测量程序设计

朱学军

2014 年 3 月

目录

第 1 章　Windows 中的编程环境

1.1　编程语言

在现在有很多的编程语言出现，每一种语言都有在解决某一方面问题上的独到之处。比较常用的语言如 C 和 C++、Visual Basic、Delphi、Java、dotNet 环境下的C#、VB.net 等语言。C 语言是典型的功能强大的面向过程编程语言，许多操作系统和系统软件都用它编写。Visual Basic 是 windows 平台上可以进行快速开发的一种语言，它的基本语言为 Basic，具有事件驱动编程的能力，成为众多编程爱好者所喜爱的编程工具。Delphi 以其良好的语言性能和快速开发能力而闻名于 windows 平台，有众多的爱好者，Object Pascal 语言是其工具，编程者需要有一定的面向对象知识。Java 语言是一种纯面向对象语言，以前主要用于网络编程，目前在传统的编程领域的应用也越来越多。dotNet 环境下的各种编程语言也是面向对象的编程语言，它们在网络和传统编程领域里的应用也越来越广，它们是编程领域的一次革命和创新。

在这些语言中 C++ 不仅兼容 C，还具有面向对象的能力。相对于其它语言，是效率最高、功能最强大的语言，当然也是较为难学的语言。C 和 C++ 是各种系统软件和高效率、高性能算法软件与图形软件开发的首选。

1.2　编程工具：Visual C++ 与其它编程工具

1.2.1　Turbo C 2.0

Turbo C 是 16 位 DOS 操作系统下的 C 语言集成编程环境，集编辑、编译、调试等多种功能于一体，是初学者入门阶段较为喜欢的一个软件。

但由于 32 位的 windows 操作系统的普及应用而被淘汰。

1.2.2 其它的 Borland 编译工具

Borland C++、Borland C++ builder 等系列为 Borland 公司的 C 和 C++ 编译器，它们均有自己的类库和快速开发能力，是 windows 环境下功能强大的开发工具。

1.2.3 Microsoft 的编译工具

现在在 windows 平台上 Visual C++ 成为了事实上的 C 和 C++ 编程工具，拥有了很大比例的市场份额。支持 C 语言和 C++ 语言。

1.2.4 其它的编译工具

如 GNU GCC，它是开源组织的 C 和 C++ 编译工具，能够在多个平台上运行，如 windows、Linux 等操作系统。

1.3 Visual C++ 的 C 与 C++ 编程环境

1.3.1 GUI 程序和 Consol 程序

windows 操作系统下的程序分为两种，用通俗的方式可以理解为，一种为窗口形式的程序即图形界面程序；一种没有图形界面即控制台（命令式）程序，它一般运行在 cmd.exe 环境中。Visual C++ 具备开发这两种程序的能力，它所对应的工程类型为：

图形程序工程：Win32 Application

控制台程序工程：Win32 Console Application

一般初学者可以选用控制台程序工程，这类程序没有图形界面，初学者可以将注意力完全集中在语言和程序的功能上。

1.3.2 Visual C++ 的文件管理和组织方式

Visual C++ 是以工作区、工程、文件的方式来组织文件的，通常一个较大的软件开发项目是用一个工作区 (workspace) 进行管理的，一个工作区里包含多个工程 (project), 一般一个工程对应一个具体的生成文件 (如可

执行文件等)。一个工程中则包含多个文件 (如 C 语言的头文件 (*.h)，源文件 (*.c) 等)。工作区文件的扩展名为 dsw，当一个工程创建好以后，下次再打开时，只需双击扩展名为 dsw 的文件，即可将所有工程下的所有文件打开。Visual C++ 的工作区下的文件视图如图 1.1所示。

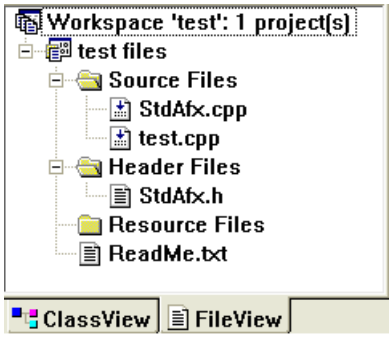

图 1.1: Visual C++ 的文件视图

相应的 Windows 资源管理器下的文件组织形式如图 1.2所示。

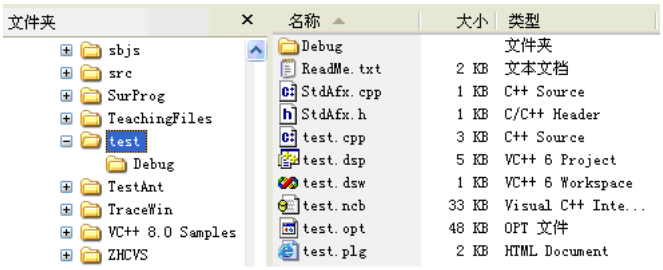

图 1.2: Windows 资源管理器下的文件组织形式

1.3.3 Visual C++ 下程序的编辑、运行、调试和发布

程序的编辑与编译

一段可执行程序是需要正确的语法的支持并要经过编译才能运行的 (Visual C++ 的语法检查相对与 Turbo C 而言更加严格，可能原来在 Turbo C 2.0 中能编译的程序在 Visual C++ 不能编译通过)。Visual C++ 提供了功能强大的编辑、编译功能。语法不正确的程序在程序的编译阶段可以检查出来，Visual C++ 在检查出这些语法错误后，会同时给出相应的

语法信息供纠正的，在实践操作环节，我们应该仔细的阅读这些信息，以帮组我们快速的找到这些语法错误并进行纠正。

程序的调试

语法正确的程序不一定在功能上也是正确的，因此在程序的运行阶段，我们也需要一些工具来帮组我们找出程序逻辑上的错误。在 Visual C++ 集成开发环境 (IDE) 中，为程序员提供了强大的调试程序功能。利用这些调试功能可以有效的跟踪和查找、排除程序在运行阶段或程序逻辑上存在的错误。

在 Visual C++ 中，按 F9 键可以在程序的的任何可执行语句处设置断点，按 F5 键即可让程序在调试模式下运行，在程序执行到有断点的地方就会停下来。这时我们就可以利用各种工具分析和查找程序中存在的逻辑错误。图 1.3为 VC6 的程序调试截图。

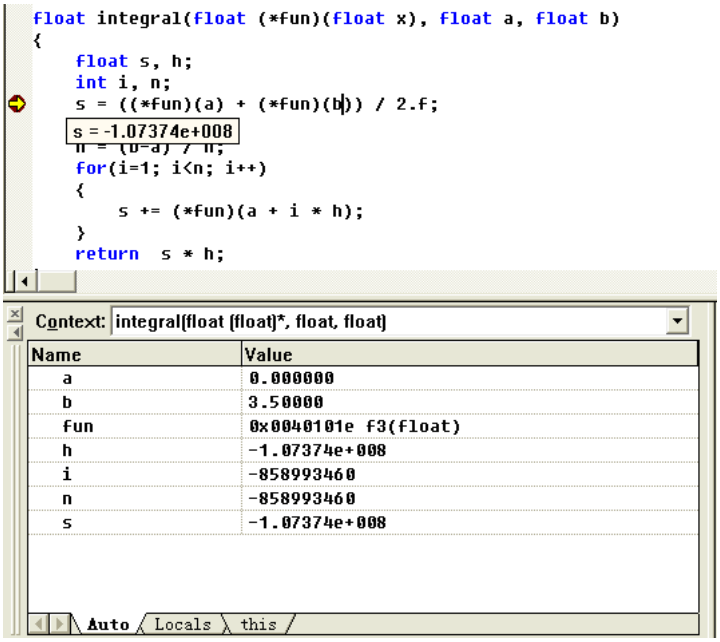

图 1.3: VC 程序调试截图

程序的发布

在 Visual C++ 中的程序有两种形式：Debug 和 Release，一般在程序编辑及调试阶段都使用 Debug 模式，在将程序正式交付给客户等情况下使用 Release 模式。Debug 模式下的程序含有大量的调试信息，可帮助程序员发现和寻找程序中的错误。Release 模式下的程序则不包含调试信息，可执行文件更小，程序运行速度更快。

第 2 章　C/C++ 语言基础

本章将对 C 语言的主要知识点进行复习，以期能对 C 语言进行更深层次的掌握。

2.1　编程环境与 C 语言的基本数据类型

程序设计要受到硬件和操作系统等系统环境的限制的。在 16 位编程环境下数据类型 int 一般为 2 个字节即 16 位，在 32 位编程环境下则为 4 个字节 32 位。同时由于字符集扩展的需要，也引入了多字节字符集和宽字节字符集。因此将 char 字符称为单字节字符，unicode 字符即宽字符用`wchar_t` 定义，为 16 位的字符。这些都是在编程过程中应该注意的问题，否则在字符串的处理中会遇到麻烦。

2.2　C 语言下的 struct

C 语言使用 struct 来让用户定义自己的数据类型，现在其它编程语言也有相应的定义自定义数据类型的方式。如测量上常用的点，简单的说它具有点名（name）、平面坐标（x、y）和高程（z）等属性，在编程时我们可以定义为如下形式：

```
1 struct Point
2 {
3     char name[11]; //点名
4     double x, y; //点的平面坐标
5     float z; //点的高程
6 };
```

从而将 Pnt 作为我们自定义的一个数据类型。如果我们要定义十个两个元素的数组，可定义为：`struct Point pnts[10];`，在扩展的 C 语言模式下，在 Pnt 前可以不加`struct`。

对结构成员元素的引用形式为：

```
pnts[0].x = 100; pnts[0].y = 100; pnts[0].z = 100;
.....................
for(int i=0; i<2; i++)
{
    printf("%s:x=%lf,y=%lf,z=%lf\n",
    pnts[i].name, pnts[i].x,
    pnts[i].y,    pnts[i].z );
}
```

对结构类型的数据也可用指针的方式进行引用，即使用 “->” 进行引用结构成员。如下例所示：

```
struct Point * pp = &pnt;
strcpy(pp->name, "A01");
pp->x = 100; pp->y = 100; pp->z = 100;
```

结构体是 C 语言的重要的构造自定义类型的工具。其用途主要表现在以下几个方面：

2.3 指针

指针是 C 和 C++ 语言的重要组成部分。32 位的操作系统上一个指针的大小为 4 个字节。

在 C 中，指针有很多用途，如用在函数参数中返回多个值等。

例如：我们需要一个函数将一个弧度值转换成度分秒并将其分别返回。这是一个有效的的方法就是使用指针参数。

```
#define PI 3.14159265f
int RADtoDMS(float rad, int * degree, int * minute, float
     * second)
{
    rad = rad / PI * 180;//将弧度化为度
    *degree = (int)rad;//取度
```

```
6
7       rad = (rad - *degree)*60;
8       *minute = (int)rad;//取分
9
10      *second = (rad - *minute)* 60;//取秒
11      return *degree + *minute/100.f + *second/10000.f;
12  }
```

程序调用形式为：

```
1  int main(int argc, char* argv[])
2  {
3      int d, m;
4      float s;
5
6      RADtoDMS(1.3254f, &d, &m, &s); //注意 d,m,s 要传地址
7      printf("%d %d %f\n", d, m, s);
8
9      return 0;
10 }
```

程序输出结果为：`75 56 23.377075`

2.4 函数

1. 主函数：`int main(int argc, char* argv[ ])`

其它形式的主函数：`int main()` 和`void main()`

2. 函数的声明与返回类型：严格写出函数的声明形式（返回值类型、函数名、参数类型、参数顺序和个数）和返回类型。

```
1  int add(int x, int y)
2  {
3      return x + y;
4  }
```

