高级程序设计课程设计指导书

洪中华 周汝雁 耿永胜 编著

上海海洋大学海洋智能信息实验教学示范中心

实验一 使用 Vc++6.0 环境，控制台 main 函数创建与运行简单的窗口

一、实验目的

熟悉使用 Vc++6.0 环境，了解控制台 main 函数的使用。

二、实验环境

Windows10 、Visual Studio2017

三、实验内容

1.创建窗口

四、实验步骤（描述详细过程）

1.创建项目

1）双击桌面上的“Visual Studio2017”图标。

2）弹出 VS2017 的主界面，选择 Windows 桌面应用程序，如下图所示。

图 1-1. Visual Studio2017 的主界面

图 1-2. 项目创建

3）创建完成生成如下代码。

```
// My_windows.cpp : 定义应用程序的入口点。
//

#include "framework.h"
#include "My_windows.h"

#define MAX_LOADSTRING 100

// 全局变量:
HINSTANCE hInst;                                // 当前实例
WCHAR szTitle[MAX_LOADSTRING];                  // 标题栏文本
WCHAR szWindowClass[MAX_LOADSTRING];            // 主窗口类名

// 此代码模块中包含的函数的前向声明:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

4）在 wWinmain 函数中输入如下代码。

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: 在此处放置代码。

    // 初始化全局字符串
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_MYWINDOWS, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);

    // 执行应用程序初始化:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_MYWINDOWS));

    MSG msg;

    // 主消息循环:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}
```

5）然后再下面写一个注册窗口的函数。

```
//
//  函数: MyRegisterClass()
//
//  目标: 注册窗口类。
//
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEXW wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);

    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc    = WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, MAKEINTRESOURCE(IDI_MYWINDOWS));
    wcex.hCursor        = LoadCursor(nullptr, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = MAKEINTRESOURCEW(IDC_MYWINDOWS);
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI_SMALL));

    return RegisterClassExW(&wcex);
}
```

6）写一个实例句柄函数，用来创建和显示主程序窗口。

```
//
//   函数: InitInstance(HINSTANCE, int)
//
//   目标: 保存实例句柄并创建主窗口
//
//   注释:
//
//        在此函数中，我们在全局变量中保存实例句柄并
//        创建和显示主程序窗口。
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   hInst = hInstance; // 将实例句柄存储在全局变量中

   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);

   return TRUE;
}
```

7）写一个处理主窗口的消息函数。

```
//  函数: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  目标: 处理主窗口的消息。
//
//  WM_COMMAND  - 处理应用程序菜单
//  WM_PAINT    - 绘制主窗口
//  WM_DESTROY  - 发送退出消息并返回
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // 分析菜单选择:
            switch (wmId)
            {
            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: 在此处添加使用 hdc 的任何绘图代码...
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

8）点击运行就可得到如下图界面

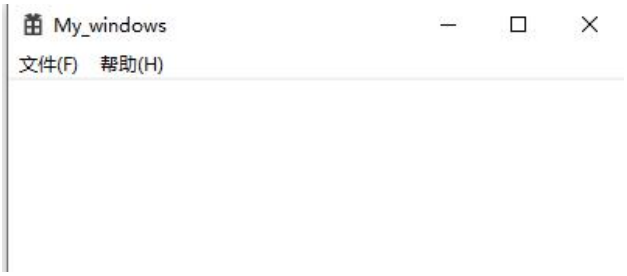


图 1-3. 窗口程序执行结果

实验二 windows 应用程序框架

一、实验目的

熟悉 Windows 应用环境框架

二、实验环境

Vc++ 6.0、Windows10

三、实验内容

1.创建窗口类函数

四、实验步骤

1）相关知识

在用 VisualC++开发面向对象应用程序时,主要使用了两种方法，一种是使用 Windows 提供的 Windows API 函数,另一种方法是直接使用 Microsoft 提供的 MFC 类库。

API 是应用程序编程接口(Application Programming Interface) 的缩写，WindowsAPI 是 Windows 系统和 Windows 应用程序间的标准程序接口。API 为应用程序提供系统的各种特殊函数及数据结构定义,Windows 应用程序可以利用上千个标准 API 函数调用系统功能。

根据 Windows API 函数完成的功能,可将其分为三类。

●窗口管理函数:实现窗口的创建、移动和修改功能。

●图形设备(GDI)函数:实现与设备无关的图形操作功能。

●系统服务函数:实现与操作系统有关的多种功能。

MFC 类库集成了大量已经预先定义好的类,用户可以根据编程的需要调用相应的类,或根据需要自定义有关的类。本书将重点讲述 API 函数及 MFC 类库的应用,并通过一些列实例来加深对它们的理解。

利用 WindowsAPI 函数和 MFC 类库编写 Windows 应用程序与编写 DOS 应用程序有很大区别,掌握 Windows 编程方法必须首先了解以下内容:

●窗口的概念。

●事件驱动的概念。

●消息及其在编程中的应用。

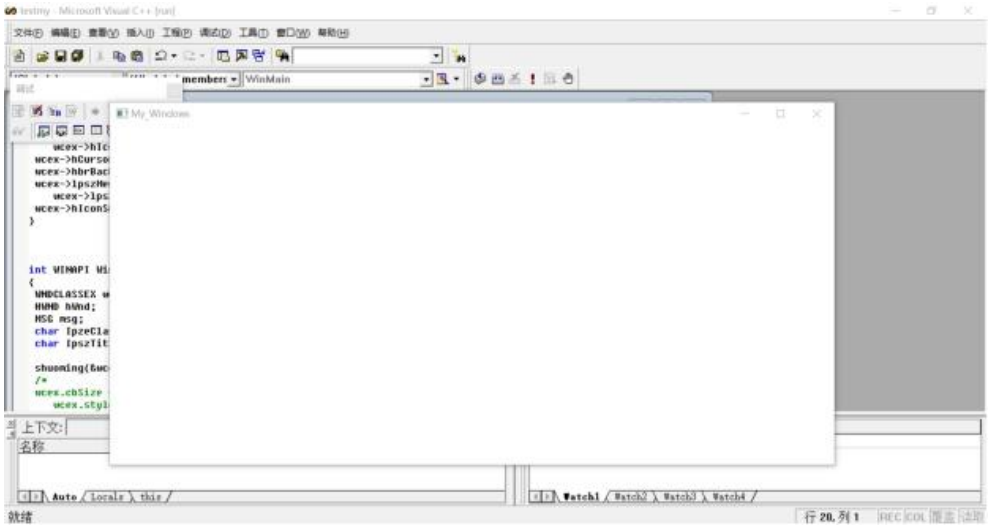
●对象与句柄。

2）编程要求

编写 windows 应用程序，并将窗口的建立编写一个函数，然后在主程序里调用。

3）测试说明

运行程序，成功生成一个窗口，运行成功。



4）代码如下

