagPoint point3)
{
    double len1, len2, len3, s;
    len1 = GetLenOfLine(point1,point2);
    len2 = GetLenOfLine(point2,point3);
    len3 = GetLenOfLine(point3,point1);
    s = (len1+len2+len3)/2;
    return sqrt(s*(s-len1)*(s-len2)*(s-len3));
}
//计算矩形面积函数
double GetShapeArea(TagPoint point1, TagPoint point2)
{
   double len1,len2;
   len1 = std::fabs(point2.x-point1.x);
   len2 = std::fabs(point2.y-point2.y);
   return len1*len2;
}
```

```
//计算圆面积函数
double GetShapeArea(double r)
{
     return PI*r*r;
}
```

工程实例

```
#include <iostream>
#include "shapecal.h"
using namespace std;

#define DEBUG

#ifdef DEBUG
void display(double x)
{
   cout<<"x="<<x<<endl;
}
#endif

int main()
{
  TagPoint p1, p2, p3;
  double trigCircum, rectCircum;

  p1.x=0;  p1.y=0;
  p2.x=3;  p2.y=4;
  p3.x=7;  p3.y=8;

  trigCircum = GetShapeCircum(p1,p2,p3);
  rectCircum = GetShapeCircum(p1,p3);
#ifdef DEBUG
  display(trigCircum);
  display(rectCircum);
#endif

  return 0;
}
```

条件编译

外部存储类型

在程序工程中，会面临多文件结构，即多个代码文件合在一起构成一个程序

- 外部存储类型：用extern修饰的名字（一般指函数和变量这两个实体）
- 作用：为各代码文件所共享
- 存储：在全局数据区
- 遵循一次定义原则，先声明后使用原则
- 在一个程序中，要严格遵守外部存储类型的类型声明严格一致，否则会产生运行错误。

静态存储类型

- 静态存储数据意味着一旦创建（实体）,则不再消失,除非程序结束运行（驻留在全局数据区）
- 静态存储类型分静态数据与静态函数
  - 静态存储类型以代码文件为共享区域，只在定义的文件可见，在代码区域隔绝其它代码文件
- 静态数据分为静态全局数据和静态局部数据
  - 静态全局数据以一个代码文件为共享区域，在代码区域上隔绝其它代码文件
  - 静态局部数据以一个函数的前后不同调用为共享区域，在时间区域上隔绝任何其他函数

作用域与生命期

- 作用域
  - 指标识符可见的范围，或者说是标识符有效的范围。
  - 全局作用域：
    - 全局变量、函数：从定义开始到文件结束，可通过外部声明扩展到其它文件
    - 静态全局变量和静态函数：定义开始到文件结束，文件作用域
  - 局部作用域：
    - 局部变量：
      - 函数内部定义的变量，定义开始到函数结束。
      - 复合语句块内部定义的变量，如：
      - 局部变量与全局变量同名时，局部变量优先于全局变量

```
for(int i=0; i<10; i++)
    cout<<i;
i=10;
```

作用域与生命期

- 生命期：指数据驻留内存空间的生存期
- 静态生命期——在全局数据区驻留
  - 数据一旦创建就不会消失(与程序共存亡),有全局数据,全局静态数据,局部静态数据
- 局部生命期——在栈数据区驻留
  - 随函数调用和返回,形成函数作用域的数据生命期
  - 随语句块开始和结束,形成局部作用域的数据生命期
- 动态生命期——在堆数据区驻留
  - 随new创建和delete销毁,人为决定其生存期,或称动态内存数据