不能写成：`add(int x, int y)` 形式。

2.5 函数指针

如编写程序求解以下定积分：

$$y_1 = \int_0^1 (1+x^2)dx \tag{2.1}$$

$$y_2 = \int_0^2 (1+x+x^2+x^3)dx \tag{2.2}$$

$$y_3 = \int_0^{3.5} (\frac{x}{1+x^2})dx \tag{2.3}$$

以上三式可以统一写为 $y = \int_a^b f(x)dx$，在此采用梯形法求定积分，函数 $f(x)$ 在区间 (a,b) 的梯形法计算公式为：

$$s = h[\frac{f(a)+f(b)}{2} + f(a+h) + f(a+2h) + ... + f(a+(n-1).h)] \tag{2.4}$$

式中：$h = (b-a)/n$

编程思路：用函数指针变量的方法，先编写一个求定积分的通用函数 *integral*，再将各个积分函数以参数的形式传递给它，进行计算。

积分示例程序：

```
1  float f1(float);
2  float f2(float);
3  float f3(float);
4  float integral(float (*fun)(float), float, float);
5
6  int main(int argc, char* argv[])
7  {
8      printf("␣y1=␣%6.2f␣\n␣y2=␣%6.2f␣\n␣y3=␣%6.2f␣\n␣",
9          integral(f1, 0.f, 1.f), integral(f2, 0.f, 2.f),
10         integral(f3, 0.f, 3.5f) );
11
12     return 0;
13 }
14
15 float f1(float x)
16 {
```

```
17      return 1 + x * x;
18  }
19
20  float f2(float x)
21  {
22      return 1 + x + x*x + x*x*x;
23  }
24
25  float f3(float x)
26  {
27      return x / (1 + x * x);
28  }
29
30  float integral(float (*fun)(float x), float a, float b)
31  {
32      float s, h;
33      int i, n;
34      s = ((*fun)(a) + (*fun)(b)) / 2.f;
35      n = 100;
36      h = (b-a) / n;
37      for(i=1; i<n; i++)
38      {
39          s += (*fun)(a + i * h);
40      }
41      return  s * h;
42  }
```

运行结果为：

```
y1=   1.33
y2=  10.67
y3=   1.29
```

2.6 内存的动态分配和动态数组的实现

计算机内存有两种分配方式：在栈（ stack ）上和在堆（ heap ）上分配。一般的变量由系统自动分配在栈上，也由系统自动回收。优点是效率高，缺点是作用范围小。如果要在更大范围内使用变量，则要使用堆分配

方式。

在堆上分配的内存，要由 malloc 函数显式分配，要由 free 函数显式回收。

我们看下面例子：

```
#include <stdio.h>
#include <malloc.h>

void fun(int n)
{
    double x[n];//语法错误

    int i;
    for(i=0; i<n; i++)
    {
        x[i] = (double)i;
    }

    for(i=0; i<n; i++)
    {
        printf("x[%d]=%lf\n", i, x[i]);
    }
}

int main()
{
    int n;
    printf("Enter value of n:");
    scanf("%d", &n)
    fun(n);
    return 0;
}
```

在 fun 函数中，我们想根据参数值 n 动态的定义一个数组 x[n] ，但这种方式在 C 语言中是不允许的，在编译时会出现编译错误。

正确的 fun 函数实现方法为：

```
void fun(int n)
{
    double * x;//把x声明double类型的指针

    int i;

    //通过malloc函数申请分配n个double类型的内存空间
    x = (double *)malloc(n * sizeof(double));

    for(i=0; i<n; i++)
    {
        x[i] = (double)i;
    }

    for(i=0; i<n; i++)
    {
        printf("x[%d]=%lf\n", i, x[i]);
    }

    free(x);//使用完毕后应该用free函数释放内存
}
```

以上示例程序说明了如何用指针实现动态数组。在这种用法中，要注意申请的内存一定要释放，否则，会造成内存泄漏。

下面我们在举一例子来说明堆上分配的内存的存在周期。

```
#include <stdio.h>
#include <malloc.h>
double * fun1(int n)
{
    double * x;//把x声明double类型的指针
    //通过malloc函数申请分配n个double类型的内存空间
    x = (double *)malloc(n * sizeof(double));
```

```
    return x;
}

void fun2(double x[], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        x[i] = i;
    }
}

void fun3(double x[], int n)
{
    int i;
    for(i=0; i<n; i++)
    {
        printf("x[%d]=%lf\n", i, x[i]);
    }
}

int main()
{
    int n;
    double * p;
    printf("Enter value of n:");
    scanf("%d", &n);
    /*调用fun1函数分配n个double类型元素的内存空间*/
    p = fun1(n);
    if(p == NULL) return -1;
    /*调用fun2函数为指针p所指向的数组赋值*/
    fun2(p, n);
    /*调用fun3函数将指针p所指向的数组中的值输出*/
```

```
    fun3(p, n);
    /*释放指针p所指向的内存*/
    free(p);

    return 0;
}
```

2.7 文本文件的读写

文本文件读写示例程序：

```
#include <stdio.h>
#include <string.h>
int main(int argc, char* argv[])
{
    FILE * in;
    char fileName[256], buffer[256];

    if(argc == 1)
    {
        printf("请输入文件名:");
        scanf("%s", fileName);
    }
    else
        strcpy(fileName, argv[1]);

    in = fopen(fileName, "r");
    if(in == NULL)
    {
        printf("你输入的文件名不存在! ");
        exit(-1);
    }

    while( !feof(in) )
```

```
    {
        fscanf(in, "%s", buffer);
        printf("%s\n", buffer);
    }

    fclose(in);
    return 0;
}
```

2.8 用 C++ 的非面向对象特性对 C 扩充

2.8.1 行注释

保留 C 语言的块注释方式：/* ……*/
新增 C++ 的行注释方式：//

2.8.2 灵活的局部变量说明方式

允许在代码块的任何地方说明局部变量，作用范围为从其说明点到所在的最小分程序末的范围有效。

例：

```
int fn()
{
    int i = 10, k = 0;
    ……
    k++;
    int j = i + 20; //在C中不允许，C++中可以，
                    //作用范围为该句至所在范围的“}”
    ……
}
```

2.8.3 结构、联合和枚举名直接作为类型名

结构、联合和枚举名均为类型名，在定义变量时，不必在其前冠以 struct、union、enum。如例 4 所示：

```
#include <iostream>
using namespace std;
enum Week{Sunday, Monday, Tuesday,
Wednesday, Thursday, Friday, Saturday};
int main()
{
    count << Sunday << endl;
    return 0;
}
```

输出结果为：0

2.8.4 用 const 修饰符取代宏进行常量的声明

如 C 中定义常量：`#define PI 3.14159265`

C++ 中定义如下：`const double PI = 3.14159265;`

const 定义的常量有数据类型，C++ 编译程序可以进行更加严格的类型检查，具有良好的编译时检测性。

2.8.5 用内置函数（inline）取代宏进行表达式的定义

如 C 中：`#define MAX(X, Y) ((X) > (Y) ? (X) : (Y))`

C++ 中定义为：`inline int MAX(int X, int Y){ return X > Y ? X : Y; }`

2.8.6 更加严格的函数原型声明

C++ 中函数的原型为函数的名称、参数类型、参数顺序及返回值的类型。例：

```
#include <iostream>
using namespace std;

int Add(int, int);//严格的函数原型声明
int main()
{
    int a = 10, b = 20;
```

```
    cout << Add(a, b) << endl;
    return 0;
}

int Add(int x, int y) { return x + y; }
```

2.8.7 增加了带有缺省参数的函数

如`int Add(int x, int y = 0){……}`，则在调用时以下两种形式是等价的：`int sum = Add(5, 0);` 与`int sum = Add(5);`

2.8.8 增加了函数重载功能

以下函数均为合法函数：

名称相同，但参数的个数或参数的类型不相同的多个函数称为函数重载。

```
int Max(int X, int Y){return X > Y ? X : Y;}
float Max(float X, float Y){return X > Y ? X : Y;}
```

2.8.9 作用域运算符::

在局部可使用“::”访问全局变量。

2.8.10 无名联合

如`union{int i; float f;};` 变量 i 和 f 具有相同的存储地址。

2.8.11 强制类型转换

C 中：`int i = 10; float f = (float)i;`

C++ 中：`int i = 10; float f = float(i);`

2.8.12 新增运算符 new 和 delete

使用运算符 new 和 delete 代替函数 malloc() 和 free() 动态分配内存和释放动态分配的内存。

如在 C 中，申请 100 个 int 类型的内存的方式为：

```
int * m = (int *)malloc(100 * sizeof(int) );
```

而在 C++ 中只需写成：

```
int * m = new int[100];
```

释放内存 C 的方式为：`free(m);`

释放内存 C++ 的方式为：`delete[] m;`

它们最关键的区别在于 *malloc* 和 *free* 是函数，*new* 和 *delete* 则是运算符。

2.8.13 引用

自动间接引用的一种指针。可为变量起别名，主要用作函数参数及函数的返回类型。如交换两个变量的值的函数我们用指针参数的形式实现如下：

```
void Swap(double * x, double * y)
{
  double tmp;
  tmp = *x;
  *x = *y;  //用指针的形式取值
  *y = tmp;
}
```

调用的形式只能为如下形式：

```
.....
double m = 123.123;
double n = 345.345;
Swap(&m, &n);
.....
```

从以上我们可以看出，函数实现中我们得频繁的使用 $*X$ 等符号，在调用中，我们得向其传递变量的地址，这使我们的程序显得不太自然。

如果我们使用引用，则可将以上函数实现为：

```
void Swap(double & x, double & y)  {
  double tmp;
  tmp = x; //
  x = y;   //对变量的引用与一般变量一样
  y = tmp; //
}
```

调用的形式与一般的函数调用形式相同：

```
  ...........
  double m = 123.123;
  double n = 345.345;
  Swap(m, n);//与一般函数调用形式相同
  ...........
```

由此可见，使用引用在函数传值中显得更好，更自然。

第 3 章　面向对象的 C++

3.1　结构与类

在 C++ 中，结构（struct）与类（class）具有相同的作用，它们的区别是在默认情况下 struct 中的成员 (member) 的访问类型是 public，而 class 中成员的访问类型是 private。

3.2　类的声明形式

```
class className
{
private:
    //私有成员变量和成员函数，只能由类的成员函数访问
protected:
    //保护成员变量和成员函数，可由类的成员及其子类的成员函数访问
public:
    //公开的成员变量和成员函数，可由外部函数访问
};
```

3.3　成员函数的定义

1．在 class 的内部定义，即为类的 inline 函数，如：

```
class point
{
private:
```

```
    int x;
public:
    int getX(){ return x; }
    void setX(int value){ x = value;}
};
```

2. 在 class 的外部定义，在 class 内进行声明如：

```
class point
{
private:
    int x;
public:
    int getX();
    void setX(int value);
};
int point::getX(){ return x; }
void point::setX(int value){ x = value;}
```

3.4 对象的定义及引用

我们可以简单的把类看作与 int，double 等一样的数据类型，对象就是类的实例。如定义一个 point 类的实例 pt0，可定义为：point pt；

引用方法为：

```
pt.setX(5); cout << pt.getX() << endl;
```

如果我们再定义：`point * p = &pt;`

则引用方法为：`p->setX(5); cout << pt.getX() << endl;`

3.5 类的作用域

为在类的声明中的符号 “{” 与 “}” 之间，类的所有成员都在该类的作用域内。类的任何成员均可引用该类的其它成员，但在类的作用域之外，对类成员的应用则会受到限制。