```
#include<windows.h>
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

Void create(WNDCLASSEX* wcex,HINSTANCE hInstance,char IpzeClassName[])
{
 wcex->cbSize = sizeof(WNDCLASSEX); //窗口类的大小
      wcex->style= 0;                    //窗口类型为默认类型
      wcex->lpfnWndProc = WndProc; //窗口处理函数为 WndProc
      wcex->cbClsExtra      = 0;          //窗口类无扩展
      wcex->cbWndExtra      = 0;   //窗口实例无扩展
      wcex->hInstance       = hInstance;   //当前实例句柄
      wcex->hIcon=  LoadIcon(hInstance,  MAKEINTRESOURCE  (IDI_APPLICATION));//
窗口的图标为默认图标
      wcex->hCursor    = LoadCursor(NULL, IDC_ARROW);
      wcex->hbrBackground    = (HBRUSH)GetStockObject(WHITE_BRUSH); //窗口背景为白
色
      wcex->lpszMenuName     = NULL;   //窗口中无菜单
      wcex->lpszClassName    = IpzeClassName;//窗口类名为“窗口示例”
      wcex->hIconSm                                    =      LoadIcon(wcex->hInstance,
MAKEINTRESOURCE(IDI_APPLICATION));
```

```
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,LPSTR lpszCmdLine,int nCmdShow)
{
 WNDCLASSEX wcex;
 HWND hWnd;
 MSG msg;
 char IpzeClassName[] ="window";
 char IpszTitle[]="My_Windows";

Creat(&wcex,hInstance,IpzeClassName);
//-------以下进行窗口类的注册 -------
 if(!RegisterClassEx(&wcex))
 {
 MessageBox(NULL, "窗口类注册失败!","窗口注册", NULL);
 return 1;
 }
 //hWnd=create(IpzeClassName,IpszTitle,hInstance);
 hWnd=CreateWindow
 (
  IpzeClassName,  //窗口类名
  IpszTitle,  //窗口实例的标题名
 WS_OVERLAPPEDWINDOW,  //窗口的风格
 CW_USEDEFAULT,
 CW_USEDEFAULT,  //窗口左上角坐标为缺省值
 CW_USEDEFAULT,
 CW_USEDEFAULT,,  //窗口的高和宽为缺省值
 NULL,  //此窗口无父窗口
 NULL,  //此窗口无主菜单
 hInstance,  //创建此窗口的应用程序的当前句柄
 NULL  //不使用该值
 );
 ShowWindow(hWnd, nCmdShow) ;
 UpdateWindow(hWnd);
 while( GetMessage(&msg, NULL, 0, 0))
 {
 TranslateMessage( &msg) ;
 DispatchMessage( &msg) ;
 }
 return msg.wParam; //消息循环结束即程序终止时将信息返回系统
}

