C++面向对象技术课程实验指导用书

课程名称：程序设计（II）实验

单　　位：中山大学软件学院

编　　写：张锋

2010 年 2 月(第六版)

一、目的与要求：

通过上机实验理解C++程序的执行过程，掌握C++语言的语法特征；熟练掌握在Visual C++开发环境下独立编写和调试C++程序；掌握面向对象程序和泛型程序设计思想和技术，建立起用面向对象思维和泛型思维进行问题分析和程序设计的习惯；掌握用C++语言解决实际问题的方法，建立优秀的面向对象编程和泛型编程风格，为进行大型程序设计和科研编程打下基础。

二、学时分配

6个实验共36学时

实验名称	学时分配
实验一、C++程序设计基础与开发环境	4
实验二、类与对象	6
实验三、运算符重载	6
实验四、继承机制、虚基类、虚函数与运行时多态	8
实验五、模板、STL与泛型编程	8
实验六、异常处理、C++ I/O流	4

三、实验目的和实验内容

实验一、C++程序设计基础与开发环境

【实验目的】

1. 通过实验体会 C++在面向过程程序设计方面与 C 的异同
2. 初步了解 C++的程序设计风格
3. 掌握函数重载和命名空间
4. 进一步熟悉 Visual C++集成开发环境

【实验内容】

1. 编写一个完整的面向过程风格的 C++程序，满足以下条件：
   1) 三个重载函数，分别接收两个整数、两个浮点数和两个字符串做参数(引用类型)，实现两个数互相交换。
   2) 在主函数中输入交换前和输出交换后的两个数，实现两数值交换。
   3) 主函数应给出必要的输入提示。
2. 有以下 namespace 定义，

```
namespace mynspace
{
    const int SIZE = 100;
    int A[SIZE];
    int mode(int [], int);
};
```

mode 函数的功能是输出数组中的众数，及其出现的次数，其返回值是众数的个数。完成 mynspace 的定义，并在 main 函数中写出完整程序，演示函数的功能(将 mynspace::A 作为实参传给 mode 函数)。

3. 分别写 C 和 C++程序，将文本文件 old.txt 中的所有内容拷贝到新文件 new.txt 中。
4. 输入下面的程序，用编译器的单步调试功能观察其输出，说明为什么？如果把注释语句中的注释符号“//”去掉，程序能运行吗？试解释原因。

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

```
namespace alip
{
      int ai=16, aj=15, ak=23;
}
int aj=0;
void mainip()
{
      //cout<<"ai:"<<ai<<endl;
      using namespace alip;
      ++ai;
      cout<<"ai:"<<ai<<endl;
      //++aj;
      ++::aj;
      cout<<"::aj:"<<::aj<<endl;
      ++alip::aj;
      cout<<"alip::aj:"<<alip::aj<<endl;
      cout<<"ak:"<<ak<<endl;
      int ak=97;
      cout<<"ak:"<<ak<<endl;
      ++ak;
      cout<<"ak:"<<ak<<endl;
}

namespace blip
{
      int bi=16, bj=15, bk=23;
}
int bj=0;
//int worongInit=bk;
int main()
{
      cout<<"main() output starting"<<endl;
      //cout<<"bi:"<<bi<<endl;
      using blip::bi;
      ++bi;
      cout<<"bi:"<<bi<<endl;
      cout<<"bj:"<<bj<<endl;
      using blip::bj;
      ++bj;
      cout<<"bj:"<<bj<<endl;
      int bk;
      cout<<"bk:"<<bk<<endl;
      //using blip::bk;
```

```
        cout<<"mainip() output starting"<<endl;
        mainip();
        return 0;
    }
```

实验二、类与对象

【实验目的】

1. 学习类的定义和使用方法
2. 学习对象的声明和使用
3. 学习对象构造中，默认构造函数、拷贝构造函数、类型转换构造函数和普通多参数构造函数的定义和使用
4. 类静态成员的定义和使用
5. 体会深拷贝和浅拷贝的区别
6. 使用 Visual C++的 Debug 功能，观察程序流程，观察类的构造函数、析构函数的执行顺序和过程
7. 体会基于对象程序设计方法和面向过程程序设计方法的区别

【实验内容】

1. 完成以下类定义：

```
class seriesComp {
        int n; //n 大于或等于 1
        seriesComp(int n);
        int sum();
        int fib();
        double taylor(double x);
};
```

其中 sum()计算 1+2+3+...+n；

fib()计算斐波纳契(Fibonacci)数列前 n 项和；

taylor()计算 $1-\frac{x}{1!}+\frac{x^2}{2!}-...+(-1)^n\frac{x^n}{n!}$

2. 定义一个类，确保该类实例化的对象数目最多只能有一个(提示，使

用 static 数据成员和成员函数)。

3. Josephus 问题：n 个人围坐在一圈，现从指定的第 s 个人开始报数，数到第 m 个人出列，然后从出列的下一个人重新开始报数，数到第 m 个人又出列，如此重复，直到所有的人全部出列为止。

   定义一个链表类和 Josephus 类。编制应用程序，给出人数 n，起始位置 s 和数人数 m，解决 Josephus 问题。仔细体会你的方法与面向过程的解决方法的区别。

4. 定义一个名为 Date 的类，用于输入并验证日期，类中的数据成员和成员函数应满足下面的规则；在主函数中编写相应代码验证这些规则是否正确地实现了；并用 Debug 功能仔细体会程序的运行过程。

   1) 日期包括年(year)、月(month)、日(day)，均为整型数据，以及一个布尔数据成员 pass 用来判断输入日期是否正确。

   2) 布尔类型 private 成员函数 checkFormat()，有一个字符串引用形参，验证读入的日期格式是否正确（yyyy:mm:dd 格式）？

   3) void 类型 private 成员函数 validate()，没有形参。判断输入的日期是否合法。

   4) 默认构造函数，没有形参，将 year，month，day 分别初始化为 2006，1，1；在构造函数中要调用 validate（）。

   5) 定义拷贝构造函数

   6) 定义类型转换构造函数，能从字符串转换成 Date 对象。

   7) 使用带默认参数的构造函数，三个整形形参 int y，int m=2，int d=29，分别对 year，month，day 赋值；注意要调用 validate（）验证。

   8) void 类型 public 成员函数 setDate()，有一个字符串引用形参，负责设置新日期，首先调用 checkFormat()判断日期数据格式是否正确，如正确，继续调用 validate()验证输入日期

   9) void 类型 public 成员函数 printData()，没有形参，根据 pass 值决定是否打印设置的日期，如果打印，要按照”yyyy 年 mm 月 dd 日”的格式。

实验三、运算符重载

【实验目的】

1. 掌握常用运算符重载的定义方法
2. 掌握运算符重载的两种方式：成员函数和友元函数
3. 体会运算符重载对构建 ADT 的作用

【实验内容】

1. 参考教材第 11 章的例子。实现数学中的复数类 Complex，它能实现普通的+、-、*、/数学运算，=、+=、-=、*=、/=赋值运算，前后序的自增自减（++a，a++，--a，a--）运算，==、!=关系运算。

2. 以下类定义代码段，Vector 是一个简单的向量类，vectorContainer 向量容器类可以包含多个向量，它重载了[],->,()三个运算符。试完成类定义，使得类 vectorContainer 的对象 VC 表现出以下行为：

   VC[i]; //打印 VC 中第 i 个向量的所有元素，i 为整数,下同
   VC->output();//打印 VC 中某个向量所有元素及其元素和，该向量的元素和在所有向量元素和中是最大的
   VC(); //把 VC 中的向量按照其元素和大小排序，然后顺序输出向量及其元素和
   VC(i); //把 VC 中的向量按照其元素和由小到大排序，然后输出第 i 个向量所有元素及其元素和
   VC(i, j);//输出第 i 个向量的第 j 个元素。

   思考以下问题：
   Vector 类和 vectorContainer 类有良好的“封装性”吗？

   如果 vectorContainer 不是 Vector 的友元，代码应如何改动？

.
```
class Vector
{
    float v[4];
    //...
    friend class vectorContainer;
public:
    Vector(float ve[]){
        for (int i=0; i<4; i++)
            v[i]=ve[i];
    }
    Vector(){
        for (int i=0; i<4; i++)
            v[i]=0;
    }
    ~Vector(){}

    void output();
    }
};

class vectorContainer
{
```

```
    Vector* _v;
    int _size;
    //...

public:
    vectorContainer(Vector *ve, int size){
        _v = new Vector[size];
        _size = size;
        for (int i=0; i<size; i++)
            _v[i]=ve[i];
    }

    vectorContainer();
    ~vectorContainer();

    Vector& operator[](int idx);
    Vector* operator->();
    void operator()();
    Vector& operator()(int i);
    float& operator()(int i, int j);
};
```

实验四、继承机制、虚基类、虚函数与运行时多态

【实验目的】

1. 学习类的继承关系，声明派生类
2. 熟悉不同继承方式下派生类对基类成员的访问限制
3. 学习多继承的原理，虚基类的用途及其使用方法
4. 虚函数的功能，虚函数支持面向对象多态机制的工作原理
5. 学习纯虚函数与抽象基类的定义以及其作用，理解抽象基类和具体类的区别。
6. 基类与派生类构造函数和释构函数的调用顺序。
7. 理解基本的设计模式

【实验内容】

1. 编写一个基类，声明有 public、protected、private 成员，然后使用 public 继承，protected 继承，private 继承分别生成派生类，观察派生类对基类不同类型成员的访问，验证以下继承规则：

   1) A *public* member is accessible from anywhere within a program.