3.6 构造函数与析构函数

构造函数是类的特殊成员函数，主要作用为对类进行初始化。

构造函数的名字必须与类型相同，可以有参数，不能有返回类型。

只能隐式调用，不能显式调用。

3.6.1 一般构造函数

如在类 point 中，point() 与 point(int initValue) 均为构造函数。

```
class point
{
private:
    int x;
public:
    point();
    point(int initValue);
    int getX();
    void setX(int value);
};
int point::getX(){ return x; }
void point::setX(int value){ x = value;}
point::point(){}
point::point(int initValue){ x = initValue;}
```

由上也可看出，构造函数可以有重载形式。

3.6.2 拷贝构造函数

是一种特殊的构造函数，是根据已存在的对象创建一个新的对象，并将其值作为新的对象的值。

如类 point 中的`point(const point& object)` 函数。

```
class point
{
private:
```

```
    int x;
public:
    point();
    point(int initValue);
    point(const point& object);
    int getX();
    void setX(int value);
};
int point::getX(){ return x; }
void point::setX(int value){ x = value;}
point::point(){}
point::point(int initValue){ x = initValue;}
point::point(const point& object){ x = object.x; }
```

3.6.3 析构函数

是特殊的类成员函数，执行与构造函数相反的操作，用于清理或释放对象所占用的资源。它的名字与类名相同，但前必须加一符号”~”；它没有参数，没有返回值，不能有重载形式，不能显示调用，它在类释放时被自动调用。

下面我们看一实例，以验证类的构造函数与析构函数的执行顺序。

```
#include <iostream>
using namespace std;
class point
{
public:
   point(){ cout << "类point的构造函数: point()执行…" << endl;}
   ~point(){cout <<"类point的析构函数: ~point()执行…" << endl;}
};
void main()
{
point pt;
cout << "主函数main()执行…" << endl;
}
```

程序输出为：

```
类point的构造函数：point()执行…
主函数main()执行…
类point的析构函数：~point()执行…
```

3.7 静态数据成员和静态成员函数

在类中用关键字 static 声明的数据成员为类的静态数据成员，主要用于表示类的唯一的公共属性。用关键字 static 声明的成员函数为类的静态成员函数，主要用于对类的静态成员的访问。

3.8 对象数组与对象指针

对象数组为用对象声明的数组，如：`point pnts[10];`，则 pnts 为 10 个元素的 point 类型的数组。

用对象声明的指针为对象指针。如`point * pPnt;`，则 pPnt 为 point 类型的指针。

可以看出，这些概念与普通的数据类型的概念是一致的。

每个类都有一个默认的指针：this，如以上 point 类的函数 setX 的函数体可以写为：

```
this->x = initValue
```

3.9 友元

友元函数

友元成员

友元类

第 4 章　Windows 图形程序设计基础

第 5 章　常用测量函数设计

5.1　常用测量计算公式

我们以 C 的宏来定义测量计算中的常数如下：

```
#define _PI    3.14159265358979323846264338327795
#define _2PI   6.28318530717958647692528676655900
#define _PI2   1.57079632679489661923132169163980
#define _TORAD 0.01745329251994329576923690768488612
#define _TODEG 57.29577951308232087679815481410517
```

5.1.1　角度弧度互换函数：

在测量工程中角度的常用习惯表示法是度分秒的形式，而计算程序中采用的格式为 xxx.xxxxx，即以小数点前的整数部分表示度，小数点后两位数表示分，从小数点后第三位起表示秒。而在计算机编程时所用的角度要使用弧度表示的，需要设计函数相互转换。

在设计函数之前，我们应该首先定义一下两个常量，$TOANG$ 表示 $180/\pi$，用于将弧度化为度，$TORAD$ 表示 $\pi/180$，用于将度化为弧度。

```
#define TOANG 57.29577951308232087679815481410517
#define TORAD 0.01745329251994329576923690768488612
```

1．角度化弧度函数：

```
double DMStoRAD(double dms)
{
   int d, m, f; double s;
```

```
    f = dms>=0 ? 1 : -1;
    //0.001秒 = 4.848136811095359935899141023579 5e-9弧度
    dms += f * 0.0000001;

    d = (int)dms;

    dms = (dms - d) * 100.0;
    m = (int)dms;

    s = (dms - m) * 100.0;

    return (d + m / 60.0 + s / 3600.0) * TORAD
         - f * 4.848136811095359935899141023579 5e-9;
 }
```

如某一角度为 1°40 00，以 1.4000 形式输入，返回值为 0.02908882，单位为弧度。在该函数的第四行加 0.001 秒主要是考虑到浮点数的表示时有舍入误差存在，如 1°40 00 用 1.4 表示时，计算机中被描述为 1.3999999，这时不能直接取整，否则计算出的角度就会多出 40 秒。加入 0.001 秒就可消除这种舍入误差，但这样就会对别的角度造成影响，故需在计算结果中将多加的 0.001 秒的弧度值减去。

2. 弧度化角度函数

```
double RADtoDMS(double rad)
{
    int f = rad >= 0 ? 1 : -1;              // 符号 十 一
    //加0.001秒（用弧度表示），化为度
  rad = (rad + f * 4.848136811095359935899141023579 5e-9) * TOANG;

    int d = (int)rad;

    rad = (rad - d) * 60.0;
    int m = (int)rad;
```

```
    double s = (rad - m) * 60.0;

    return d + m / 100.0 + s / 10000.0 - f * 0.0000001;
}
```

5.1.2 坐标方位角推算

1. 已知 01 边的坐标方位角 α_0 和 01 边和 12 边间的水平角 β，计算 12 边的坐标方位角。

```
double Azimuth(double azimuth0, double angle)
{
    return To0_2PI(azimuth0 + angle + _PI);
}
```

2. 将角度规划到 0 ～ 2，单位为弧度

```
double To0_2PI(double rad)
{
  int f = rad >= 0 ? 0 : 1;
  int n = (int)(rad / TWO_PI);

  return rad - n * TWO_PI + f * TWO_PI;
}
```

5.1.3 平面坐标正反算

1. 坐标正算：

根据 0->1 点的坐标方位角和水平边长，计算 0->1 点的坐标增量。

```
void double dxdy(double azimuth, double distance,
                 double& dx, double& dy)
{
    dx = cos(azimuth) * distance;
    dy = sin(azimuth) * distance;
}
```

根据 0 点的坐标和 0->i 点的坐标方位角和水平边长，计算 i 点的坐标。

```
void Coordinate(double x0, double y0,
        double azimuth, double distance,
        double& xi, double& yi)
{
    xi = x0 + cos(azimuth) * distance;
    yi = y0 + sin(azimuth) * distance;
}
```

根据 1 点的坐标和后视边 (0->1) 点的坐标方位角，水平角 (0-1-i)，水平边长 (1-i)，计算 i 点的坐标。

```
void Coordinate(double x0, double y0, double azimuth0,
        double angle, double distance,
        double& xi, double& yi)
{
    double azimuthi = Azimuth(azimuth0, angle);
    xi = x0 + cos(azimuthi) * distance;
    yi = y0 + sin(azimuthi) * distance;
}
```

2. 坐标反算：

计算 0 点至 1 点的坐标方位角，返回值单位为弧度。

```
double Azimuth(double x0, double y0, double x1, double y1)
{
   double dx = x1 - x0;
   double dy = y1 - y0;
   return atan2(dy, dx) + (dy < 0 ? 1 : 0) * _2PI;
}
```

计算两点间 (0->1) 的平距

```
double Distance(double x0, double y0, double x1, double y1)
{
```

```
    double dx = x1 - x0;
    double dy = y1 - y0;
    return sqrt(dx * dx + dy * dy);
}
```

5.1.4 以类的形式进行封装

这些函数还可以以类的形式用静态成员函数的方法把它们封装在一个类里面。这在纯面向对象语言如JAVA、C# 等里面，尤其应该这样封装。

第 6 章　全站仪数据提取

内外业一体化是测量工作的一个重要内容，其中最关键的环节就是外业观测数据与内业数据处理软件的联系问题，即由测量仪器到处理软件的数据通信和格式转换。

6.1　全站仪使用及其数据通信

6.1.1　全站仪的数据存储格式

现在的全站仪内存容量是于越来越大，存储的数据也是越来越多。但它们都是以文件的形式进行组织的，用户在使用之前可以选取或创建一个文件作为当前工作文件。从测量数据的内容上讲，测量的数据主要以测角测距数据和碎部点的坐标为主。测角测距数据主要用于导线等形式的控制测量，而坐标形式的数据主要用于数字化成图。

下面为索佳全站仪 Set2110 仪器的坐标数据存储格式。

```
00NMSDR33␣V04-04.02␣␣␣␣␣01-Jan-02␣00:00␣113111

10NMJOB1␣␣␣␣␣␣␣␣␣␣␣␣121111

06NM1.00000000

01NM:SET2110␣␣␣V41-01␣␣␣021417SET210␣␣␣V41-01␣␣␣02141731␣␣␣␣␣␣␣0.000

03NM1.300

08TP␣␣␣␣␣␣␣␣␣␣␣␣B0014205205.890␣␣␣␣␣495732.686␣␣␣␣␣␣1024.991␣␣␣␣␣␣␣␣3
```

```
08TP␣␣␣␣␣␣␣␣␣␣␣␣B0024205239.848␣␣␣␣␣495724.684␣␣␣␣␣␣1024.854␣␣␣␣␣␣␣␣3

08TP␣␣␣␣␣␣␣␣␣␣␣␣B0034205258.284␣␣␣␣␣495707.995␣␣␣␣␣␣1024.709␣␣␣␣␣␣␣␣3

08TP␣␣␣␣␣␣␣␣␣␣␣␣B0044205299.872␣␣␣␣␣495698.037␣␣␣␣␣␣1024.642␣␣␣␣␣␣␣␣3
```

6.1.2 全站仪数据的下载

全站仪内存储数据的下载方法目前主要是通过串口的形式进行的。这方面的软件很多，最常见的就是 windows 操作系统中所附带的超级终端。另外，各个全站仪厂商也基本上提供相应的数据下载软件和数据提取程序。

(1) 超级终端

Windows 的超级终端是一个功能强大的串口通讯软件，利用超级终端可以输出全站仪内存中的存储数据。输出数据之前，应先连接好通信电缆，设置好通讯端口、波特率、数据位数、停止位、奇偶检校等通讯参数。输出完毕后，还应该利用超级终端所提供的选取和复制功能将下载的数据粘贴到其它形式的文件中进行保存。但须注意的是 windows 操作系统所带的超级终端不是一商用软件，因此，缓冲区的大小被限制在 500 行。如果数据文件较大的话，应该分多次进行下载，以获取完整的数据，防止数据丢失。

(2) 厂商的下载软件

如徕卡等系列的全站仪，厂商均提供了如徕卡办公室等优秀的数据下载和提取软件。它们的功能很强大，也可将数据根据用户的需求转换为相应的格式。缺点就是软件的适用范围为相应的特定仪器。

(3) 成图软件

现在各种成图软件也带有相应的数据下载和数据提取软件，如 MapGIS、广州开思等。它们的优点是能针对多种仪器、多种内存数据格式进行数据下载和提取，缺点是提取的数据一般指针对它们自己的软件。