LRESULT CALLBACK WndProc( HWND hwnd, UINT message,WPARAM
```

```
wParam,LPARAM lParam)
      {
       switch(message)
          {
       case WM_DESTROY:
       PostQuitMessage(0);
       default:   //缺省时采用系统消息缺省处理函数
       return DefWindowProc(hwnd,message,wParam,lParam);
       }
       return(0);
      }
```

实验三模拟时钟

一、 实验目的

利用 C++编写一个时钟

二、实验环境

Vc++ 6.0、Windows10

三、实验内容

绘制一个模拟时钟，要求表面为一个粉色的圆，并带有刻度、秒针、分针、时钟。

四、实验步骤（描述详细过程）

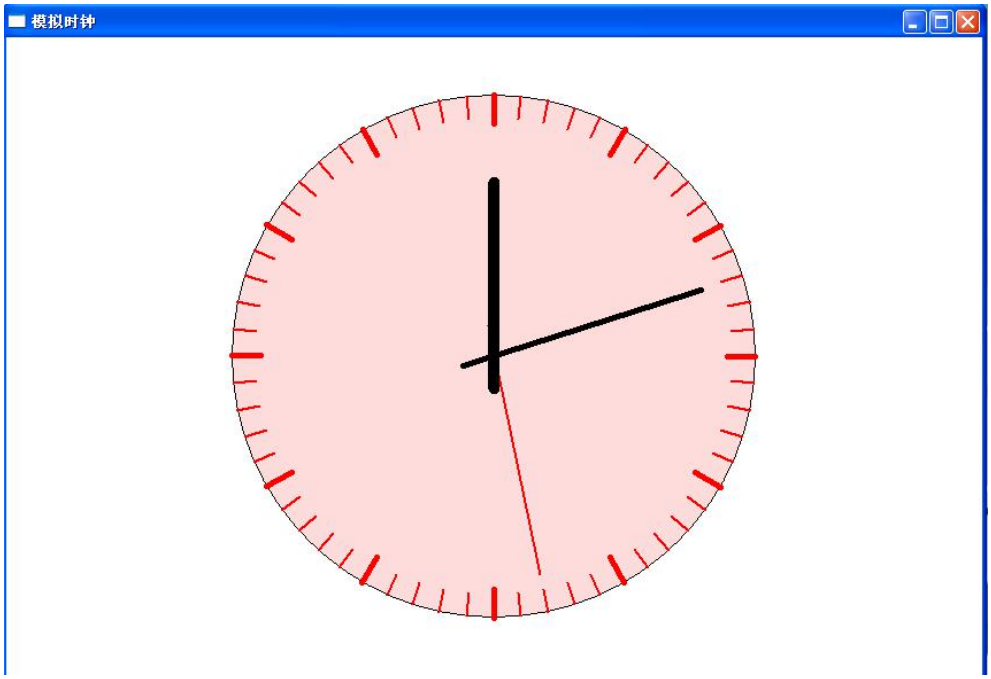
1）相关知识

Windows 图形设备接口(GDI）是为与设备无关的图形设计的。所谓设备的无关性，就是操作系统屏蔽了硬件设备的差异，因而设备无关性能使用户编程时无需考虑特殊的硬件设置 GDI 负责系统与用户或绘图程序之间的信息交换，并控制在输出设备上显示图形或文字， 是 Windows 系统的重要组成部分当用户区的内容需要刷新时， 系统向应用程序消息队列发送 WM_PAINT 消息，系统在应用程序的消息队列中加入该消息，以通知窗口函数执行刷新处理。

2）编程要求

绘制一个模拟时钟，要求表面为一个粉色的圆，并带有刻度、秒针、分针、时钟。

3）测试说明



4）代码如下

```
HDC hDC;
PAINTSTRUCT ps;
HBRUSH hBrush;
HPEN hPen;
RECT clientRect;
static TimeStructure x;
float sita=0;
int xOrg,yOrg,rSec,rMin,rHour,rClock,xBegin,xEnd,yBegin,yEnd;
switch (message)
{
case WM_CREATE:   //创建窗口时，响应的消息
      SetTimer(hWnd,9999,1000,NULL); //设置定时器
      break;
case WM_PAINT:
      {x.sec++;
      AdjustTime(&x);
      hDC=BeginPaint(hWnd,&ps);
      GetClientRect(hWnd,&clientRect);   //获取客户区的尺寸
      hPen=(HPEN)GetStockObject(BLACK_PEN);
      hBrush=CreateSolidBrush(RGB(255,220,220));//创建粉红色的单色画刷
      SelectObject(hDC,hPen);                  //选择画笔
      SelectObject(hDC,hBrush);                //选择画刷
      xOrg=(clientRect.left+clientRect.right)/2;
      yOrg=(clientRect.top+clientRect.bottom)/2;
      rClock=min(xOrg,yOrg)-50;
```

```
rSec=rClock*6/7;
rMin=rClock*5/6;
rHour=rClock*2/3;
Ellipse(hDC,xOrg-rClock,yOrg-rClock,xOrg+rClock,yOrg+rClock);
for(int i=0;i<60;i++)
{
    if(i%5)
    {
        hPen=CreatePen(PS_SOLID,2,RGB(255,0,0));
        SelectObject(hDC,hPen);
        xBegin=xOrg+rClock*sin(2*3.1415926*i/60);
        yBegin=yOrg+rClock*cos(2*3.1415926*i/60);
        MoveToEx(hDC,xBegin,yBegin,NULL);
        xEnd=xOrg+(rClock-20)*sin(2*3.14159*i/60);
        yEnd=yOrg+(rClock-20)*cos(2*3.14159*i/60);
    }
    else          //绘制表面表面的非整点刻度
    {
        hPen=CreatePen(PS_SOLID,5,RGB(255,0,0));
        SelectObject(hDC,hPen);
        xBegin=xOrg+rClock*sin(2*3.1415926*i/60);
        yBegin=yOrg+rClock*cos(2*3.1415926*i/60);
        MoveToEx(hDC,xBegin,yBegin,NULL);
        xEnd=xOrg+(rClock-25)*sin(2*3.1415926*i/60);
        yEnd=yOrg+(rClock-25)*cos(2*3.1415926*i/60);
    }
    LineTo(hDC,xEnd,yEnd);
    DeleteObject(hPen);
}
hPen=CreatePen(PS_SOLID,2,RGB(255,0,0));
SelectObject(hDC,hPen);
sita=2*3.1415926*x.sec/60;
xBegin=xOrg+(int)(rSec*sin(sita));
yBegin=yOrg-(int)(rSec*cos(sita));
xEnd=xOrg+(int)(rClock*sin(sita+3.1415926)/8);
yEnd=yOrg-(int)(rClock*cos(sita+3.1415926)/8);
MoveToEx(hDC,xBegin,yBegin,NULL);
LineTo(hDC,xEnd,yEnd);
hPen=CreatePen(PS_SOLID,5,RGB(0,0,0));
SelectObject(hDC,hPen);
sita=2*3.1415926*x.min/60;
xBegin=xOrg+(int)(rMin*sin(sita));
yBegin=yOrg-(int)(rMin*cos(sita));
xEnd=xOrg+(int)(rClock*sin(sita+3.1415926)/8);
```