   2) A *private* member can be accessed only by the member functions and friends of its class.

   3) A *protected* member behaves as a public member to a derived class, and behaves as a private member to the rest of the program.

2. 有类定义

```
class base
{
public:
   virtual void iam() { cout<<"base\n"; }
};
```

从类 base 派生两个类，每个派生类定义函数 iam()输出派生类的名字。

创建这些类的对象，并分别通过这些对象调用 iam()。

把指向两个继承类对象的指针分别赋值给 base*指针，通过这些指针调用 iam()。

3. 防真农场，一个哺乳动物类如下所示：

```
#include <iostream.h>
class Mammal
{
public:
    Mammal():itsAge(2),itsWeight(5)
    {
        cout <<”Mammal constructor…\n”;
    }
    ~Mammal(){cout <<”Mammal destructor…\n”;}
    int GetAge()const {return itsAge;}
    void SetAge(int age) {itsAge=age;}
    int GetWeight() const {return itsWeight;}
```

```
    void SetWeight(int weight) {itsWeight=weight;}
    void Move() const {cout <<"Mammal move one step \n";}
    void Speak() const {cout <<"Mammal speak!\n";}
    void sleep() const {cout <<"shhh,I'm sleeping.\n";}
protected:
    int itsAge;
    int itsWeight;
};
```

1) 狗属哺乳动物，且它的属性有品种之分（在哺乳类基础上增加品种数据成员），叫声区别于其它动物（Speak()，输出”Woof!”），还会摇尾巴(增加成员函数，输出”Tail wagging…”),乞讨食物(增加成员函数，输出”bagging for food…”)。

2) 驴、马、猪也属哺乳动物，其叫声分别为:“Meow!”,”Winnie!”,”Oink!”。

3) 骡既是驴，也是马，叫声为”mule”。

4) 编程分别使各个动物表现为不一样的行为，并观察基类与派生类的构造函数与释构函数的调用顺序。

5) 把 Mammal 中的非构造、非释构成员函数改为虚函数，然后派生不同的动物类，编写程序说明虚函数的作用（与普通成员函数相比较）。

6) 如果 Mamal 中的非构造、非释构成员函数中有纯虚函数，派生不同的动物类（具体类），编写程序声明对象体会抽象类和具体类的区别。

4. 建立一个中山大学学生基类 sysuStudent，有注册交费 reg()、选课 select()等成员函数；派生出软件学院学生类 ssStudent，管理学院学生类 msStudent；从 ssStudent 和 msStudent 派生出双学位学生 ddStudent。它们都定义了 reg()和 select()成员函数，编写程序体验**虚基类**、**虚函数**的作用。

5*. 定义类，满足以下基本需求：

1) 该类主要功能是对数组进行排序；

2）排序的算法有简单选择排序、冒泡排序等，可选；

3）类设计尽量满足“良好的可扩展性”要求，也就是类功能有良好的可扩充性，比如排序算法还可以添加插入排序、快速排序等。

6*. 某市某产鞋乡镇企业，需要开发一套管理软件，对企业的产鞋过程进行管理。在该企业中，生产一种鞋需要一条生产线(比如皮鞋需要

一种生产线，运动鞋需要另一种)。该企业随着业务的不断发展，会不断引进新生产线，生产新品种的鞋，所以企业希望开发的软件能够不断适应这个需求。你能用面向对象的程序设计思想，模拟该软件的核心功能实现吗？

实验五、模板、STL 与泛型编程

【实验目的】

1. 掌握 C++中的模板机制对泛型编程思想的支持
2. 掌握泛型函数、泛型类的定义及其使用方法
3. 熟悉 STL Sequence 和 STL Associative 容器类的使用方法
4. 练习 STL 通用算法、函数对象和函数适配器的使用

【实验内容】

1. 用模板机制实现泛型函数，重写实验一的第一题，功能要求不变。
2. 泛型化实验三的 complex 类，使其能处理不同的数据类型的数据。
3. 下面的源代码是模拟 STL list 的实现，试仔细体会链表的性质和 Iterator 的实现方法（请补充完成未定义的成员函数）。并编写程序比较这个实现和标准 STL list 实现的异同。

```
/*************************************************************
************/

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

class list {
public:
    struct listelem;    //forward declarations
    class iterator;
    friend iterator;
    list():h(0), t(0) {}//construct the empty list
    list(size_t n_elements,    char c);
    list(const list& x);
    list(iterator b, iterator e);
    ~list() { release(); }
    iterator begin()const { return h; }
    iterator end()const {return t;}
    void push_front(char c);
```

```
    void pop_front();
    char& front(){return h -> data;}
    char& back() {return t -> data;}
     bool empty()const{ return h == 0;}
    void release();
    friend ostream& operator<<(ostream& out, list& x);

    struct listelem          //list cell
    {
        char    data;
        listelem* next, *prev;
        listelem(char c, listelem* n, listelem* p)
                :data(c), next(n), prev(p){}
     };
    //scoped within class list
    class iterator{
    public:
        iterator(listelem* p = 0):ptr(p){}
        iterator operator++();
        iterator operator--();
        iterator operator++(int);
        iterator operator--(int);
        listelem* operator->(){return ptr;};
        char& operator*(){return ptr -> data;}
        operator listelem*(){return ptr;} //conversion
    private:
        listelem* ptr;//current listelem or 0
    };

    private:
    listelem* h, *t;        //head    and tail
};

list::list(size_t n_elements,    char c)
{
    assert(n_elements > 0);
    h = t = 0;
    for(size_t i = 0; i < n_elements; ++i)
        push_front(c);
}

list::list(const list& x)
{
    list::iterator r = x.begin();
```

```
    h = t = 0; //needed for empty list
    while (r != 0)
        push_front(*r++);
}

void list::release()
{
   while (h != 0)
       pop_front();
}

ostream& operator<<(ostream& out, list& x)
{
    list::iterator p = x.begin(); //gets x.h

    out << "list = (";
    while (p != 0) {
        out << *p    << ","; //gets a char&
        ++p; //advances iterator using next
    }
    cout << ")\n";
    return out;
}

void list::push_front(char c)
{
    listelem* temp = new listelem(c, h, 0);

    if (h != 0) {       //was a nonempty list
        h -> prev = temp;
    temp -> next = h;
        h = temp;
    }
    else                    //was an empty list
        h = t = temp;
}

void list::pop_front()
{
    listelem* temp = h;

    if (h != 0) {       //was a nonempty list
        h = h -> next;
    delete temp;
```

```
        }
        else                        //was an empty list
            cout << "empty list!" <<endl;
    }

    list::iterator list::iterator::operator++()
    {
        assert(ptr != 0);
        ptr = ptr -> next;
        return *this;
    }

    list::iterator   list::iterator::operator++(int)
    {
        assert(ptr != 0);
        iterator temp = *this;
        ptr = ptr -> next;
        return temp;
    }

    int main()
    {
         list l1;
         l1.push_front('A');
         l1.push_front('F');
         l1.push_front('C');
         l1.push_front('z');
         l1.push_front('l');

         cout <<    l1;
         return 0;
    }
```

4. 理解教材给出的 Sequence 容器成员类型，Sequence 容器成员函数用法。分别编写程序，演示 Sequence 容器 vector、list、deque 的构造、插入、删除、访问、赋值、交换操作。

5. 理解教材给出的 Associative 容器成员类型，Associative 容器成员函数功能。分别编写程序，演示 Associative 容器 set、map、multiset、multimap 的构造、插入、删除、访问、赋值、交换操作。

6. Using a random number generator, generate 10, 000 integers between 0 and 9,999. Place them in a list<int> container. Compute and print the median value. What was expected? Compute the frequencies of each value; in other words, how many 0s were generated, how many 1s were generated, and so forth. Print the value with greatest frequency. Use a

vector<int> to store the frequencies. Be noticed that you should use algorithm, function object and function adapter.