6.1.3 数据提取

全站仪的数据下载方法基本上是一样的，也较为简单。但由于全站仪的种类和型号很多，各个全站仪厂商的数据格式也大不一样，加之各个成图软件所要求的数据格式也不一样。很难编写一个通用软件来满足各种要求。尽管如此，只要我们了解了全站仪的数据格式，要编写相应的提取程序并不是一件难事。下节我们以索佳 Set2110 全站仪的坐标数据格式为例，讲述数据提取的方法。

6.2 数据提取程序编制

为了满足我们数据处理的需要，我们将研究怎样从全站仪的数据文件格式中提取我们所需要的数据。由上节内容我们大致了解了索佳 Set2110 全站仪坐标数据格式，下面我们对其进行详细分析。

6.2.1 索佳 SDR33 数据格式

索佳 SET2110 全站仪存储的数据格式为 SDK33 格式，主要内容包括仪器型号、通信端口、存储文件、仪器类型和测站信息等等。对于坐标数据而言，包括各点的点号，X、Y、H 等数据，根据需要还可能包括其编码和连接信息。

(1) 相关信息

索佳全站仪的数据有很多信息，包括数据输出方式、仪器型号版本号、觇标高等内容，一般有 5 类，即以 00NM、01NM、03NM、06NM、10NM 开头，其每行信息的个数分别为 5、6、1、1、2。此类信息不用。

(2) 坐标信息

坐标数据的信息一般包括类型码、来源码、X 坐标、Y 坐标、高程、编码和连接信息等内容。索佳 SET2110 全站仪坐标信息中，类型码一般为 08 或 02，来源码一般为 TP，它们为一个整体，可以舍去不用。一般的测图方法中，测点的编码和连接信息也可以舍去不用。其中 02TP 表示测站点数据，08TP 表示测点坐标数据，对我们而言，只需提取以 08TP 开头的数据行数据。

6.2.2 一般数字测图软件数据的格式

不同的数字测图软件有不同的数据格式，一般可以考虑按下列方法设计。

(1) 以行为单位用分隔符为空格

```
点名(或点号)  代码  X坐标  Y坐标  H高程
...  ...  ...  ...  ...
点名(或点号)  代码  X坐标  Y坐标  H高程
```

(2) 分隔符为逗号或其它符号

```
总点数
点名(或点号), 代码,Y坐标,X坐标, H高程
...  , ...,... , ... , ...
点名(或点号), 代码,Y坐标,X坐标,H高程
```

南方 CASS 软件的数据格式即为该格式。

(3) 以行为单位

```
点名
代码
Y坐标
X坐标
H高程
```

这三种格式各有优缺点，在我们的程序设计中，应注意相关软件的使用格式。

6.2.3 程序的功能

针对以上的全站仪的数据格式和成图软件所要求的数据格式，在我们的程序中，应该完成以下功能：

(1) 对坐标点进行计数 (如果其它软件要求的话)

(2) 忽略无用信息

(3) 生成需要格式的数据文件

6.2.4 程序的编写要点

针对索佳 SDR33 格式，编写程序应注意以下几点：

(1) 判断行信息

每行信息的前四个字符是固定的一些字母或数字，如 $00NM$、$01NM$ 或 $08TP$、$02TP$ 等，在读操作时，可以先读前四个字符，和这些固定的字母或数字相比较，以判断此行是什么信息。

(2) 忽略空行

空行中没有字符，因此在判断行信息时所用的方法是不可行的。在读取行数据时要将其过滤掉。

(3) 正确提取坐标数据

以 08TP 或 02TP 开头的信息行，其后是点名、X 坐标、Y 坐标和 H 高程，它们各占 16 个字符的空间位置。其中以 08TP 开头的数据行，其后还有点的代码，占 16 个字符。对这些信息行，我们要正确的提取我们需要的数据，在此我们只讲述提取测点坐标数据（即以 08TP 开头的数据行数据）。

6.2.5 附：坐标提取程序

(为 SDR33(SOKKIA SET 2110) 至南方 CASS 软件格式)

```
#include <stdio.h>
#include <string.h>
```

```
void substr(char* destination, const char * source,
            int startPos, int length)
{
    int i = 0;
    int n = strlen(source);
    for(; (i < length) && (i < n - startPos); i++)
        destination[i] = source[i + startPos];
    destination[i] = '\0';
}

void SDR33ToCass(FILE * in,  FILE * out)
{
    char line[256];

    while(!feof(in))
    {
        fgets(line, 255, in);
        char type[5], name[17], code[11], item[17];
        double x, y, z;
        if( sscanf(line, "%4s", type)== 1)
        {
            if(strcmp(type, "08TP") == 0)
            {
                substr(item, line, 4, 16);
                sscanf(item, "%s", name);
                substr(item, line, 20, 16);
                sscanf(item, "%lf", &x);
                substr(item, line, 36, 16);
                sscanf(item, "%lf", &y);
                substr(item, line, 52, 16);
                sscanf(item, "%lf", &z);
                substr(item, line, 68, 16);
                sscanf(item, "%s", code);
```

```
                fprintf(out, "%s,", name);
                fprintf(out, "%s,", code);
                fprintf(out, "%lf,", y);
                fprintf(out, "%lf,", x);
                fprintf(out, "%lf", z);

                fprintf(out, "\n");
            }
        }
    }
}

int main(int argc, char* argv[])
{
    char SDR33[12], CG[12];

    FILE * in;  FILE * out;
    printf("请输入SDR33格式文件名（不超过11个字符）:\n");
    scanf("%s", SDR33);

    printf("请输入保存成果文件名（不超过11个字符）:\n");
    scanf("%s", CG);

    printf("\n正在运行, 请稍等!\n");
    if( ( in = fopen(SDR33, "r") ) == NULL)
    {
        printf("已知SDR33格式文件打不开!\n");
        return -1;
    }

    if ( (out = fopen(CG, "w") ) == NULL) {
        printf("成果数据文件打不开!\n");
```

```
        return -1;
    }

    SDR33ToCass(in, out);

    fclose(in); fclose(out);
    printf("\n数据提取完毕!\n");

    return 0;
}
```

这是一个用标准 C 语言书写的程序，在程序中由于找不到相应的截取字符串的库函数，就自定义了 substr 函数用于完成该项功能。如果用 Visual Basic 编程的话，可以用 Left、Mid 等标准字符串处理函数代替。

这个程序中，SDR33ToCass 函数不涉及界面相关的语法，因此这个函数也可以不加修改的用于图形界面（GUI）程序中。

第 7 章　高斯坐标正反算与换带

7.1　数据模型

7.1.1　投影换带的目的

为提供某些点的国家大地坐标或高斯平面直角坐标，可能需要解决大地坐标 (B, L) 和高斯平面直角坐标 (X、Y) 之间的换算问题，以及不同带之间的高斯坐标的换算问题。

7.1.2　投影换带的内容

1. 坐标正算：将点的大地坐标转换成高斯投影平面直角坐标。
2. 坐标反算：将点的高斯投影平面直角坐标转换成大地坐标。
3. 换带计算：将某带的点的高斯投影平面直角坐标转换成邻带或某中央子午线经度的高斯投影平面直角坐标。

7.1.3　投影换带的数学模型

投影换带程序是在椭球参数 (长半轴 a、短半轴 b、扁率 α) 一定的条件下，根据给定的数学模型来进行计算的。

1. 基本公式

   扁率：

$$\alpha = \frac{(a-b)}{a}$$

   第一偏心率：

$$e = \sqrt{\frac{a^2-b^2}{a^2}}$$

第二偏心率：

$$e' = \sqrt{\frac{a^2 - b^2}{b^2}}$$

极曲率半径：

$$c = \frac{a^2}{b}$$

赤道子午曲率半径：

$$d = \frac{b^2}{a}$$

卯酉圈曲率半径：

$$N = \frac{c}{V} = \frac{c}{\sqrt{1 + e'^2 \cos^2 B}}$$

辅助符号：

$$t = \tan B \qquad \eta = e' \cos B$$

2. 子午线弧长

$$X = d(A_0 B - B_0 \sin B \cdot \cos B - C_0 \sin^3 B \cos B - D_0 \sin^5 B \cos B - E_0 \sin^7 \cos B)$$

式中：

$$A_0 = 1 + \frac{3}{4}e^2 + \frac{45}{64}e^4 + \frac{175}{256}e^6 + \frac{11025}{16384}e^8$$
$$B_0 = \frac{3}{4}e^2 + \frac{45}{64}e^4 + \frac{175}{256}e^6 + \frac{11025}{16384}e^8$$
$$C_0 = \frac{15}{32}e^4 + \frac{175}{384}e^6 + \frac{3675}{8192}e^8$$
$$D_0 = \frac{35}{96}e^6 + \frac{735}{2048}e^8$$
$$E_0 = \frac{315}{1024}e^8$$

3. 坐标正算

$$x = X + \frac{N}{2} sinB cosB l^2 + \frac{N}{24} sinB cos^3 B(5 - t^2 + 9\eta^2 + 4\eta^4) l^4 + \frac{N}{720} sinB cos^5 B(61 - 58t^2 + t^4 + 270\eta^2 - 330\eta^2 t^2) l^6$$

$$y = Nl cosB + \frac{N}{6} cos^3 B(1 - t^2 + \eta^2) l^3 + \frac{N}{120} cos^5 B (5 - 18t^2 + t^4 + 14\eta^2 - 58t^2\eta^2) l^5$$

4. 坐标反算

$$l = \frac{1}{N_f cosB_f} y - \frac{1+2t_f^2+\eta_f^2}{6N_f^3 cosB_f} y^3 + \frac{5+28t_f^2+24t_f^4+6\eta_f^2+8\eta_f^2 t_f^2}{120N_f^5 cosB_f} y^5$$

$$B = B_f - \frac{t_f(1+\eta_f^2}{24N_f^2} y^2 + \frac{t_f(5+3t_f^2+6\eta_f^2-6t_f^2\eta_f^2-3\eta_f^4+9\eta_f^4 t_f^4)}{24N_f^4} y^4 - \frac{t_f(61+90t_f^2+45t_f^4+107\eta_f^2+162\eta_f^2 t_f^2+45\eta_f^2 t_f^4)}{720N_f^6} y^6$$

7.1.4 投影换带程序设计

(一) 功能设计程序应能够计算投影换带的三项基本功能，即：

大地坐标 转换为 高斯平面直角坐标；

高斯平面坐标 转换为 大地坐标；

高斯平面坐标 转换为 高斯平面坐标。

如果考虑国家高斯平面坐标系统与独立坐标系统，主要是改变投影面（实质是改变椭球参数）的独立坐标系统的转换，程序的功能将更完善。

(二) 模块设计

当计算的点位较多时，需要对某些参数进行循环计算，这些计算过程可以编写成模块，以调用模块的方式处理，使主函数更简明、易读。

根据程序的功能设计需要，可以考虑设计三个模块或函数，即：

大地坐标计算高斯平面直角坐标函数；

高斯平面直角坐标计算大地坐标函数；

高斯平面直角坐标换带函数。

在主函数中只考虑数据的输入、输出，简单的计算以及函数的调用等主要过程即可。

(三) 程序的功能：

1．能够计算 54、80 及自定义参考椭球的高斯坐标正反算；

2．能进行换代计算；

3．能读写文件进行多点的高斯坐标正反算和换代；

4．具有简单的命令行式的提示界面。

(四) 程序设计中的注意问题：

1．避免过多的使用全局变量。由于全局变量具有许多意想不到的副作用，在现在的程序设计方法中是要尽量避免的；

2．注意函数的声明方式。C 语言和 C++ 语言函数原型声明是不一样的，如果直接把 C 方式用到 C++ 方式，程序就会编译出错；

3．不要让主函数 (main 函数) 承担太多的逻辑功能。如果我们将所有的函数功能都写在 main 函数中，将导致我们在其它的地方无法复用现有的函数功能；

4．程序中界面 (命令式界面或图形界面) 与算法两部分功能应尽量分离。如果耦合处太多，若我们要将其由一种界面 (如命令界面) 改到另一种界面 (如图形界面）下，其中的大部分代码将不能使用。