```
            yEnd=yOrg-(int)(rClock*cos(sita+3.1415926)/8);
            MoveToEx(hDC,xBegin,yBegin,NULL);
            LineTo(hDC,xEnd,yEnd);
            hPen=CreatePen(PS_SOLID,10,RGB(0,0,0));
            SelectObject(hDC,hPen);
            sita=2*3.1415926*x.hour/12;
            xBegin=xOrg+(int)(rHour*sin(sita));
            yBegin=yOrg-(int)(rHour*cos(sita));
            xEnd=xOrg+(int)(rClock*sin(sita+3.1415926)/8);
            yEnd=yOrg-(int)(rClock*cos(sita+3.1415926)/8);
            MoveToEx(hDC,xBegin,yBegin,NULL);
            LineTo(hDC,xEnd,yEnd);
            DeleteObject(hPen);
            DeleteObject(hBrush);
            EndPaint(hWnd,&ps);
            break;}
        case WM_TIMER:       //响应定时器发出的定时消息
            if(wParam==9999)     //判断是否是设置的定时器发出的消息
                InvalidateRect(hWnd,NULL,true);     //刷新屏幕
            break;
        case WM_SIZE:                //窗口尺寸改变时，刷新窗口
            InvalidateRect(hWnd,NULL,true);
            break;
        };
        return 1;
}
```

实验四 五子棋游戏

一、实验目的

用 C++窗口类函数写一个五子棋小游戏

二、实验环境

VC++6.0、Windows10

三、实验内容

创建对话框，绘制五子棋，并实现五子棋的输赢判断。

四、实验步骤（描述详细过程）

1）相关知识

掌握 GDI 绘图，打印字符串等、鼠标事件响应。

2）编程要求

实现五子棋棋盘的格子定位，实现五子棋游戏正常进行。

3）测试说明

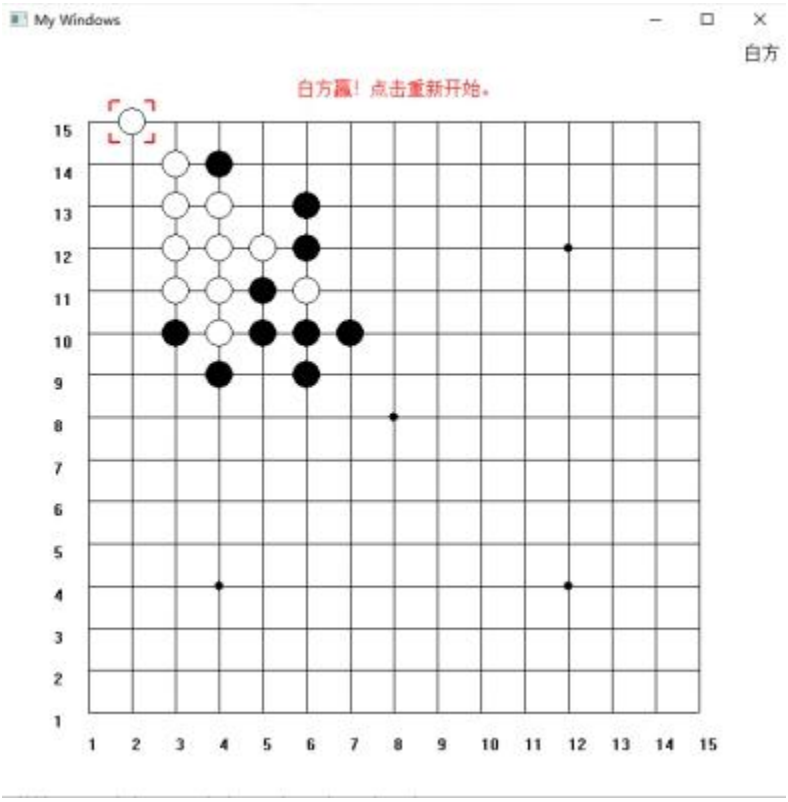


图 4-1. 五子棋测试

4）代码如下

```
#include<tchar.h>
#include<math.h>
#include<string.h>
#include<stdlib.h>
#include<windows.h>
#define MAX_LOADSTRING 100