实验六、异常处理、C++ I/O 流

【实验目的】

1. 理解 C++语言的异常抛出处理机制
2. 学习异常处理的声明和执行过程
3. 学习 C++中的格式化 I/O
4. 掌握 C++文件 I/O 流

【实验内容】

1. 编写计算整数阶乘的程序，用异常抛出处理机制进行输入、溢出时异常的处理。在主程序中设置不同条件，观察程序的处理流程。

2. Suppose you are given line-oriented data in a file formatted as follows:

```
Australia
5E56,7667230284,Langler,Tyson,31.2147,0.00042117361
2B97,7586701,Oneill,Zeke,553.429,0.0074673053156065
4D75,7907252710,Nickerson,Kelly,761.612,0.010276276
9F2,6882945012,Hartenbach,Neil,47.9637,0.0006471644
Austria
480F,7187262472,Oneill,Dee,264.012,0.00356226040013
1B65,4754732628,Haney,Kim,7.33843,0.000099015948475
DA1,1954960784,Pascente,Lester,56.5452,0.0007629529
3F18,1839715659,Elsea,Chelsy,801.901,0.010819887645
Belgium
BDF,5993489554,Oneill,Meredith,283.404,0.0038239127
5AC6,6612945602,Parisienne,Biff,557.74,0.0075254727
6AD,6477082,Pennington,Lizanne,31.0807,0.0004193544
4D0E,7861652688,Sisca,Francis,704.751,0.00950906238
Bahamas
37D8,6837424208,Parisienne,Samson,396.104,0.0053445
5E98,6384069,Willis,Pam,90.4257,0.00122009564059246
1462,1288616408,Stover,Hazal,583.939,0.007878970561
5FF3,8028775718,Stromstedt,Bunk,39.8712,0.000537974
1095,3737212,Stover,Denny,3.05387,0.000041205248883
7428,2019381883,Parisienne,Shane,363.272,0.00490155
```

The heading of each section is a region, and every line under that heading is a seller in that region. Each comma-separated field represents the data about each seller. The first field in a line is the SELLER_ID which unfortunately was written out in hexadecimal format. The second is the PHONE_NUMBER (notice that some are missing area codes). LAST_NAME and FIRST_NAME then follow. TOTAL_SALES is the second to the last column. The last column is the decimal amount of the total sales that the seller represents for the company. You are to format the data on the terminal window so that an executive can easily interpret the trends. Sample output is given below.

Australia

LastName	*FirstName*	*ID*	*Phone*	*Sales*	*Percent*
Langler	Tyson	24150	766-723-0284	31.24	4.21E-02
Oneill	Zeke	11159	XXX-758-6701	553.43	7.47E-01

3．有磁盘文件 student.txt，文件记录格式为：

学号 姓名 性别 出生年月 生源地 高考成绩

修改实验五(3)的链表结构，使其结点能存储 student.txt 的记录。并在相应成员函数里加入异常抛出处理代码。用 student.txt 的所有数据初始化列表。列表能实现按照高考程序排序，排序后的所有学生纪录输出到 student_sort.txt 中。

附录 VC 程序调试(本部分录自网上资料)

在开发程序的过程中，经常需要查找程序中的错误，这就需要利用调试工具来帮助你进行程序的调试，当然目前有许多调试工具，而集成在 VC 中的调试工具以其强大的功能，一定使你爱不释手。下面我们先来介绍 VC 中的调试工具的使用。

1 VC 调试工具

1．1 调试环境的建立

在 VC 中每当建立一个工程(Project)时，VC 都会自动建立两个版本：Release 版本，和 Debug 版本，正如其字面意思所说的，Release 版本是当程序完成后，准备发行时用来编译的版本，而 Debug 版本是用在开发过程中进行调试时所用的版本。

DEBUG 版本当中，包含着 MICROSOFT 格式的调试信息，不进行任何代码优化，而在 RELEASE 版本对可执行程序的二进制代码进行了优化，但是其中不包含任何的调试信息。

在新建立的工程中，你所看到是 DEBUG 版本，若要选择 RELEASE 版本，可以选择菜单 PROJECT 中的 SETTING 命令，这时屏幕上面弹出 PROJECT SETTEING 对话框，在 SETTING FOR 下拉列表中选择 RELEASE，按 OK 退出，如图 4.1。

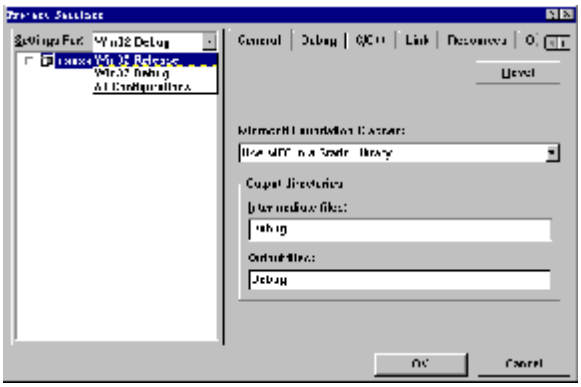


图 4.1

在调试程序的时候必须使用 DEBUG 版本，我们可以在 Project Setting 对话框的 C/C++页中设置调试选项。

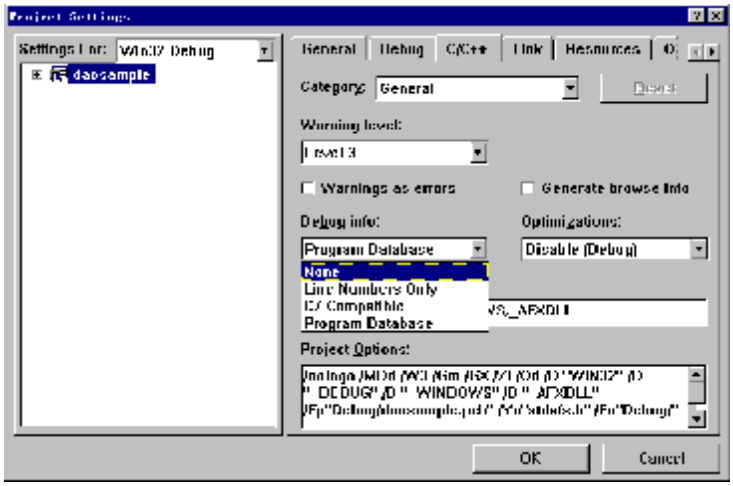


图 4.2

各个选项的含意如下：

- Program Database 表示产生一个存储程序信息的数据文件(.PDB)，它包含了类型信息和符号化的调试信息；
- Line Numbers Only 表示程序经过编译和链接产生的.OBJ 或.EXE 文件仅仅包含全局和外部符号以及行号信息；
- C7 Compatible 表示产生一个.OBJ 或.EXE 文件行号信息以及符号化的调试信息；
- None 表示不产生任何调试信息。

1．2 调试的一般过程

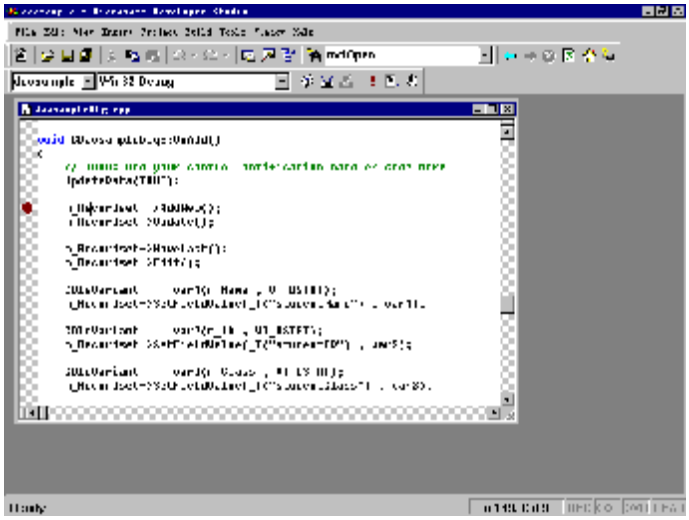
调试，说到底就是在程序的运行过程的某一阶段观测程序的状态，而在一般情况下程序是连续运行的，所以我们必须使程序在某一地点停下来。所以我们所做的第一项工作就是设立断点。其次,再运行程序,当程序在设立断点处停下来时,再利用各种工具观察程序的状态。程序在断点停下来后，有时我们需要按我们的要求控制程序的运行，以进一步观测程序的流向，所以下面我们依次来介绍断点的设置，如何控制程序的运行以及各种观察工具的利用。

1．3 如何设置断点

在 VC 中，你可以设置多种类型的断点，我们可以根据断点起作用的方式把这些断点分为三类:1、与位置有关的断点；2、与逻辑条件有关的断点 3、与 WINDOWS 消息有关的断点下面我们分别介绍这三类断点。