7.2 用 C 语言进行程序设计

C 语言是典型的面向过程设计语言，小巧、灵活，功能强大。下面我们将从一个小功能开始构造满足以上全部功能的程序。

在此我们先完成正算功能，我们设定正算中所用的角度均为弧度，从而避免在正算中进行度分秒与弧度的转换问题，我们也先把读数据文件、多点等问题去掉，从只计算一个点开始考虑问题。我们现在只考虑 54 北京坐标系的问题，别的暂且不管。假如我们的主函数的调用形式如下：

```
void main()
{
    //克拉索夫斯基参考椭球
    double a = 6378245.0,  f = 298.3;
    //正算点的纬度和经差
    double B = 0.3836189311, l = 0.0423314484;
    // B = 21°58 47.0845  l = 2°25 31.4880
    double x, y;
    CalEcd(a, f);
    GaussZs(B, l, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
    //算出的x=2433586.692,  y=250547.403
}
```

以上程序的流程为：

(1) 给定一个类型的参考椭球：如长半轴 a，扁率 f

(2) 计算相关的参数：计算偏心率，子午线弧长系数等，这些相对于一个给定的椭球来说，它是一个定值；

(3) 根据给定的 B，l 计算相应的高斯平面坐标；

(4) 输出计算值。

从流程上可看出，GaussZs 函数的原型如下：

```
void GaussZs(double B, double l,
             double * x, double * y)
```

这里 B，l 为给定值，x，y 设定为指针类型，用于返回计算后的值。下面我们来实现这个函数，它的计算流程如下：

(1) 计算子午线弧长；

(2) 计算 x 和 y 坐标；

我们设计该函数如下：

```
void GaussZs(double B, double l,
             double * x, double * y)
{
    double sinB = sin( B );
    double cosB = cos( B );
    double cosB2 = cosB * cosB;
    double cosB4 = cosB2 * cosB2;
    double l2 = l * l;
    double l4 = l2 * l2;

    double g = _eT * cosB;
    double g2 = g * g;
    double g4 = g2 * g2;
    double t = tan(B);
    double t2 = t * t;
    double t4 = t2 * t2;

    //计算子午线弧长
    double X = MeridianArcLength( B );
    double N = _c / sqrt(1.0 + g2);
```

```
    *x = X + N * sinB * cosB * l2 *
    ( 0.5
      + cosB2 * l2 * (5.0 - t2 + 9.0 * g2 + 4.0 * g4) / 24.0
      + cosB4 * l4 * (61.0 - 58.0 * t2
          + t4 + 270.0 * g2 - 330.0 * g2 * t2)  / 720.0
    );
    *y = N * cosB * l *
    ( 1.0
      + cosB2 * l2 * (1.0 - t2 + g2) / 6.0
      + cosB4 * l4 * (5.0 - 18.0 * t2 + t4
          + 14.0 * g2 - 58.0 * t2 * g2) / 120.0
    );
}
```

从函数的实现看，前面的变量只是为了简化后面的计算式，整个函数中没有逻辑判断和循环，因此较为简单。但子午线弧长的计算函数 MeridianArcLength(double) 我们还没实现，根据算法，它的实现如下：

```
double MeridianArcLength(double B)
{
    double sinB = sin(B);
    double cosB = cos(B);
    double sinB3 = sinB * sinB * sinB;
    double sinB5 = sinB * sinB * sinB3;
    double sinB7 = sinB * sinB * sinB5;
    return _A0 * B - cosB * (_B0 * sinB + _C0 * sinB3
           + _D0 * sinB5 + _E0 * sinB7);
}
```

程序中的_A0、_B0、_C0、_D0、_E0 为子午线弧长计算的系数，在此我把它们设为全局变量，在 CalEcd 函数中进行计算。如：

```
double _a, _b, _c, _d, _e, _eT;
double _A0, _B0, _C0 , _D0, _E0;
double _e2, _e4, _e6, _e8;
```

由于这些相关的系数只与椭球的参数有关，我们只在函数 CalEcd 计算一次。其实现如下：

```
void CalEcd(double a, double f)
{
    _a = a; _b = _a * (1.0 - 1.0 / f);

    _e = sqrt(_a * _a - _b * _b) / _a;
    _eT = sqrt(_a * _a - _b * _b) / _b;
    _c = _a / _b * _a;
    _d = _b / _a * _b;

    _e2 = _e * _e;
    _e4 = _e2 * _e2;
    _e6 = _e2 * _e4;
    _e8 = _e2 * _e6;

    _A0 = _d * (  1    + 3.0 /4.0 * _e2
                  + 45.0 / 64.0 * _e4
              + 175.0 / 256.0 * _e6
              + 11025.0 / 16384.0 * _e8 );
    _B0 = _d * (         3.0 /4.0 * _e2
                  + 45.0 / 64.0 * _e4
              + 175.0 / 256.0 * _e6
              + 11025.0 / 16384.0 * _e8);
    _C0 = _d * (         15.0 / 32.0 * _e4
              + 175.0 / 384.0 * _e6
                      + 3675.0 /  8192.0 * _e8);
    _D0 = _d * (         35.0 /  96.0 * _e6
                  + 735.0 /  2048.0 * _e8);
    _E0 = _d * (     315.0 /  1024.0 * _e8 );
}
```

至此，我们完成了高斯平面直角坐标的正算功能的程序设计。为了利用 C++ 语言增强了的 C 功能，我们把文件的扩展名由 c 改为 cpp，以利

用 C++ 编译器进行严格检查。

为了进行源代码一级的复用，我们在工程中新加入一个头文件（.h），将上面除主函数 main() 之外的所有代码剪切到头文件中（如头文件为 Gauss.h），在主函数所在的文件中将头文件包含在内，即在文件顶端加入：`#include "Gauss.h"`，在别的地方我们就可继续使用这些函数。

为了直接利用纬度、经度和中央子午线的经度计算高斯平面坐标，我们可以利用函数重载的功能继续实现坐标正算：

```
void GaussZs(double B, double L, double L0,
             double * x, double * y)
{
    double l = L - L0;
    GaussZs(B, l, x, y);
}
```

这里的实现很简单，只是计算了经差，调用了原来的正算函数。我们就可在主函数中作如下形式的调用了：

```
void main()
{
    //21°58 47.0845 的弧度形式
    double B = 0.3836189311;
    // 113°25 31.4880 的弧度形式
    double L = 1.9796469181;
    // 111°的弧度形式
    double L0 = 1.9373154697;
    //54//x:2433586.692, y:250547.403
    GaussZs(B, L, L0, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
}
```

至此，我们已经完成了高斯坐标正算的全部计算了。以上的函数调用中的角度均使用了弧度的形式，利用前面所讲的角度化弧度的函数，我们可以以下形式的调用：

```
void main()
```

```
{
    //21°58 47.0845
    double B = DMStoRAD(21.58470845);
    //113°25 31.4880
    double L = DMStoRAD (113.25314880);
    double L0 = DMStoRAD(111.0); //111°
    //54//x:2433586.692, y:250547.403
    GaussZs(B, L, L0, &x, &y);
    printf("x = %lf, y = %lf \n", x, y);
}
```

同样，坐标反算的程序设计的实现方法也基本一样。由于面向过程的程序设计不是我们的重点，就将这部分的实现放入面向对象中。

7.3 * 面向对象的高斯坐标正反算程序设计

在前面的程序设计中，我们将椭球参数用全局变量表示，为了消除这些全局变量的影响，我们采用类的形式对其进行封装，封装的形式如下：

```
class CEarth
{
private:
    double _a, _b, _c, _d, _e, _eT;
    double _A0, _B0, _C0 , _D0, _E0;
    double _e2, _e4, _e6, _e8, _e10;
};
```

将它们设为 private，防止类以外的代码随意访问它们。再将上面实现的正算等函数作为其成员函数，则形式为：

```
class CEarth
{
private:
    double _a, _b, _c, _d, _e, _eT;
    double _A0, _B0, _C0 , _D0, _E0;
```

```
    double _e2, _e4, _e6, _e8, _e10;
public:
    CEarth();//构造函数
    virtual ~CEarth();  //析构函数
    //计算相关系数
    void CalEcd(double a, double alf);
    //正算
    void GsZs(double l, double B,
              double& x, double& y);
    void GsZs(double L, double B, double L0,
              double& x, double& y);
    //反算
    void GsFs(double x, double y,
              double& l, double& B);
    void GsFs(double x, double y, double L0,
              double & L, double & B);
    //子午线收敛角
    double MeridianConvergentAngleByBl(
              double l, double B);
    double MeridianConvergentAngleByxy(
              double x, double y);
    //子午线弧长
    double MeridianArcLength(double B);
     //底点纬度
    double Bf(double x);
};
```

在类的成员函数定义中，将传进的参数用引用的形式加 const 进行限定，同时用引用取代指针进行函数返回值，如 GsZs，GsFs 函数。

在这里，我们看看反算的实现。在反算的计算流程中，经差 l 的计算很简单，直接计算即可，但纬度的计算需要先计算底点纬度，由于我们的程序是针对多种椭球的，底点纬度的数值计算公式是不能使用的。我们用子午线弧长计算公式进行迭代计算，在开始迭代时，取弧长的初值为x / _A0，其具体实现如下：

```
double CEarth::Bf(double x)
{
    double Bf0 = x / _A0; //子午线弧长的初值
    int i = 0;
    while( i < 10000 )//设定迭代次数
    {
        double sinBf = sin(Bf0);
        double cosBf = cos(Bf0);
        double sinBf3 = sinBf * sinBf * sinBf;
        double sinBf5 = sinBf * sinBf * sinBf3;
        double sinBf7 = sinBf * sinBf * sinBf5;

        double Bf = (  x
                 + cosBf * ( _B0 * sinBf
                               + _C0 * sinBf3
                    + _D0 * sinBf5
                    + _E0 * sinBf7)
            ) / _A0;
        if( fabs(Bf - Bf0) < 1e-10) //计算精度
            return Bf;
        else
        {
            Bf0 = Bf;
            i++;
        }
    }
    return -1e12;
}
```

反算的实现如下：

```
void CEarth::GsFs(double x, double y,
                  double & l, double & B)
{
    double Bf0 = Bf( x );
```

```
double cosBf = cos( Bf0 );

double gf = _eT * cosBf;
double gf2 = gf * gf;
double gf4 = gf2 * gf2;

double tf = tan(Bf0);
double tf2 = tf * tf;
double tf4 = tf2 * tf2;

double Nf = _c / sqrt(1.0 + gf2);
double Nf2 = Nf * Nf;
double Nf4 = Nf2 * Nf2;

double y2 = y * y;
double y4 = y2 * y2;

l = y / (Nf * cosBf) *
( 1.0
  - y2 / (6.0 * Nf2) * (1.0 + 2.0 * tf2 + gf2)
      + y4 / (120.0 * Nf4 ) * (5.0
  + 28.0 * tf2
      + 24.0 * tf4
  + 6.0 * gf2
  + 8.0 * gf2 * tf2)
);
B = Bf0 - y2 / Nf2 * tf * 0.5 *
  ( (1.0 + gf2)
         - y2 * (5.0
                 + 3.0 * tf2
                 + 6.0 * gf2
                 - 6.0 * tf2 * gf2
                 - 3.0 * gf4
```

```
                    + 9.0 * gf4 * tf4
                   ) / (12.0 * Nf2)
            + y4 * (61.0
                    + 90.0 * tf2
                    + 45.0 * tf4
                    + 107.0 * gf2
                    + 162.0 * gf2 * tf2
                    + 45.0 * gf2 * tf4
                   ) / ( 360 * Nf4)
     );
}

//L: 经度（弧度）, B: 纬度（弧度）， L0: 中央子午线经度
void CEarth::GsFs(double x, double y, double L0,
                  double& L, double& B)
{
    double l;
    GsFs(x, y, l, B);
    L = L0 + l;
}
```