LRESULT CALLBACK WndProc (HWND , UINT , WPARAM , LPARAM) ;
// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    WNDCLASS wndclass ;
    HWND hWnd ;
    MSG msg ;
    char szWindowClass[] = "窗口示例" ; //窗口类名
    char szTitle[] = "My Windows" ; //窗口标题
    wndclass.cbClsExtra = 0 ;//窗口类无扩展
    wndclass.cbWndExtra = 0 ;   //窗口实例无扩展
    wndclass.hbrBackground = (HBRUSH)GetStockObject(WHITE_BRUSH) ;//窗口背景为白色
    wndclass.hCursor = LoadCursor(NULL , IDC_ARROW) ;//窗口采用箭头光标
    wndclass.hIcon = LoadIcon(hInstance ,MAKEINTRESOURCE(IDI_APPLICATION)) ;//窗口的最小化图标为默认图标
    wndclass.hInstance = hInstance ;//当前实例句柄
    wndclass.lpfnWndProc = WndProc ;//窗口处理函数为 WndProc
    wndclass.lpszMenuName = NULL ; //窗口中无菜单
    wndclass.lpszClassName = szWindowClass ; //窗口的类名
    wndclass.style = CS_HREDRAW | CS_VREDRAW|CS_DBLCLKS   ; //窗口类型为默认类型
    //.
    //注册窗口类
    if(!RegisterClass(&wndclass)){
        MessageBox(NULL , _T("窗口类注册失败！") , _T("窗口注册") , NULL) ;
        return 1 ;
}


//创建窗口
    hWnd=CreateWindow(
        szWindowClass ,   //窗口类名
        szTitle , //窗口实例的标题名
        WS_OVERLAPPEDWINDOW , //窗口的样式
        CW_USEDEFAULT , CW_USEDEFAULT , //窗口左上角坐标为默认  //窗口右上角坐标为默认
        700 , 700 ,   //窗口宽度为默认  //窗口高度为默认
        NULL , //此窗口无父窗口
        NULL ,   //此窗口无主菜单
        hInstance,   //创建此窗口应用程序的当前句柄
        NULL   //不使用一个传递给窗口的参数值的指
```

```
            );

        if(!hWnd){
            MessageBox(NULL , "创建窗口失败!" ,("创建窗口") , NULL) ;
            return 1 ;
        }
        ShowWindow(hWnd,nCmdShow);   //显示窗口
        UpdateWindow(hWnd);   //绘制用户区
        //消息循环
        while(GetMessage(&msg,NULL,0,0)){

            TranslateMessage(&msg);   //将消息的虚拟键转换为字符信息
            DispatchMessage(&msg);     //将信消息发送到指定窗口函数
        }
        return(int) msg.wParam ; //程序终止时讲信息返回系统
}

HFONT CreateMyFont(){

        return CreateFont(
            0,//字体高度
            0,//字体宽度
            0,
            0,
            600,//字体粗细
            0,//字体倾斜  0 为非倾斜
            0,//下划线  0 为非
            0,//中划线  0 为非
            ANSI_CHARSET,
            OUT_DEFAULT_PRECIS,
            CLIP_DEFAULT_PRECIS,
            DEFAULT_QUALITY,
            DEFAULT_PITCH,
            "楷体_GB2312"//字体名
            ) ;
}
typedef struct point{
        int x ,y ;
        int color ;
        int sf_flag ;
}MAP;
MAP map[15][15] ;
int a , b ;
```

```
void init(){
    for(a = 0 ;a<15 ;a++){
        for(b = 0 ; b<15 ;b++){
            map[a][b].sf_flag = 0 ;//初始化每个点
            map[a][b].color = -1 ;
        }
    }
}
int c_flag = 0 ;
int w_b = 0 ;//0 下黑棋  非白棋
int pos_x ,pos_y ;
int x ,y ;
int Lbd_flag = 0;
int i_flag ,j_flag ;
int step1 = 0 ,step2=0 ,step3=0 ,step4=0 ;
char *str[]={"黑方赢！点击重新开始。" ,"白方赢！点击重新开始。","黑方","白方"};
char *index[] = {"1","2","3","4","5","6","7","8","9","10","11","12","13","14","15"};
int over_flag = 0 ;

//窗口函数（回调函数)
LRESULT CALLBACK WndProc( HWND hWnd,UINT message,WPARAM wParam,LPARAM lParam){

    HDC hDC ;
    RECT rect ;
    PAINTSTRUCT ps ;
    HFONT hF ;
    HBRUSH hBR1 ;
    HBRUSH hBR2 ;
    HPEN hPen1 ;
    HPEN hPen2 ;
    HCURSOR hcursor ;
    int i ,j;
    SIZE size ;
    switch(message)
    {
        //鼠标信息
    case WM_LBUTTONDOWN:
        pos_x = LOWORD(lParam) ;
        pos_y = HIWORD(lParam) ;
        Lbd_flag = 1 ;
        //InvalidateRect(hWnd ,NULL ,TRUE) ;
        InvalidateRect(hWnd ,&rect ,TRUE) ;
        break ;
```