首先我们介绍与位置有关的断点。

1、最简单的是设置一般位置断点，你只要把光标移到你要设断点的位置，当然这一行必须包含一条有效语句的；然后按工具条上的 add/remove breakpoint 按钮或按快捷键 F9；这时你将会在屏幕上看到在这一行的左边出现一个红色的圆点表示这二设 立了一



个断点。

图 4.3

2、有的时候你可能并不需要程序每次运行到这儿都停下来，而是在满足一定条件的情况下才停下来，这时你就需要设置一种与位置有关的逻辑断点。要设置这种断点我们只需要从 EDIT 菜单中选中 breakpoint 命令，这时 Breakpoint 对话框将会出现在屏幕上。选中 Breakpoint 对话框中的 LOCATION 标签，使 LOCATION 页面弹出，如图 4.4

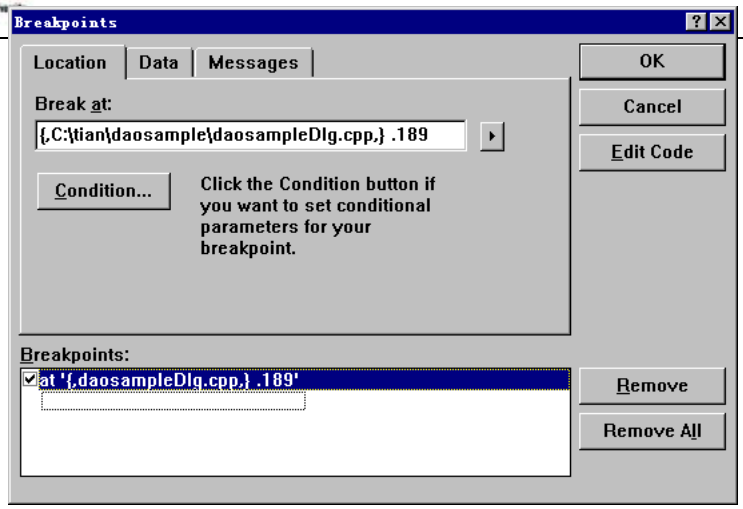


图 4.4

单击 condition 按钮，弹出 Breakpoint 对话框，在 Expression 编辑框中写出你的逻辑表达式，如 X>=3 或 a+b>25,最后按 OK 返回。

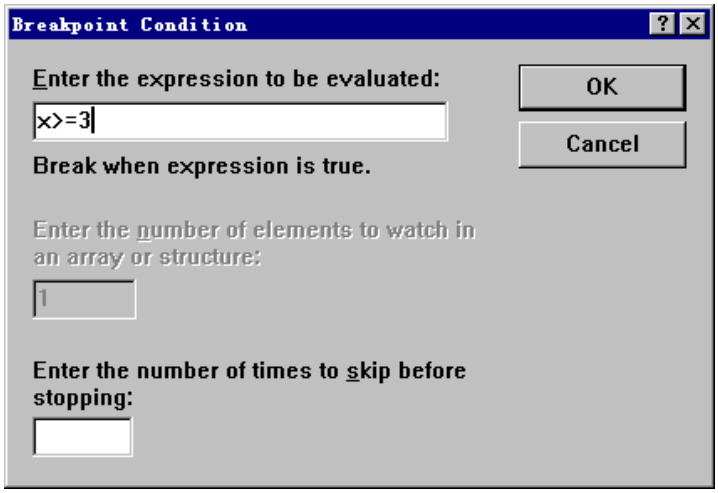


图 4.5

这种断点主要是由其位置发生作用的，但也结合了逻辑条件，使之更灵活。

3、有时我们需要更深入地调试程序，我们需要进入程序的汇编代码，因此我们需要在在汇编代码上设立断点：要设立这种断点我们只需从 View 菜单中选 Debug window 命令 ，

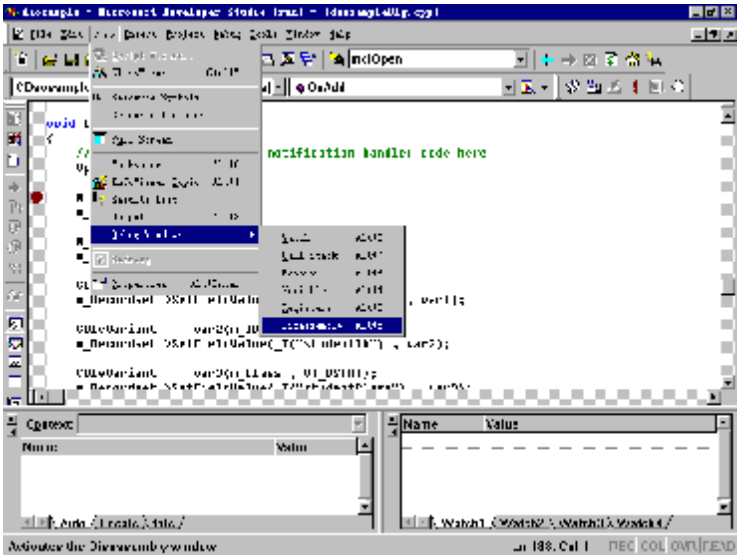
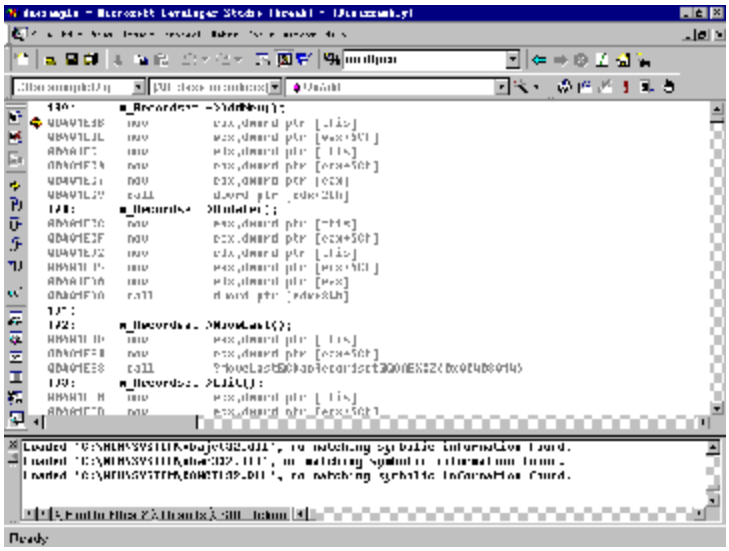


图 4.6



再选 Disassembly 子命令，这时汇编窗口将会出现在屏幕上。

图 4.7

在图 4.7 中的汇编窗口中你将看到对应于源程序的汇编代码，其中源程序是用黑体字显示，下面是且对应的汇编代码。要设立断点，我们只需将光标移到你想设断点处然后点击工具条上的 Insert/Remove Breakpoints 按钮，此后你将会看到一个红圆点出现在该汇编代码的右边。

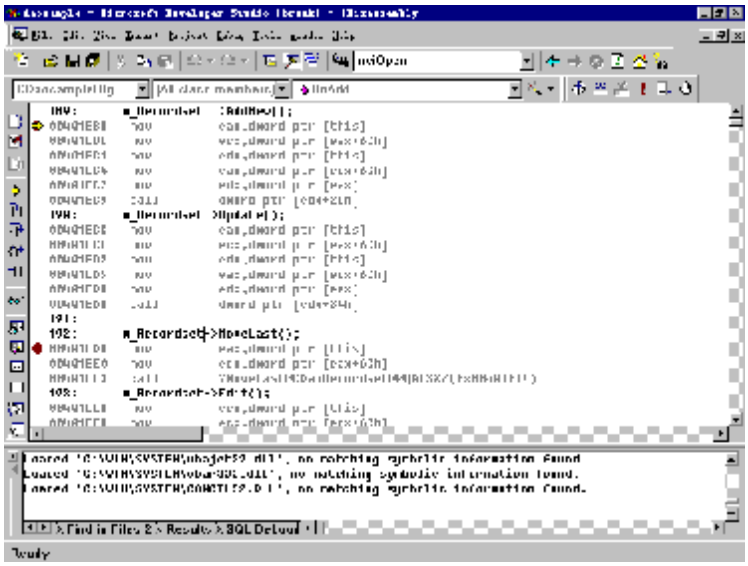
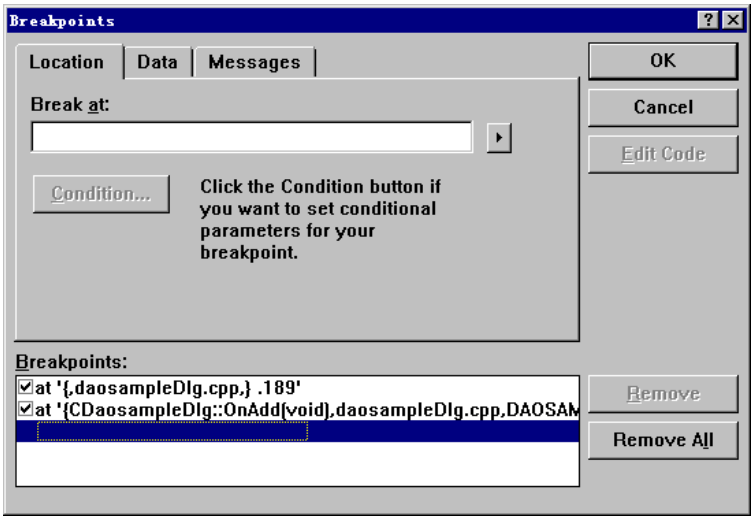