在以上实现中，由于我们的成员变量均为 private 类型，而构造函数为默认的形式，无法将椭球的参数值传递进去。由于要实现 54、80 和自定义椭球的坐标系，相应的解决方法有：

1．将默认的构造函数改为 CEarth (double a, double alf) 形式，将不同的椭球参数传给类。比如要建立 54 坐标系，只需要如下作即可：

```
CEarth earth54(6378245.0, 298.3);
CEarth * pEarth54 = new CEarth(6378245.0, 298.3);
```

这种形式很直观，但缺点也很明显，对于已知的常用椭球，编程人员每次均要提供其具体的参数值，很不方便，也容易出错。

2．利用类厂的方法，将生成已知的常用椭球定义为静态成员函数，同时将构造函数声明为 private 类型，防止象方法 1 那样直接访问。这种方法的优点是显而易见的，它的实现如下：

在类 CEarth 中将默认的构造函数的访问域改为：

```
private:
    CEarth();
    void CalEcd(double a, double alf);
```

同时 CalEcd 函数计算椭球内部的基本系数，不需要外部程序访问，也将其定义为 private。

同时在类中声明以静态成员指针变量：

```
private:
    static CEarth * _pEarth;
```

在类的实现文件前面进行初始化：

```
CEarth* CEarth::_pEarth = NULL;
```

再在类中定义用于创建椭球的静态成员函数：

```
public:
    static CEarth * CreateEarth54();
    static CEarth * CreateEarth80();
    static CEarth * CreateEarthCustomize(double a, double alf);
```

它们的实现为：

```
CEarth * CEarth::CreateEarth54()
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth->CalEcd(6378245.0, 298.3);
    return _pEarth;
}
CEarth * CEarth::CreateEarth80()
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth->CalEcd(6378140.0, 298.257);
    return _pEarth;
}
CEarth * CEarth::CreateEarthCustomize(double a, double alf)
```

```
{
    if(_pEarth == NULL) _pEarth = new CEarth();
    _pEarth = new CEarth();
    _pEarth->CalEcd(a, alf);
    return _pEarth;
}
```

从它们的实现看，是用 new 的方式生成了椭球对象，为了避免内存泄漏，类的析构函数中，负责将申请的内存进行释放，它的实现如下：

```
CEarth::~CEarth()
{
    if(_pEarth != NULL)
    {
        delete _pEarth;
        _pEarth = NULL;
    }
}
```

主函数调用的形式为：

```
void main()
{
    double x, y;
CEarth * pEarth = CEarth::CreateEarth54();
    double B = zx::CSurMath::DmsToRad(21.58470845);
    double l = zx::CSurMath::DmsToRad(113.25314880)
                  - zx::CSurMath::DmsToRad(111);
    pEarth->GsZs( l, B, x, y);
printf("x = %lf, y = %lf \n", x, y);
}
```

现在我们基本上用面向对象的方法完成了高斯投影正反算的问题了。

7.4 实现换带计算和文件读写

7.4.1 换带计算

换带计算的基本方法：已知某一带（中央子午线经度已知 oldL0）的点坐标（oldX, oldY），利用高斯反算计算出大地坐标（B，L），再根据新带的中央子午线经度（newL0），高斯正算计算新带中的高斯平面坐标（newX, newY）。用算法描述如下：

1. $B\ L\ = GsFs(oldX, oldY, oldL0)$
2. $(newX, newY) = GsZs(B, L, newL0)$

用一函数描述它为：

```
void CEarth::GsHd(double oldX, double oldY,
                  double oldL0,  double newL0,
                  double& newX, double& newY)
{
    double B, L;
    GsFs(oldX, oldY, oldL0, L, B);
    GsZs(L, B, newL0, newX, newY);
}
```

注意：高斯换带计算一般是指同一参考椭球的不同带之间的换算，不同的参考椭球的坐标系是不能采用这种方法的。

计算示例如下：

```
void main()
{
    double oldX = 3275110.535;
    double oldY = 235437.233;
    double oldL0 = zx::CSurMath::DmsToRad(117);
    double newL0 = zx::CSurMath::DmsToRad(120);
    double newX, newY;

   CEarth * pEarth = CEarth::CreateEarth54();
   pEarth->GsHd(oldX, oldY, oldL0, newL0, newX, newY);
   printf("newX = %lf,  newY = %lf \n", newX,  newY);
```

```
   // 3272782.315, -55299.545
}
```

7.4.2 多点计算和文件读写

以上我们的算法和测试验证程序均是针对一个点而言的，如果我们在一个文本形式的数据文件里存放有多个点，怎么计算呢？

1、我们设计正算的数据文件格式为：

```
成果文件名

转换点个数 ~ 中央子午线经度

点名 ~ 大地纬度 ~ 大地经度
```

示例数据文件为：

```
BLXY.txt
3  111
p1  21.58470845  113.25314880
p2  31.04416832  111.47248974
p3  30.45254425  111.17583596
```

则设计函数为：

```
struct PntInfo
{
    char name[10];
    double B, L;
    double x, y;
};
void Zs()
{
    double L0;//所在带的中央子午线经度
    int n;//转换点的个数
    char cg[256];//成果文件名
    FILE *in;
```

```
    //读取文本文件的数据
    in = fopen("BLtoXY.txt", "r"); //打开已知文件
    fscanf(in,"%s",cg); //首先读入成果文件名称
    fscanf(in,"%d %lf", &n, &L0); //转换点的个数  中央子午线经度
    PntInfo * pnts = new PntInfo[n];//动态数组
    for(int i = 0; i < n; i++)//循环读入点名B、L
     fscanf(in, "%s %lf %lf", pnts[i].name, &pnts[i].B, &pnts[i].L);
    fclose(in);
    //以下进行正算计算
    CEarth * pEarth = CEarth::CreateEarth54();
    for(i = 0; i < n; i++)
        pEarth->GsZs(zx::CSurMath::DmsToRad(pnts[i].L),
         zx::CSurMath::DmsToRad(pnts[i].B),
             zx::CSurMath::DmsToRad(L0),
         pnts[i].x, pnts[i].y);
    //将计算后的数据写到成果文件中
    FILE * out;
    out = fopen(cg, "w");
    fprintf(out, "大地坐标(B,L)=====>国家坐标(X,Y)\n ");
    fprintf(out,"中央子午线经度: L0 = %lf\n", L0);
    fprintf(out,"序号 点名 B      L ===> X坐标(m)  Y坐标(m)\n");
    for(i = 0; i < n; i++)
    {
    fprintf(out," %3d %s %lf %lf %11.3f %11.3f\n",
        i+1, pnts[i].name, pnts[i].B, pnts[i].L,
            pnts[i].x, pnts[i].y );
    }
    fclose(out);
    delete[] pnts;//释放申请的内存
}
```

同样也可实现多点的数据文件反算和换带计算

反算的数据文件格式设为：XYBL.txt

```
3   111
```

```
p1      2433586.692   250547.403
p2      3439978.970    75412.872
p3      3404139.839    28680.571
```

换带计算的数据文件格式设为：

```
XYBL.txt
3   111 112
p1      2433586.692   250547.403
p2      3439978.970    75412.872
p3      3404139.839    28680.571
```

相应的实现请参照正算实现。

7.5 小结

我们从一个逻辑结构不太明显的程序开始，将其改为一个结构较好的面向过程的程序。并从一个算法开始逐步实现了整个程序。最后用面向对象的方法将其改写并实现了全部功能。最后的程序从结构上看明显要比第一个程序要好，且更易维护和扩充。当然高斯换带计算较为简单，我们的程序实现同样也较为简单。从这个例子，我们知道了怎样设计结构良好的程序。

第 8 章　平面坐标系统之间的转换

在某些工程中，由于不知道新旧两种坐标系的建立方法或参数，因此无法用换带计算的方法进行坐标转换。如果知道某些点在两个坐标系中的坐标值，我们就可以采用一些近似的转换方法将其它的点也转换到新坐标系中，求出其坐标值。尤其对于较低等级的大量控制点来说，采用这些近似方法，能够快速得到转换结果。

8.1　原理和数学模型

8.1.1　原理

这些方法的实质是根据新旧网的重合点（又称为公共点）的坐标值之差，按一定的规律修正旧网的各点坐标值，使旧网最恰当的配合到新网内。修正时因不合观测值改正数平方和为最小的原则，故为近似方法。

常用的方法有简单变换方法（又称赫尔默特法或相似变换法）、仿射变换法、正形变换法等。在这里我们主要讲解简单变换法。

8.1.2　相似变换法的数学模型

实质是使旧网坐标系平移、旋转和进行尺度因子改正，将旧网配合到新网上。因旧网形状保持不变，故称为平面相似变换法。

变换方程为：

$$\left.\begin{aligned} x &= a + k(x' \cos\alpha + y' \sin\alpha) \\ y &= b + k(-x' \sin\alpha + y' \cos\alpha) \end{aligned}\right\}$$

式中 a，b 表示平移，α 是旧网 x' 轴逆转至新网 x 轴的转角，k 为尺度因子。这些变换参数是未知的，要根据新旧网公共点上的已知坐标 x，y 和 x'、y' 求解确定。

因此必须至少有两个公共点，列出四个方程式，解算出这四个未知参数值。如果具有两个以上的公共点时，就应该应用最小二乘平差方法，求解最或是参数值。

为解算出这些参数，我们引入参数 c、d：

$c = k\cos\alpha$，$d = k\sin\alpha$

将公式转换为：

$$\left.\begin{aligned} x &= a + x'c + y'd \\ y &= b + y'c - x'd \end{aligned}\right\}$$

由于新旧网都存在测量误差，设新旧坐标 x，y 和 x'，y' 的误差分别为 v_x，v_y 和 $v_{x'}$，$v_{y'}$，因此上式改写为：

$$\left.\begin{aligned} x + v_x &= a + (x' + v_{x'})c + (y' + v_{y'})d \\ y + v_y &= b + (y' + v_{y'})c - (x' + v_{x'})d \end{aligned}\right\}$$

设：

$$\left.\begin{aligned} n_x &= -v_x + cv_{x'} + dv_{y'} \\ n_y &= -v_y - dv_{x'} + cv_{y'} \end{aligned}\right\}$$

则有：

$$\left.\begin{aligned} -n_x &= a + x'c + y'd - x \\ -n_y &= b + y'c - x'd - y \end{aligned}\right\}$$

若有 r 个新旧网的公共点，则可组成 r 对方程：

$$V = BX - l$$

上式即为参数平差时的方程，l 代表观测向量，V 代表改正数向量，B 代表系数矩阵，X 是参数向量。它们的值为：

$$\mathbf{V} = \begin{pmatrix} -n_{x1} \\ -n_{y1} \\ \vdots \\ -n_{xr} \\ -n_{yr} \end{pmatrix} \quad \mathbf{B} = \begin{pmatrix} 1 & 0 & x'_1 & y'_1 \\ 0 & 1 & y'_1 & -x'_1 \\ \cdots & \cdots & \cdots & \cdots \\ 1 & 0 & x'_r & y'_r \\ 0 & 1 & y'_r & -x'_r \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \quad \mathbf{l} = \begin{pmatrix} x_1 \\ y_1 \\ \vdots \\ x_r \\ y_r \end{pmatrix}$$

根据最小二乘原理 $V^TV = min$ 可得到法方程：

$$B^TBX - B^Tl = 0$$

解法方程可求得 a、b、c、d 的值：

$$X = (B^TB)^{-1}(B^Tl)$$

旋转角 α 和尺度比 k 为：

$$\alpha = \arctan\frac{d}{c}$$

$$k = \sqrt{c^2 + d^2}$$

之后，就可计算旧网中所有待转换点的新坐标。

8.2 程序功能设计

从数学模型可看出，程序的关键是矩阵解算, 其次是数据的组织问题。我们首先考虑数据文件的组织格式。

8.2.1 数据文件和成果文件格式

由于程序的功能较为单一，数据文件的格式也较为简单。我们设计格式如下：

公共点个数
待转换点个数
//公共点在旧坐标系中的坐标
点名 ␣x坐标 ␣y坐标
. . . . . .
//公共点在新坐标系中的坐标
点名 ␣x坐标 ␣y坐标
. . . . . .
//待转换点在旧坐标系中的坐标
点名 ␣x坐标 ␣y坐标
. . . . . .