```
case WM_MOUSEMOVE:
    pos_x = LOWORD(lParam) ;
    pos_y = HIWORD(lParam) ;
    if(over_flag==0)
    InvalidateRect(hWnd ,NULL ,TRUE) ;
    break ;
case WM_LBUTTONUP:
    break ;
case WM_DESTROY:
    PostQuitMessage(0) ;
    break ;
case WM_SIZE :
    InvalidateRect(hWnd ,NULL ,1) ;
    break ;
case WM_PAINT:
    hDC = BeginPaint(hWnd ,&ps) ;
    GetClientRect(hWnd, &rect) ;
    hPen1 = CreatePen(PS_SOLID ,0 ,RGB(0,0,0)) ;
    hPen2 = CreatePen(PS_SOLID ,2 ,RGB(255,0,0)) ;
    hBR1 = CreateSolidBrush(RGB(255,255,255)) ;
    hBR2 = CreateSolidBrush(RGB(0,0,0)) ;
    SelectObject(hDC,hPen1) ;
    if(c_flag)
        TextOut(hDC,5,5,str[2],strlen(str[2])) ;
    else{
        GetTextExtentPoint32(hDC ,str[3],strlen(str[3]),&size) ;
        TextOut(hDC,rect.right-size.cx-5,5,str[3],strlen(str[3])) ;
    }
    //确定坐标
    for(i = 0 ;i<15 ;i++){
        for(j = 0 ; j<15 ;j++){
            map[i][j].x = rect.right*(j+2)/18 ;
            map[i][j].y = rect.bottom*(i+2)/18 ;
        }
    }
    //画图
    for(j = 0 ;j<15 ;j++){
        MoveToEx(hDC,map[0][j].x,map[0][j].y ,NULL);
        LineTo(hDC,map[14][j].x ,map[14][j].y) ;
        TextOut(hDC,map[14][j].x,map[14][j].y+20,index[j],strlen(index[j])) ;
    }
    for(i = 0 ;i<15 ;i++){
        MoveToEx(hDC,map[i][0].x,map[i][0].y ,NULL);
        LineTo(hDC,map[i][14].x ,map[i][14].y) ;
```

```
        TextOut(hDC,map[i][0].x-30,map[i][0].y,index[14-i],strlen(index[14-i])) ;
            }

            //五个小黑点
            SelectObject(hDC,hBR2) ;
            Ellipse(hDC ,map[3][3].x-4 ,map[3][3].y-4,
                map[3][3].x+4 ,map[3][3].y+4) ;
            Ellipse(hDC ,map[3][11].x-4 ,map[3][11].y-4,
                map[3][11].x+4 ,map[3][11].y+4) ;
            Ellipse(hDC ,map[7][7].x-4 ,map[7][7].y-4,
                map[7][7].x+4 ,map[7][7].y+4) ;
            Ellipse(hDC ,map[11][3].x-4 ,map[11][3].y-4,
                map[11][3].x+4 ,map[11][3].y+4) ;
            Ellipse(hDC ,map[11][11].x-4 ,map[11][11].y-4,
                map[11][11].x+4 ,map[11][11].y+4) ;
            //画小边界框与鼠标的变化
            for(i = 0 ;i < 15 ;i++){
                for(j = 0 ;j<15 ;j++){
                    if(abs(pos_x-map[i][j].x)<rect.right/36&&
abs(pos_y-map[i][j].y)<rect.bottom/36 ){
                        x = map[i][j].x ;
                        y = map[i][j].y ;
                        hcursor = LoadCursor(NULL,IDC_SIZEALL);
                        SetCursor(hcursor) ;
                        break ;
                    }
                }
            }
            SelectObject(hDC,hPen2) ;
            MoveToEx(hDC,x-rect.right/36 ,y-rect.bottom/62 ,NULL) ;
            LineTo(hDC ,x-rect.right/36 ,y-rect.bottom/36) ;
            LineTo(hDC ,x-rect.right/62 ,y-rect.bottom/36) ;
        MoveToEx(hDC,x+rect.right/36 ,y-rect.bottom/62 ,NULL) ;
            LineTo(hDC ,x+rect.right/36 ,y-rect.bottom/36) ;
            LineTo(hDC ,x+rect.right/62 ,y-rect.bottom/36) ;
        MoveToEx(hDC,x-rect.right/36 ,y+rect.bottom/62 ,NULL) ;
            LineTo(hDC ,x-rect.right/36 ,y+rect.bottom/36) ;
            LineTo(hDC ,x-rect.right/62 ,y+rect.bottom/36) ;

            MoveToEx(hDC,x+rect.right/36 ,y+rect.bottom/62 ,NULL) ;
            LineTo(hDC ,x+rect.right/36 ,y+rect.bottom/36) ;
            LineTo(hDC ,x+rect.right/62 ,y+rect.bottom/36) ;