图 4.8

上面所讲的断点主要是由于其位置发挥作用的，即当程序运行到设立断点的地方时程序将会停下来。但有时我们设立只与逻辑条件有关的断点，而与位置无关。所以下面介绍一下与逻辑条件有关的断点。

（1）逻辑条件触发断点的设置：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现



Breakpoint 对话框。

图 4.9

- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出

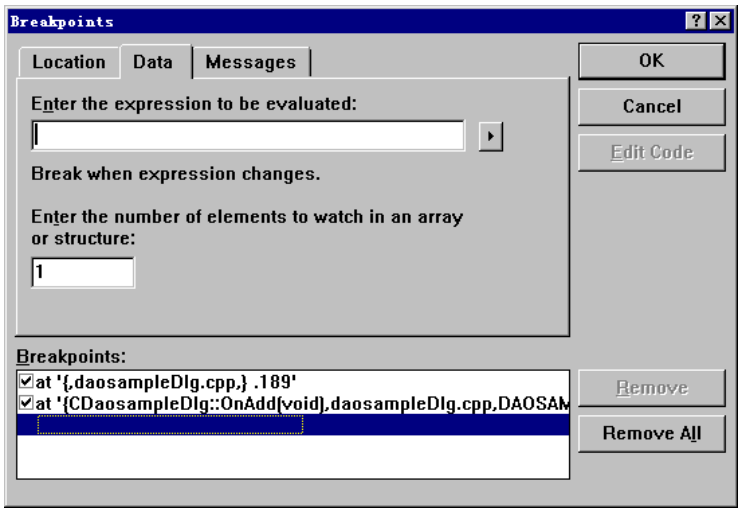


图 4.10

- 在图 4.10 的 DATA 页面中的 Expression 编辑框中写出你的逻辑表达式，如(X==3);

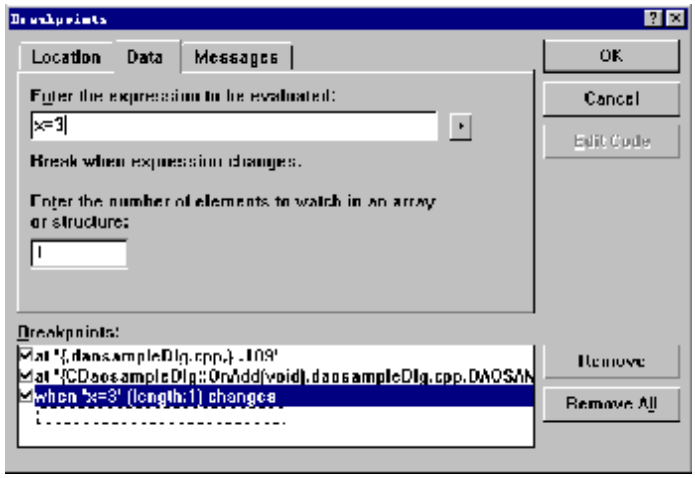


图 4.11

- 最后按 OK 返回。

其他几种断点的设置的方法都与之类似。我们一一加以说明。

（2）监视表达式发生变化断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现 Breakpoint 对话框。
- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出
- 在 Expression 编辑框中写出你需要监视的表达式
- 最后按 OK 键返回。

（3）监视数组发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会 出现 Breakpoint 对话框。
- 选中 Breakpoint 对话框中的 DATA 标签，对应的页面将会弹出
- 在 Expression 编辑框中写出你需要监视数组名；

- 在 Number of Elements 编辑框输入你需要监视数组元素的个数；
- 按 OK 键返回。

（4）监视由指针指向的数组发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令，这时在屏幕上将会出现 Breakpoint 对话框。
- 选中 Breakpoint 对话框中的 DATA 标签；
- 在 Expression 编辑框中输入形如*pointname,其中 *pointname 为指针变量名；
- 在 Number of Elements 编辑框输入你需要监视数组元素的个数；
- 按 OK 键返回。

（5）监视外部变量发生变化的断点：

- 从 EDIT 菜单中选中 breakpoint 命令这时屏幕上将会出现 Breakpoint 对话框；
- 选中 Breakpoint 对话框中的 DATA 标签；
- 在 Expression 编辑框中输入变量名；
- 点击在 Expression 编辑框的右边的下拉键头；
- 选取 Advanced 选项,这时 Advanced Breakpoint 对话框出现；
- 在 context 框中输入对应的函数名和(如果需要的话)文件名；
- 按 OK 键关闭 Advanced Breakpoint 对话框。
- 按 OK 键关闭 Breakpoints 对话框。

（6）在讲了位置断点和逻辑断点之后我们再讲一下与 WINDOWS 消息有关的断点。

注意：此类断点只能工作在 x86 或 Pentium 系统上。

- 从 EDIT 菜单中选中 breakpoint 命令，这时屏幕上将会出现 Breakpoint 对话框；
- 选中 Breakpoint 对话框中的 MESSAGE 标签，对应的页面将会弹出；
- 在 Break At WndProc 编辑框中输入 Windows 函数的名称；
- 在 Set One Breakpoint From Each Message To Watch 下拉列表框中选择对应的消息；
- 按 OK 返回。

1．4 控制程序的运行

上面我们讲了如何设置各类断点，下面我们来介绍如何控制程序的运行。当我们从菜单 Build 到子菜单 Start Debuging 选择 Go 程序开始运行在 Debug 状态下，程序会由于断点而停顿下来后，可以看到有一个小箭头，它指向即将执行的代码。