我们设计成果文件的格式如下：

```
公共点个数 待转换点个数
===公共点在旧坐标系中的坐标===
点名 x坐标 y坐标
.................
===公共点在新坐标系中的坐标===
点名 x坐标 y坐标
.................
===待转换点在旧坐标系中的坐标===
点名 x坐标 y坐标
.................
转换参数(平移量a)、(平移量b)、(旋转角 )、(尺度比k)
转换的精度
===待转换点在新坐标系中的坐标===
点名 x坐标 y坐标
.................
```

8.2.2 程序流程

根据以上分析，程序流程如下：

1. 读取公共点旧坐标
2. 读取公共点新坐标
3. 组成误差方程式
4. 解算参数向量
5. 解算待定点的坐标
6. 将计算成果写入文件

8.2.3 主要功能设计

为了实现以上功能，我们需要设计一个结构 (或类) 用于表示点，设计如下：

```
struct Pnt
```

```
{
    char _name[11];
    double _oldx, _oldy, _newx, _newy;
};
```

同时，我们设计另一个类 $CoordSys$ 来完成相应的其它功能：

```
class CoordSys
{
private:
    double _a, _b, _alpha, _k;
    int _n0, _ni;
    Pnt * _pPnts;
public:
    CoordSys(void);
    ~CoordSys(void);

    Pnt * GetPnt(const char * name);
    void ReadPntCount(FILE * in);
    void ReadCommontPntOldXY(FILE * in);
    void ReadCommontPntNewXY(FILE * in);
    void ReadUnKnownPnt(FILE * in);

    void ReadData(FILE * in);

    void WritePntCount(FILE * out);
    void WriteCommontPntOldXY(FILE * out);
    void WriteCommontPntNewXY(FILE * out);
    void WriteUnKnownPntOldXY(FILE * out);
    void WriteUnKnownPntNewXY(FILE * out);
    void WriteABK(FILE * out);

    void WriteData(FILE * out);
    void NegativeMatrix(double A[], double B[], int n);
    void Cal();
```

```
};

CoordSys::CoordSys(void)
{
    _a = _b = _alpha = _k = 0;
    _n0 = _ni = 0;
    _pPnts = NULL;
}

CoordSys::~CoordSys(void)
{
    if(_pPnts != NULL)
    {
        delete[] _pPnts;
        _pPnts = NULL;
    }
}

void CoordSys::ReadPntCount(FILE * in)
{
    fscanf(in,"%d", &_n0); //读入公共点个数
    fscanf(in,"%d", &_ni); //读入待定点个数
    _pPnts = new Pnt[_n0 + _ni];
}

Pnt * CoordSys::GetPnt(const char * name)
{
    for(int i = 0; i<_n0; i++)
    {
        if(strcmp(_pPnts[i]._name, name) == 0)
            return &_pPnts[i];
    }
    return NULL;
```

```
}

void CoordSys::ReadCommontPntOldXY(FILE * in)
{
    char line[256];
    fscanf(in, "%s", line);
    for(int i=0; i<_n0; i++)
    {
        fscanf(in, "%s %lf %lf", _pPnts[i]._name,
             &_pPnts[i]._oldx, &_pPnts[i]._oldy);
    }
}

void CoordSys::ReadCommontPntNewXY(FILE * in)
{
    char line[256];
    fscanf(in, "%s", line);
    for(int i=0; i<_n0; i++)
    {
        char name[11];
        double x, y;

        fscanf(in, "%s %lf %lf", name, &x, &y);
        Pnt * pPnt = GetPnt(name);//此处未加容错处理
        pPnt->_newx = x;
        pPnt->_newy = y;
    }
}

void CoordSys::ReadUnKnownPnt(FILE * in)
{
    char line[256];
    fscanf(in, "%s", line);
```

```
    for(int i=0; i<_ni; i++)
    {
        fscanf(in, "%s %lf %lf", _pPnts[i+_n0]._name,
        &_pPnts[i+_n0]._oldx, &_pPnts[i+_n0]._oldy);
    }
}

void CoordSys::WritePntCount(FILE * out)
{
    fprintf(out, "公共点个数: %d\n", _n0);
    fprintf(out, "待定转换点个数: %d\n", _ni);
}
void CoordSys::WriteCommontPntOldXY(FILE * out)
{
    fprintf(out, "===公共点在旧坐标系中的坐标===\n");
    for(int i=0; i<_n0; i++)
        fprintf(out, "%s\t%lf\t%lf\n", _pPnts[i]._name,
                    _pPnts[i]._oldx, _pPnts[i]._oldy);
}
void CoordSys::WriteCommontPntNewXY(FILE * out)
{
    fprintf(out, "===公共点在新坐标系中的坐标===\n");
    for(int i=0; i<_n0; i++)
        fprintf(out, "%s\t%lf\t%lf\n", _pPnts[i]._name,
                    _pPnts[i]._newx, _pPnts[i]._newy);
}
void CoordSys::WriteUnKnownPntOldXY(FILE * out)
{
    fprintf(out, "===待转换点在旧坐标系中的坐标===\n");
    for(int i=0; i<_ni; i++)
        fprintf(out, "%s\t%lf\t%lf\n", _pPnts[i+_n0]._name,
                 _pPnts[i+_n0]._oldx, _pPnts[i+_n0]._oldy);
}
```

```
void CoordSys::WriteUnKnownPntNewXY(FILE * out)
{
    fprintf(out, "===待转换点在新坐标系中的坐标===\n");
    for(int i=0; i<_ni; i++)
        fprintf(out, "%s\t%lf\t%lf\n", _pPnts[i+_n0]._name,
                _pPnts[i+_n0]._newx, _pPnts[i+_n0]._newy);

}
void CoordSys::WriteABK(FILE * out)
{
    fprintf(out, "转换参数: a=%lf, b=%lf, =%lf, k=%lf\n",
                            _a,    _b, _alpha,      _k);
}

void CoordSys::ReadData(FILE * in)
{
    ReadPntCount(in);
    ReadCommontPntOldXY(in);
    ReadCommontPntNewXY(in);
    ReadUnKnownPnt(in);
}

void CoordSys::WriteData(FILE * out)
{
    WritePntCount(out);
    WriteCommontPntOldXY(out);
    WriteCommontPntNewXY(out);
    WriteUnKnownPntOldXY(out);
    WriteABK(out);
    WriteUnKnownPntNewXY(out);
}

//解法方程AX = B, A:n×n B:n×1
```

```
void CoordSys::NegativeMatrix(double A[], double B[], int n)
{
    for(int k = 0; k < n-1; k++)
    {
        for(int i = k+1; i < n; i++)
        {
            A[i*n + k] /= A[k*n + k];
            for(int j = k+1; j < n; j++)
            {
                A[i*n + j] -= A[i*n + k] * A[k*n + j];
            }
            B[i] -= A[i*n + k] * B[k];
        }
    }

    B[n-1] /= A[(n-1)*n + (n-1)];
    for(int i= n-2; i >= 0; i--)
    {
        double s = 0.0;
        for(int j = i+1; j < n; j++)
        {
            s += A[i*n + j] * B[j];
        }
        B[i] = (B[i]-s) / A[i*n + i];
    }
}
void CoordSys::Cal()
{
    double * B = new double[2*_n0 * 4];
    double * l = new double[2*_n0];
    double * N = new double[4*4];
    double * U = new double[4];
```

```
for(int i=0; i<_n0; i++)
{
    B[(2*i)* 4 + 0] = 1.0;
    B[(2*i)* 4 + 1] = 0.0;
    B[(2*i)* 4 + 2] = _pPnts[i]._oldx;
    B[(2*i)* 4 + 3] = _pPnts[i]._oldy;
    l[2*i] = _pPnts[i]._newx;

    B[(2*i +1)* 4 + 0] = 0.0;
    B[(2*i +1)* 4 + 1] = 1.0;
    B[(2*i +1)* 4 + 2] = _pPnts[i]._oldy;
    B[(2*i +1)* 4 + 3] = -_pPnts[i]._oldx;
    l[2*i +1] = _pPnts[i]._newy;
}

for(int k=0; k<4; k++)
{
    for(int j=0; j<4; j++)
    {
        N[k*4+j] = 0.0;
        for(int i=0; i<2*_n0; i++)
        {
            N[k * 4 + j] += B[i*4 + k] * B[i*4 + j];
        }
    }

    U[k] = 0.0;
    for(int i=0; i<2*_n0; i++)
        U[k] += B[i*4 + k] * l[i];
}
NegativeMatrix(N, U, 4);

_a = U[0];
```

```
    _b = U[1];
    _alpha = RadToDms(atan2(U[3], U[2]));
    _k= sqrt(U[3]*U[3]+U[2]*U[2]);

    for(int i=0; i<_ni; i++)
    {
        _pPnts[_n0+i]._newx = U[0] + U[2]* _pPnts[_n0+i]._oldx
                                   + U[3]*_pPnts[_n0+i]._oldy;
        _pPnts[_n0+i]._newy = U[1] + U[2]* _pPnts[_n0+i]._oldy
                                   - U[3]*_pPnts[_n0+i]._oldx;
   }
   delete[] N;
   delete[] l;
   delete[] B;
}

void main()
{
    CoordSys coordSys;
    FILE * in, * out;

    in = fopen("xytoxy.txt", "r"); //打开已知文件
    coordSys.ReadData(in);
    fclose(in);

    coordSys.Cal();

    out = fopen("toxy.txt", "w");
    coordSys.WriteData(out);
    fclose(out);
}
```

计算示例数据文件内容：

```
3
```

```
6
//公共点在旧坐标系中的坐标
103 3927002.191 449256.848
100 3928471.180 451589.920
102 3928308.824 446388.500
//公共点在新坐标系中的坐标
103 327156.644 485664.463
100 328638.283 487989.669
102 328447.816 482788.977
//待转换点在旧坐标系中的坐标
11 3927202.638 448247.421
07 3928890.412 449425.214
15 3927466.357 446917.643
103 3927002.191 449256.848
100 3928471.180 451589.920
102 3928308.824 446388.500
```