            //点击下棋
```

```
            if(Lbd_flag&&over_flag==0){
                for(i = 0 ;i<15 ;i++){
                    for(j = 0 ; j<15 ;j++){
                        if(!map[i][j].sf_flag  &&abs(pos_x-map[i][j].x)<rect.right/36  &&
abs(pos_y-map[i][j].y)<rect.bottom/36 ){
                            map[i][j].sf_flag = 1 ;//标示该点已经被下了
                            i_flag = i ;
                            j_flag = j ;
                            if(c_flag){//1 代表黑色  0 为白色
                                map[i][j].color = c_flag;
                                c_flag = 0 ;
                            }else{
                                map[i][j].color = c_flag ;
                                c_flag = 1 ;
                            }
                        }
                    }
                }
            }
            //初始化
            if(over_flag){
                init();
                over_flag = 0 ;
            }
            SelectObject(hDC,hPen1) ;
            //画出棋子
            for(i = 0 ;i<15 ;i++){
                for(j = 0 ; j<15 ;j++){
                    if(map[i][j].sf_flag){
                        if(map[i][j].color==1){
                            SelectObject(hDC,hBR2) ;
    Ellipse(hDC,map[i][j].x-12,map[i][j].y-12,
                                map[i][j].x+12,map[i][j].y+12) ;
                        }else{
                            SelectObject(hDC,hBR1) ;
                            Ellipse(hDC,map[i][j].x-12 ,map[i][j].y-12,
                                map[i][j].x+12,map[i][j].y+12) ;
                        }
                    }
                }
            }

            if(Lbd_flag){
                //判断一行是否有五子
```

```
for(j=j_flag ;j<15 ;j++){
    //往右判断
    if(map[i_flag][j].sf_flag){
        if(map[i_flag][j].color==map[i_flag][j_flag].color){
            step1++ ;
        }else
            break ;
    }
}
for(j=j_flag;j>0;j--){
    //往左
    if(map[i_flag][j].sf_flag){
        if(map[i_flag][j].color==map[i_flag][j_flag].color)
            step1++ ;
        else
            break ;
    }
}

//判断一列是否有五子
for(i=i_flag ;i<15 ;i++){
    //往上判断
    if(map[i][j_flag].sf_flag){
        if(map[i][j_flag].color==map[i_flag][j_flag].color){
            step2++ ;
        }else
            break ;
    }
}
for(i=i_flag;i>=0;i--){
    //往下
    if(map[i][j_flag].sf_flag){
        if(map[i][j_flag].color==map[i_flag][j_flag].color)
            step2++ ;
        else
            break ;
    }
}
//判断\的方向
for(i=i_flag,j=j_flag ;i<15 ,j<15;i++,j++){
    //往上判断
    if(map[i][j].sf_flag){
        if(map[i][j].color==map[i_flag][j_flag].color){
            step3++ ;
```

```
                }else
                    break ;
            }
        }
        for(i=i_flag,j=j_flag;i>0,j>0;i--,j--){
            //往下
            if(map[i][j].sf_flag){
                if(map[i][j].color==map[i_flag][j_flag].color)
                    step3++ ;
                else
                    break ;
            }
        }
        //判断/的方向
        for(i=i_flag,j=j_flag ;i>0 ,j<15;i--,j++){
            //往上判断
            if(map[i][j].sf_flag){
                if(map[i][j].color==map[i_flag][j_flag].color){
                    step4++ ;
                }else
                    break ;
            }
        }
        for(i=i_flag,j=j_flag;i<15,j>0;i++,j--){
            //往下
            if(map[i][j].sf_flag){
                if(map[i][j].color==map[i_flag][j_flag].color)
                    step4++ ;
                else
                    break ;
            }
        }

        if(step1>5||step2>5||step3>5||step4>5){
            SetTextColor(hDC,RGB(255,0,0));
            SelectObject(hDC,hF) ;
            GetTextExtentPoint32(hDC ,str[0],strlen(str[0]),&size) ;
            if(map[i_flag][j_flag].color==1)

TextOut(hDC,(rect.right/2-size.cx/2),rect.bottom/18,str[0],strlen(str[0])) ;
            if(map[i_flag][j_flag].color==0)

TextOut(hDC,(rect.right/2-size.cx/2),rect.bottom/18,str[1],strlen(str[1])) ;
            over_flag = 1 ;
```

```
                }
                step1 = 0 ;
                step2 = 0 ;
                step3 = 0 ;
                step4 = 0 ;
                Lbd_flag = 0 ;
            }
            DeleteObject(hBR1) ;
            DeleteObject(hBR2) ;
            DeleteObject(hPen1) ;
            DeleteObject(hPen2) ;
            EndPaint(hWnd,&ps) ;
            break ;
        default:
            return DefWindowProc(hWnd , message , wParam ,lParam) ;
            break ;
            }

            return    0 ;
    }


    // Mesage handler for about box.
    LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
LPARAM lParam)
    {
        switch (message)
        {
            case WM_INITDIALOG:
                    return TRUE;

            case WM_COMMAND:
                if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                {
                    EndDialog(hDlg, LOWORD(wParam));
                    return TRUE;
                }
                break;
        }
        return FALSE;
    }
```

实验五 资源

一、实验目的

熟悉 MFC 中的菜单、基本控件。

二、实验环境

Vc++ 6.0、Windows10

三、 实验内容

菜单由主要由这些部分组成：窗口主菜单条、下拉式菜单框、菜单项热键标识、菜单项加速键标识、菜单项分隔线。

四、实验步骤

1）创建一个 Hello World 窗口。

2）创建一个窗口菜单的构架，单击 “创建统计计算菜单项”动态地创建“统计计算”菜单，之后， “创建统计计算菜单项”变成不可操作，而原先不可操作的 “删除统计计算菜单项”变成可操作，执行“删除统计计算菜单项”菜单命令删除“统计计算”菜单。

图 5-1. 菜单编辑

3）添加代码