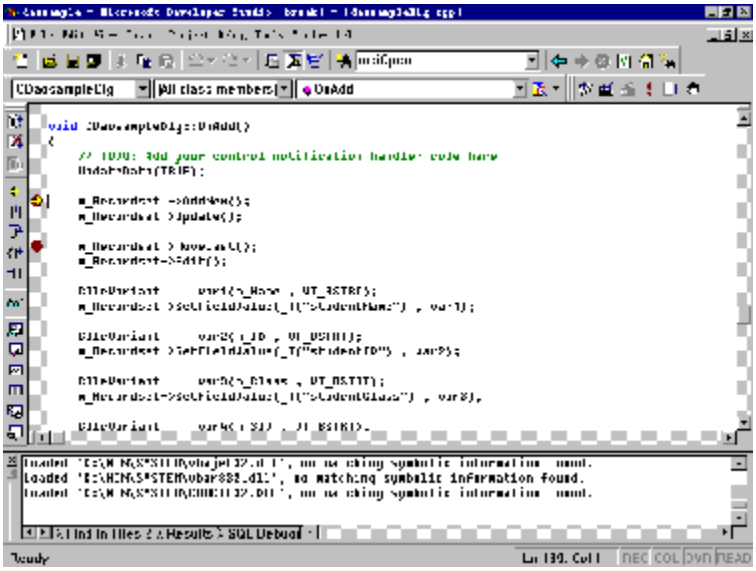


图 4.12

随后，我们就可以按要求来控制程序的运行：其中有四条命令：Step over，step Into ，Step Out ,Run to Cursor。

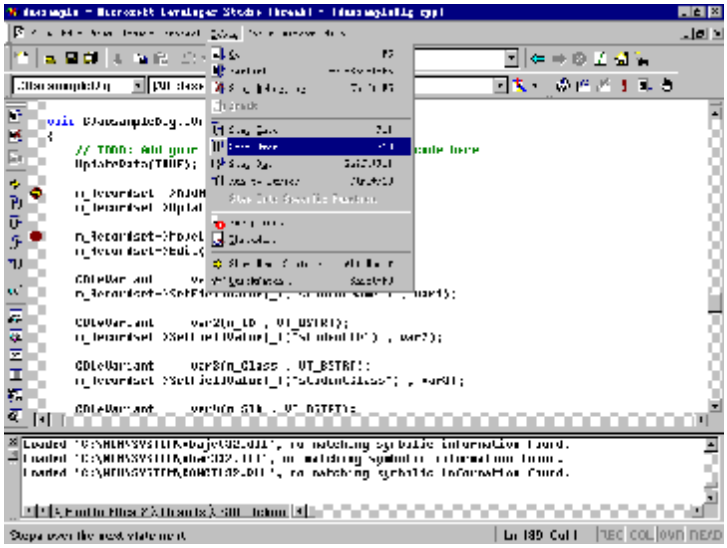


图 4.13

在图 4.13 中：

Step over 的功能是运行当前箭头指向的代码(只运行一条代码)。

Step Into的功能是如果当前箭头所指的代码是一个函数的调用，则用 Step Into 进入该函数进行单步执行。

Step Out 的功能是如当前箭头所指向的代码是在某一函数内，用它使程序运行至函数返回处。

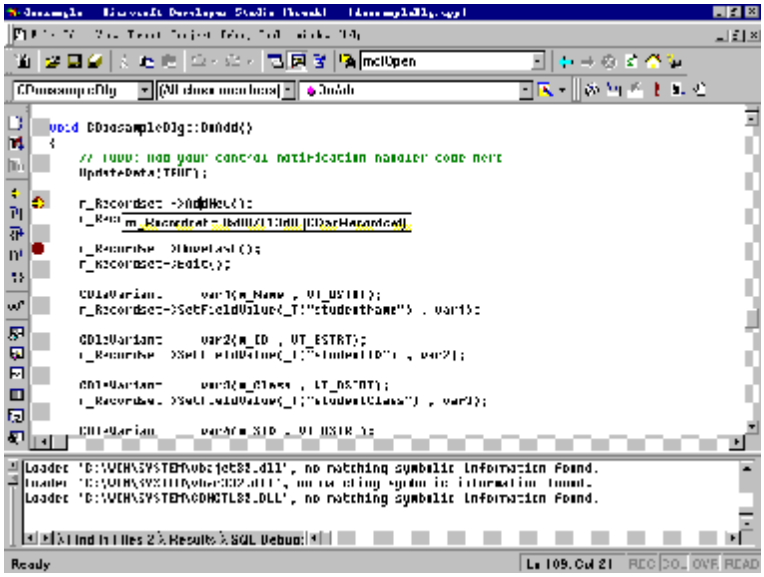
Run to Cursor的功能是使程序运行至光标所指的代码处。

1．5 查看工具的使用

调试过程中最重要的是要观察程序在运行过程中的状态，这样我们才能找出程序的错误之处。这里所说的状态包括各变量的值，寄存中的值，内存中的值，堆栈中的值 ，为此我们需要利用各种工具来帮助我们察看程序的状态。

- 弹出式调试信息泡泡(Data Tips Pop_up Information)。

当程序在断点停下来后，要观察一个变量或表达式的值的最容易的方法是利用调试信息泡泡。要看一个变量的值，只需在源程序窗口中，



将鼠标放到该变量上，你将会看到一个信息泡泡弹出，其中显示出该变量的值。

图 4.14

要查看一个表达式的值，先选中该表达式，仍后将鼠标放到选中的表达式上，同样会看到一个信息泡泡弹出以显示该表达式的值如图 4.15 所示。

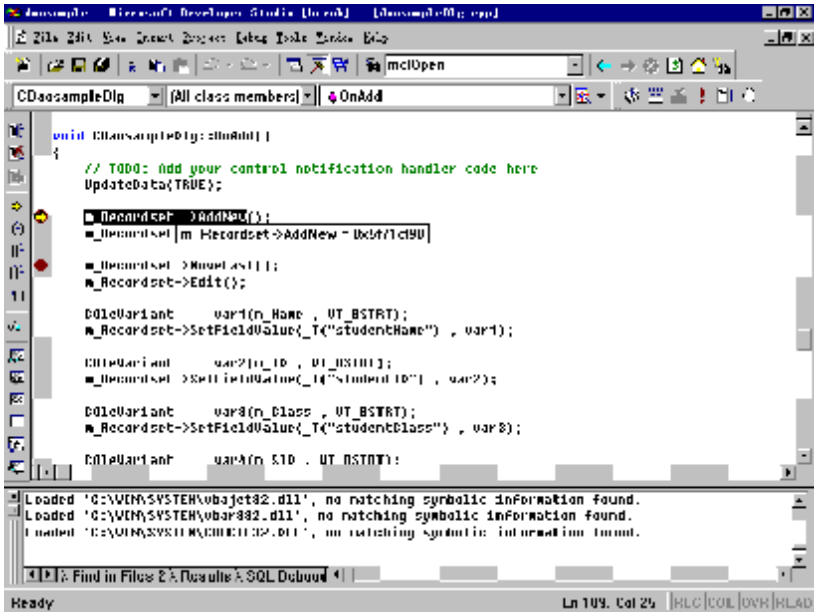
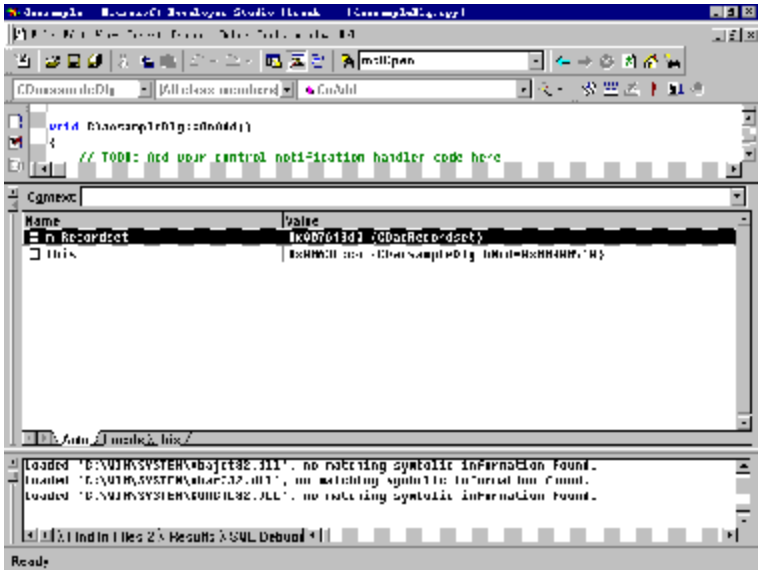


图 4.15

♦ 变量窗口(VARIABLE WINDOW)。

在 VIEW 菜单，Debug window 选 Variables window; 变量窗口将出现在屏幕上。其中显示着变量名及其对应的值。你将会看到在变量观察



窗口的下部有三个标签：AUTO ,LOCAL,THIS 选中不同的标签，不同类型的变量将会显示在该窗口中。

图 4.16

♦ 观察窗口(WATCH WINDOW)：

在 VIEW 菜单，选择 Debug window 命令，Watch window 子命令。这时变量窗口将出现在屏幕上。

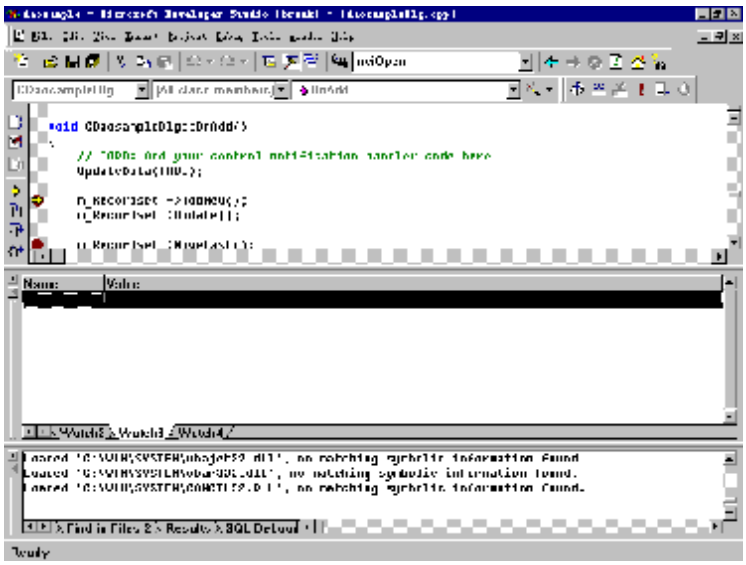
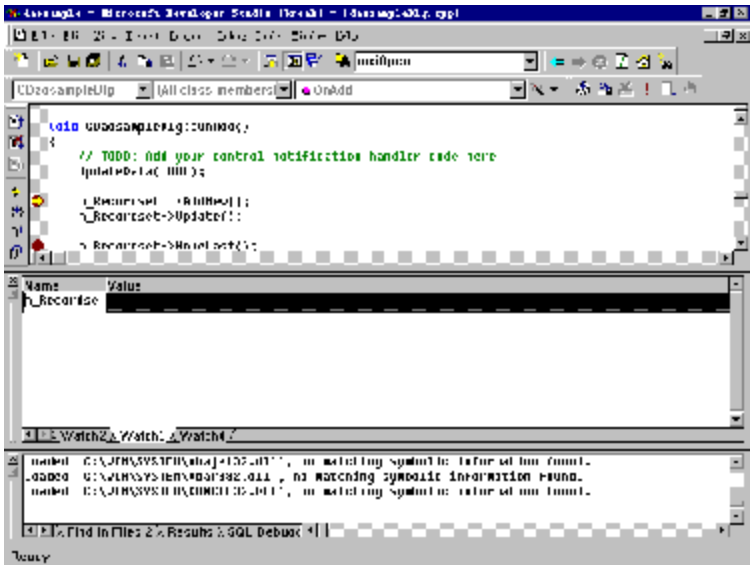