计算成果文件内容：

```
公共点个数：3
待定转换点个数：6
===公共点在旧坐标系中的坐标===
103 3927002.191000 449256.848000
100 3928471.180000 451589.920000
102 3928308.824000 446388.500000
===公共点在新坐标系中的坐标===
103 327156.644000 485664.463000
100 328638.283000 487989.669000
102 328447.816000 482788.977000
===待转换点在旧坐标系中的坐标===
11 3927202.638000 448247.421000
07 3928890.412000 449425.214000
15 3927466.357000 446917.643000
103 3927002.191000 449256.848000
100 3928471.180000 451589.920000
```

```
102 3928308.824000 446388.500000
转换参数：a=-3602385.714666, b=57613.407531,
  =0.183446, k=1.000043
===待转换点在新坐标系中的坐标===
11 327351.643066 484653.926898
07 329045.829449 485822.634190
15 327608.184475 483322.685785
103 327156.644527 485664.465940
100 328638.281925 487989.667546
102 328447.816548 482788.975515
```

第 9 章　测量平差程序设计

9.1　数学模型

9.2　控制网的数据结构

9.3　平差计算的流程

9.4　示例程序

```
// Net.h
#pragma once
struct CtrPnt;

struct Observe
{
   CtrPnt * target;
   char type[2];
   double value;
};

struct CtrPnt
{
   char name[17];
   double x, y, z;
   char attr[3];
   Observe observe[20];
```

```
    int n;//观测值的个数
};

struct Net
{
    double m0, a, b;
    CtrPnt ctrPnts[100];
    int n, n0;//总点数,  已知点数
};

void ReadData(Net * net, FILE * in);
CtrPnt * GetPnt(Net * pnet, char * name);
CtrPnt * AddPntToNet(Net * pnet, char * name);
void WriteData(Net * pnet, FILE * out);
void WriteToNasew(Net * pnet, FILE * out);

// Net.cpp
#include "stdafx.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "Net.h"

CtrPnt * GetPnt(Net * pnet, char * name)
{
    int i = 0;
    for(i=0; i<pnet->n; i++)
    {
        if(strcmp(pnet->ctrPnts[i].name, name) == 0)
            return &pnet->ctrPnts[i];
    }
```

```
    return NULL;
}

CtrPnt * AddPntToNet(Net * pnet, char * name)
{
    int i = pnet->n;
    strcpy(pnet->ctrPnts[i].name, name);
    strcpy(pnet->ctrPnts[i].attr, "00");
    pnet->ctrPnts[i].x = 0;
    pnet->ctrPnts[i].y = 0;
    pnet->ctrPnts[i].z = 0;
    pnet->n++;
    return &pnet->ctrPnts[i];
}

void ReadData(Net * pnet, FILE * in)
{
    pnet->n = pnet->n0 = 0;

    char buffer[256];

    fgets(buffer, 255, in);
  sscanf(buffer, "%lf %lf %lf", &pnet->m0, &pnet->a, &pnet->b);

    int i = 0, j = 0;
    char name[17], attr[3], type[3];
    double x, y, z, value;

    while(!feof(in))
    {
        fgets(buffer, 255, in);
        if(feof(in)) break;
        if(sscanf(buffer, "%s %s %lf %lf %lf",
```

```
            name, attr, &x, &y, &z) == 5)
        {
            strcpy(pnet->ctrPnts[i].name, name);
            strcpy(pnet->ctrPnts[i].attr, attr);
            pnet->ctrPnts[i].x = x;
            pnet->ctrPnts[i].y = y;
            pnet->ctrPnts[i].z = z;
            if(attr[0] == '1')
                pnet->n0++;
            i++;
        }
        else
            break;
    }
    pnet->n = i;//已知点个数

    while(!feof(in))
    {
        sscanf(buffer, "%s", name);//读测站名
        CtrPnt * pPnt = GetPnt(pnet, name);
        if(pPnt == NULL)
            pPnt = AddPntToNet(pnet, name);
        j = 0;
        while(!feof(in))//读测站观测数据
        {
            fgets(buffer, 255, in);
            if(feof(in)) return;
         if(sscanf(buffer, "%s %s %lf", name, type, &value) == 3)
            {
                CtrPnt * pTarget = GetPnt(pnet, name);
                if (pTarget == NULL)
                    pTarget = AddPntToNet(pnet, name);
                pPnt->observe[j].target = pTarget;
```

```
                strcpy(pPnt->observe[j].type, type);
                pPnt->observe[j].value = value;
                pPnt->n = ++j;
            }
            else
                break;
        }
    }
}

void WriteData(Net * pnet, FILE * out)
{
    int i=0, j=0;
    fprintf(out, "%lf %lf %lf\n", pnet->m0, pnet->a, pnet->b);
    for(i=0; i<pnet->n; i++)
    {
        fprintf(out, "%s %s %lf %lf %lf\n",
            pnet->ctrPnts[i].name,
            pnet->ctrPnts[i].attr,
            pnet->ctrPnts[i].x,
            pnet->ctrPnts[i].y,
            pnet->ctrPnts[i].z);
    }

    for(i=0; i<pnet->n; i++)
    {
        fprintf(out, "%s\n", pnet->ctrPnts[i].name);
        for(j=0; j<pnet->ctrPnts[i].n; j++)
        {
            fprintf(out, "\t%s %s %lf\n",
                pnet->ctrPnts[i].observe[j].target->name,
                pnet->ctrPnts[i].observe[j].type,
                pnet->ctrPnts[i].observe[j].value);
```

```
        }
    }
}

void WriteToNasew(Net * pnet, FILE * out)
{
    int i=0, j=0;
    fprintf(out, "<<\n");
    for(i=0; i<pnet->n; i++)
    {
        fprintf(out, "%-6s%2s%13.4f%13.4f%12.4f\n",
            pnet->ctrPnts[i].name,
            pnet->ctrPnts[i].attr,
            pnet->ctrPnts[i].x,
            pnet->ctrPnts[i].y,
            pnet->ctrPnts[i].z);
    }
    fprintf(out, "\n<<\n");

    for(i=0; i<pnet->n; i++)
    {
        fprintf(out, "\n%-6s", pnet->ctrPnts[i].name);
        for(j=0; j<pnet->ctrPnts[i].n; j++)
        {
            fprintf(out, "\t%-6s %2s%13.6f\n",
                pnet->ctrPnts[i].observe[j].target->name,
                pnet->ctrPnts[i].observe[j].type,
                pnet->ctrPnts[i].observe[j].value);
        }
    }
}
```

测试代码：

```
//Adjust.cpp
#include "stdafx.h"
#include <stdio.h>
#include "Net.h"

int main(int argc, char* argv[])
{
   Net net;
   FILE * in, *out;

   in = fopen("ZAtest.txt", "r");
   ReadData(&net, in);
   fclose(in);

   out = fopen("ZAResult.txt", "w");
   //WriteData(&net, out);//输出为原始数据文件格式
   WriteToNasew(&net, out);//输出为清华山维平差软件格式
   fclose(out);

   printf("n0= %d n= %d\n", net.n0, net.n);
   return 0;
}
```

示例数据文件

```
5.000000    2.000000    3.000000
C00 10  40100.000000 41010.000000 0.000000
C01 10  39973.150000 40916.459000 0.000000
C00
   C01 C0  267.393500
   D05 C0  0.000000
   D05 D0  141.469000
   D22 C0  53.495500
   D22 D0  203.930000
C01
```

```
   C00 C0  0.000000
   D01 C0  226.070400
   D01 D0  104.479000
   D32 C0  52.432400
   D32 D0  155.163000
D01
   C01 C0  0.000000
   C01 D0  104.486000
   D02 C0  195.103200
   D02 D0  74.685000
D02
   D01 C0  0.000000
   D01 D0  74.684000
   D03 C0  216.595600
   D03 D0  160.133000
D03
   D02 C0  0.000000
   D02 D0  160.128000
   D04 C0  210.372800
   D04 D0  105.980000
D04
   D03 C0  0.000000
   D03 D0  105.984000
   D07 C0  100.391800
   D07 D0  140.681000
   D08 C0  202.135300
   D08 D0  183.420000
   D09 C0  275.521800
   D09 D0  152.619000
D05
   C00 C0  211.223800
   C00 D0  141.468000
   D09 C0  0.000000
```

```
   D09 D0  151.954000
D07
   D04 C0  0.000000
   D04 D0  140.684000
   D10 C0  124.413600
   D10 D0  112.559000
D08
   D04 C0  250.152400
   D04 D0  183.424000
   D19 C0  0.000000
   D19 D0  112.028000
   D20 C0  95.165500
   D20 D0  143.676000
D09
   D04 C0  0.000000
   D04 D0  152.616000
   D05 C0  196.104200
   D05 D0  151.954000
D10
   D07 C0  0.000000
   D07 D0  112.542000
   D12 C0  137.413600
   D12 D0  87.838000
D12
   D10 C0  0.000000
   D10 D0  87.882000
   D13 C0  232.093800
   D13 D0  196.831000
D13
   D12 C0  0.000000
   D12 D0  196.817000
   D14 C0  282.450000
   D14 D0  140.582000
```

```
D14
   D13 C0  0.000000
   D13 D0  140.565000
   D15 C0  205.071400
   D15 D0  253.424000
D15
   D14 C0  0.000000
   D14 D0  253.421000
   D16 C0  226.350100
   D16 D0  95.787000
D16
   D15 c0  0.000000
   D15 D0  95.772000
   D17 C0  225.302100
   D17 D0  117.724000
D17
   D16 c0  0.000000
   D16 D0  117.699000
   D18 C0  146.432600
   D18 D0  172.034000
D18
   D17 c0  0.000000
   D17 D0  172.019000
   D19 C0  222.533600
   D19 D0  70.172000
D19
   D08 C0  207.101200
   D08 D0  112.046000
   D18 c0  0.000000
   D18 D0  70.159000
D20
   D08 c0  0.000000
   D08 D0  143.663000
```

```
   D21 C0  243.520300
   D21 D0  241.002000
D21
   D20 c0  0.000000
   D20 D0  240.988000
   D22 C0  236.303700
   D22 D0  176.088000
D22
   C00 C0  209.374700
   C00 D0  203.939000
   D21 c0  0.000000
   D21 D0  176.076000
   D23 C0  148.150700
   D23 D0  215.572000
D23
   D22 c0  0.000000
   D22 D0  215.560000
   D24 C0  203.234500
   D24 D0  221.206000
D24
   D23 C0  0.000000
   D23 D0  221.194000
   D25 C0  177.332800
   D25 D0  258.012000
D25
   D24 C0  0.000000
   D24 D0  257.997000
   D26 C0  186.414300
   D26 D0  168.174000
D26
   D25 C0  0.000000
   D27 C0  142.245200
   D27 D0  185.520000
```

```
D27
   D26 C0  0.000000
   D26 D0  185.519000
   D28 C0  224.304200
   D28 D0  204.224000
D28
   D27 C0  0.000000
   D27 D0  204.209000
   D29 C0  268.524600
   D29 D0  206.545000
D29
   D28 C0  0.000000
   D28 D0  206.530000
   D30 C0  241.412700
   D30 D0  427.916000
D30
   D29 C0  0.000000
   D29 D0  427.904000
   D31 C0  211.144200
   D31 D0  235.970000
D31
   D30 C0  0.000000
   D30 D0  235.963000
   D32 C0  164.412900
   D32 D0  229.815000
D32
   C01 C0  126.510800
   C01 D0  155.172000
   D31 C0  0.000000
   D31 D0  229.806000
```