图 4.17



在图 4.17 的观察窗口中双击 Name 栏的某一空行，输入你要查看的变量名或表达式。

图 4.18

回车后你将会看到对应的值。观察窗口可有多页，分别对应于标签 Watch1,Watch2,Watch3 等等。假如你输入的表达式是一个结构或是一个对象，你可以用鼠标点取表达式右边的形如 + ，以进一步观察其中的成员变量的值如图 4.19。

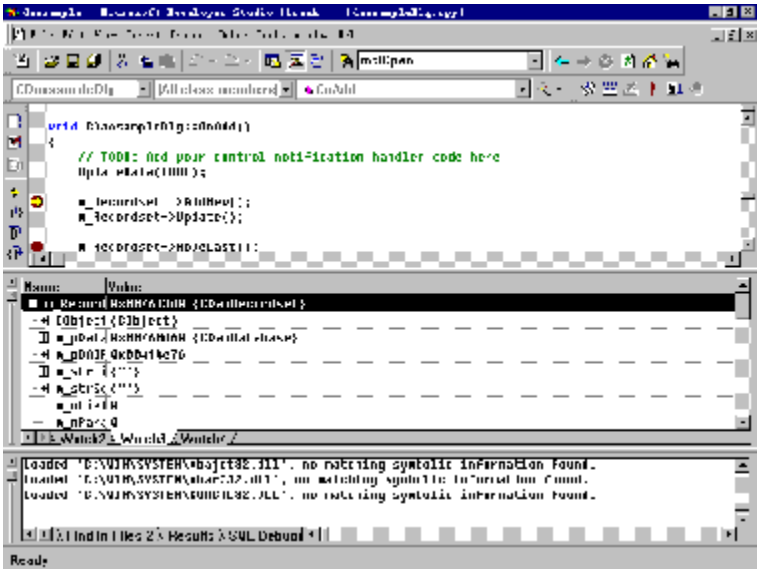


图 4.19

♦ 快速查看变量对话框(quick watch)：

在快速查看变量对话框中你可以象利用观察窗口一样来查看变量或表达式的值。但我们还可以利用它来该变运行过程中的变量，具体操作如下：

（1） 在 Debug 菜单，选择 Quick Watch 命令，这时屏幕上将会出现 Quick Watch 对话框；

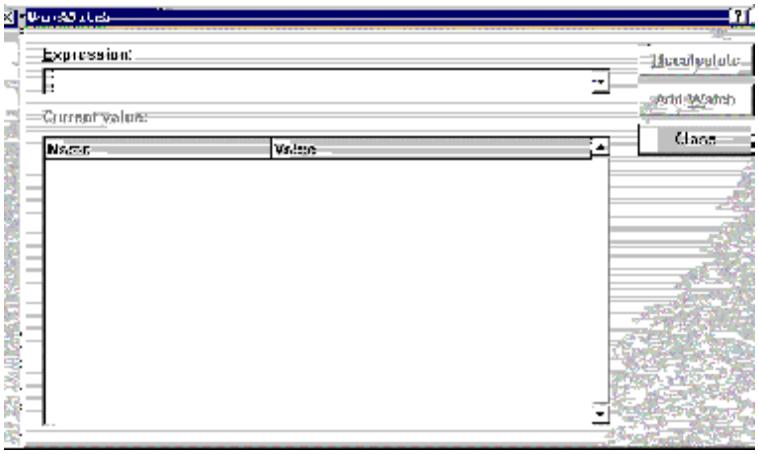


图 4.20

（2） 在 Expression 编辑框中输入变量名，按回车；

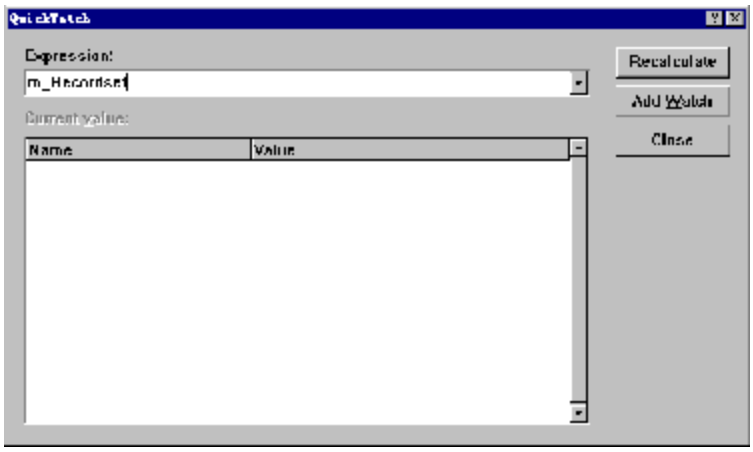


图 4.21

（3)在 Current Value 格子中将出现变量名及其当前对应的值如图 4.22:

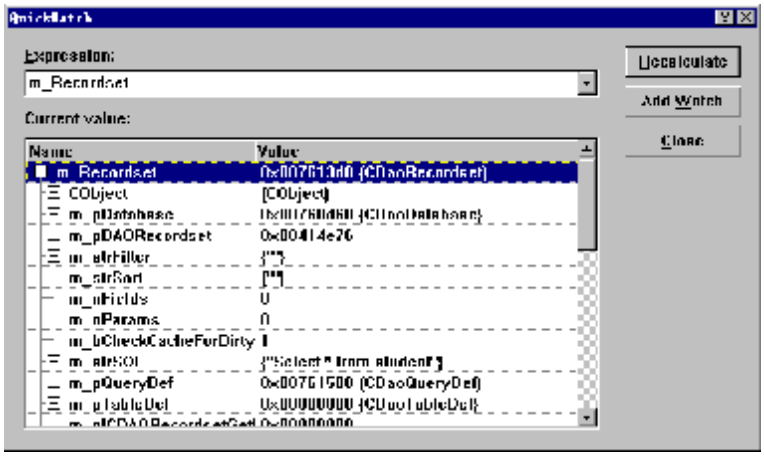


图 4.22

（4）如要改变该变量的值只需双击该变量对应的 Name 栏，输入你要改变的值；

（5）如要把该变量加入到观察窗口中，点击 Add watch 按钮；

（6）点击 Close 按钮返回；

♦ 我们还可以直接查看内存中的值

（1）从 View 菜单中选取 Debug windows 及 Memory 子命令。Memory Window 出现；

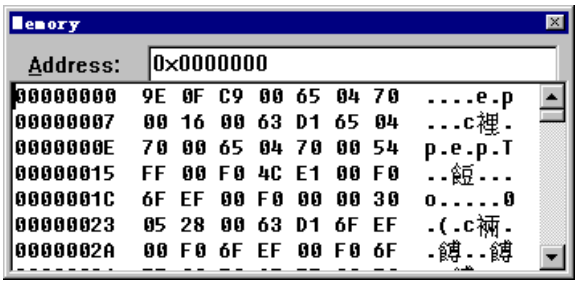


图 4.23

（2）在 Address 编辑框中输入你要查看的内存地址，回车。对应内存地址中的值将显示在 Memory window 的窗口中。

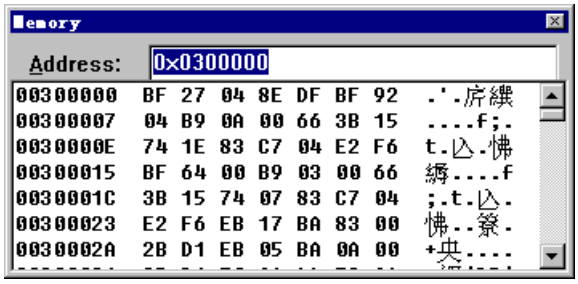


图 4.24

♦ 在调试过程中，有时我们需要查看或改寄存器中的值。我们只需：

（1）从 View 菜单中选取 Debug window 及 Registers 子选项。Registers 窗口出现。在 Registers 窗口中，信息以 Register=Value 的形式显示，其中 Register 代表寄存器的名字，Value 代表寄存器中的值。

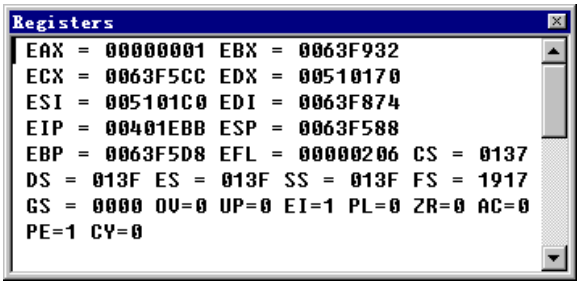


图 4.25

（2）如果你要修改某一个寄存器的值，用 TAB 键，或鼠标将光标移到你想改变的值的右边，然后输入你想要的值。回车返回。

在寄存器中，有一类特殊的寄存器称为标志寄存器，其中有八个标志位：

OV 是溢出标志；

UP 是方向标志；

EI 是中断使能标志；

Sign 是符号标志，

Zero 是零标志。

Parity 是奇偶较验标志。

Carry 是进位标志。

2 高级调试技术

前面我们讲了调试工具的使用，利用它可以就进行常规的调试，即使程序在某处停下来，再观察程序的当前壮态。而且这些工具在且它调试器中也有。但我们知道我们知道在 VC 程序的开发过程中，光有这些工具是不够的。为了更快更好地开发程序，我们还需要利用更高级的调试工具。我们知道，在利用 VC 开发过程中，利用 MFC 将会极大地方便应用程序的开发，所以开发人员往往是利用 MFC 来开发应用程序，正是这个原因 Microsoft 公司在 MFC 中提供了一些特性来帮助你进行程序的调试。

我们知道在 MFC 中，绝大多数类都是从一个叫做 Cobject 的类继承过来的，虽然这是一个虚基类，但它定义了许多成员函数，其中许多成员函数是用来支持程序的调试的，如 Dump ,Assertvalid 等成员函数。另外他们都支持如 TRACE,ASSERT 等宏，并支持内存漏洞的检查等等。我们知道，为了支持调试，类库肯定在在性能上有所损失，为此 Microsoft 公司提供了两个不同的版本的类库：Win32 Debug 版本和 Win32 Release 版本。在前面我们已经提到，每当我们建立一个工程时，我们也有对应的两个版本。在你的 DEBUG 版本的工程中，编译器连接 DEBUG 版本的 MFC 类库；在你的 RELEASE 版本的工程中编译器连接 RELEASE 版本的 MFC 类库以获得尽可能快的速度。下面我们来介绍这些工具的利用。

2.1 TRACE 宏的利用

TRACE 宏有点象我们以前在 C 语言中用的 Printf 函数，使程序在运行过程中输出一些调试信息，使我们能了解程序的一些状态。但有一点不同的是：TRACE 宏只有在调试状态下才有所输出，而以前用的 Printf 函数在任何情况下都有输出。和 Printf 函数一样，TRACE 函数可以接受多个参数如：

```
int x = 1;
int y = 16;
float z = 32.0;
TRACE( "This is a TRACE statement\n" );
TRACE( "The value of x is %d\n", x );
TRACE( "x = %d and y = %d\n", x, y );
TRACE( "x = %d and y = %x and z = %f\n", x, y, z );
```

要注意的是 TRACE 宏只对 Debug 版本的工程产生作用，在 Release 版本的工程中，TRACE 宏将被忽略。

2.2 ASSERT 宏的利用

在开发过程中我们可以假设只要程序运行正确，某一条件肯定成立。如不成立 ，那么我们可以断言程序肯定出错。在这种情况下我们可以利用 ASSERT 来设定断言。ASSERT 宏的参数是一个逻辑表达式，在程序运行过程中，若该逻辑表达式为真，则不会发生任何动作，若此表达式为假，系统将弹出一个对话框警告你，并停止程序的执行。同时要求你作出选择：Abort，Ignore，Retry。若你选择 Abort，系统将停止程序的执行;若你选择 Ignore 系统将忽略该错误，并继续执行程序；若你选择 Retry ，系统将重新计算该表达式，并激活调试器。同 TRACE 宏一样，ASSERT 宏只 DEBUG 版本中起作用，在 RELEASE 版本中 ASSERT 宏将被忽略。

2.3 ASSERT_VALID 宏的利用以及类的 AssertValid()成员函的重载

ASSERT_VALID 宏用来在运行时检查一个对象的内部合法性,比如说现在 有一个学生对象，我们知道每个学生的年龄一定大于零，若年龄小于零，则该学生对象肯定有问题。事实上，ASSERT_VALID 宏就是转化为对象的成员函数 AssertValid()的调用，只是这种方法更安全。它的参数是一个对象指针，通过这个指针来调用它的 AssertValid()成员函数。

与此相配套，每当我们创建从 Cobject 类继承而来的一个新的类时，我们可以重载该成员函数，以执行特定的合法性检查。

2.4 对象的 DUMP 函数的利用

Dump 函数用来按指定的格式输出一个对象的成员变量，来帮助你诊断一个对象的内部情况。与 AssertValid 成员函数一样，Dump 也是 Cobject 类的成员函数。Dump 函数的参数是一个 CdumpContext 对象，你可以象利用流一样往向这个对象中输入数据。当你创建一个 Cobject 继承而来的 新类时，你可以按如下步骤重载你自己的 Dump 函数：

(1) 调用基类的 Dump 函数，以输出基类的内容;

(2) 向 Cdumpcontest 对象输出该类的数据.

例如,典型的 Dump 函数定义如下:

```
#ifdef _DEBUG
void CPerson::Dump( CDumpContext& dc ) const
{
    // call base class function first
    CObject::Dump( dc );
```

```
    // now do the stuff for our specific class
    dc << "last name: " << m_lastName << "\n"
       << "first name: " << m_firstName << "\n";
}
#endif
```

你可能已经注意到整个函数的定义都包含在#ifdef _DEBUG 和#endif 中，这使得 Dump 成员函数只在 DEBUG 版本中发生作用，而对 RELEASE 版本不发生作用。

3 内存漏洞的检查

也许你已经知道，在 C++和 C 语言中指针问题也就是内存申请与释放是一个令人头疼的事情，假如你申请了内存，但没有释放，并且你的程序需要长时间地运行，那么，系统的资源将逐渐减少，当系统的资源全部被用完时，系统将会崩溃。所以在开发程序的过程中一定要保证资源的完全释放。下面我们来介绍内存漏洞的检查。

也许你会问，系统是怎样支持内存漏洞的检查的？其实在你的 Debug 版本中所有的有关内存分配的函数都是被重载过的，具体过程是这样的，当你的程序申请内存时，它首先调用一般的内存分配函数分配一块稍大的内存块。在这一内存块中分为四个小块：Heap Information, buffer , User memory block, buffer。第一块为有关堆的信息，比如，申请该内存的地点(文件名，行号)，此内存块的类型(如整型，浮点，或某一类的对象)等等。第二块是一个缓冲区,用于截获用户对其申请内存使用越界的情况。第三块是真正给用户的内存，返回的指针也是指向这儿。第四块也是一个缓冲区，作用同第二块。

当你申请的内存均被记录在案后，要检查内存漏洞就比较容易了，粗略地说，假如你要检查某一程序段是否有内存漏洞，你只需在这一程序 段的开始要求系统为你做一个内存使用情况的映象，记录下程序开始时的内存使用情况，然后在程序段的末尾再使系统为你做一次内存映象，比较两次映象，以检查是否有没释放的内存，假如有未释放的内存，根据这一块中有关分配情况的信息来告诉用户在那儿申请的内存没释放。

具体地讲检查内存漏洞需要以下几个步骤：

- 在你所检测的程序段的开始处建立一个 CmemoryState 对象，调用其成员函数 Checkpoint，以取得当前内存使用情况的快照；
- 在你所检测的程序段的末尾处再建立一个 CmemoryState 对象，调用其成员函数 Checkpoint ,以取得当前内存使用情况的快照；
- 再建立第三个 CmemoryState 对象，调用其成员函数 Difference,把第一个 CmemoryState 对象和第二个 CmemeoryState 对象作为其参数.,如果两次内存快照不相同，则该函数返回非零，说明此程序 段中有内存漏洞。下面我们来看一个典型的例子：

```
// Declare the variables needed
#ifdef _DEBUG
```

```
        CMemoryState oldMemState, newMemState, diffMemState;
        OldMemState.Checkpoint();
    #endif
   // do your memory allocations and deallocations...
        CString s = "This is a frame variable";
        // the next object is a heap object
        CPerson* p = new CPerson( "Smith", "Alan", "581_0215" );
    #ifdef _DEBUG
        newMemState.Checkpoint();
        if( diffMemState.Difference( oldMemState, newMemState ) )
        {
        TRACE( "Memory leaked!\n" );
    }
#endif
```