```
#include "stdafx.h"
#include "resource.h"
```

```
#define MAX_LOADSTRING 100
// Global Variables:
HINSTANCE hInst;                                // current instance
TCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
TCHAR szWindowClass[MAX_LOADSTRING];            // The title bar text

// Foward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
LRESULT CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPSTR     lpCmdLine,
                     int       nCmdShow)
{
    // TODO: Place code here.
    MSG msg;
    HACCEL hAccelTable;
    // Initialize global strings
    LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadString(hInstance, IDC_LIST1, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }
    hAccelTable = LoadAccelerators(hInstance, (LPCTSTR)IDC_LIST1);

    // Main message loop:
    while (GetMessage(&msg, NULL, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return msg.wParam;
}
// FUNCTION: MyRegisterClass()
```

```
//  PURPOSE: Registers the window class.
//  COMMENTS:
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style          = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = (WNDPROC)WndProc;
    wcex.cbClsExtra     = 0;
    wcex.cbWndExtra     = 0;
    wcex.hInstance      = hInstance;
    wcex.hIcon          = LoadIcon(hInstance, (LPCTSTR)IDI_LIST1);
    wcex.hCursor        = LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground  = (HBRUSH)(COLOR_WINDOW+1);
    wcex.lpszMenuName   = (LPCSTR)IDC_LIST1;
    wcex.lpszClassName  = szWindowClass;
    wcex.hIconSm        = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);
    return RegisterClassEx(&wcex);
}

//
//   FUNCTION: InitInstance(HANDLE, int)
//   PURPOSE: Saves instance handle and creates main window
//   COMMENTS:
//
//        In this function, we save the instance handle in a global variable and
//        create and display the main program window.
//
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
   HWND hWnd;
   hInst = hInstance; // Store instance handle in our global variable

   hWnd = CreateWindow(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

   if (!hWnd)
   {
      return FALSE;
   }

   ShowWindow(hWnd, nCmdShow);
   UpdateWindow(hWnd);
```

```
    return TRUE;
}
HMENU hmenu,haddmenu; //定义菜单句柄
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    int wmId, wmEvent;
    PAINTSTRUCT ps;
    HDC hdc;
    TCHAR szHello[MAX_LOADSTRING];
    LoadString(hInst, IDS_HELLO, szHello, MAX_LOADSTRING);
    HBITMAP    hBm;
    switch (message)
    {
        case WM_COMMAND:
            wmId       = LOWORD(wParam);
            wmEvent = HIWORD(wParam);
            // Parse the menu selections:
            switch(wmId)
            {
                    case IDM_ADDMENU:        //在主菜单中添加弹出式菜单
     hmenu=GetMenu(hWnd);          //获取主菜单句柄
     haddmenu=CreateMenu();        //动态创建菜单
     //在创建的菜单中增加菜单项
     AppendMenu(haddmenu,MF_ENABLED,IDM_qiuhe,"求和");
     AppendMenu(haddmenu,MF_ENABLED,IDM_fangcha,"方差");
     AppendMenu(haddmenu,MF_ENABLED,IDM_pinjunzhi,"平均值");
     AppendMenu(haddmenu,MF_ENABLED,IDM_junfanggen,"均方根");
     //将创建的弹出式菜单插入主菜单中
     InsertMenu(hmenu,2,MF_POPUP|MF_BYPOSITION,
                    (UINT)haddmenu,"统计计算(&C)");
     //相应改变菜单中有关绘图统计计算菜单项的属性
     EnableMenuItem(hmenu,IDM_ADDMENU,MF_GRAYED);
     EnableMenuItem(hmenu,IDM_DELMENU,MF_ENABLED);
     DrawMenuBar(hWnd);            //重新显示窗口菜单
        break;
        case IDM_DELMENU:
    DeleteMenu(hmenu,2,MF_BYPOSITION);  //删除统计计算菜单项
    //相应改变“文件”菜单中有关统计计算菜单项的属性
    EnableMenuItem(hmenu,IDM_ADDMENU,MF_ENABLED);
    EnableMenuItem(hmenu,IDM_DELMENU,MF_GRAYED);
    DrawMenuBar(hWnd);            //重新显示窗口菜单
    break;
```

```
            case IDM_EXIT:
                SendMessage(hWnd,WM_DESTROY,0,0);
                break;
            }
            break;

                    case WM_PAINT:
                        hdc = BeginPaint(hWnd, &ps);
                        // TODO: Add any drawing code here...
                        RECT rt;
                        GetClientRect(hWnd, &rt);
                        DrawText(hdc, szHello, strlen(szHello), &rt, DT_CENTER);
                        EndPaint(hWnd, &ps);
                        break;
                    case WM_DESTROY:
                        DeleteObject(hBm);      //释放位图
                        PostQuitMessage(0);
                        break;
                    default:
                        return DefWindowProc(hWnd, message, wParam, lParam);
                }
                return 0;
            }
            // Mesage handler for about box.
            LRESULT CALLBACK About(HWND hDlg, UINT message, WPARAM wParam,
        LPARAM lParam)
            {
                switch (message)
                {
                    case WM_INITDIALOG:
                            return TRUE;
                    case WM_COMMAND:
                        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
                        {
                            EndDialog(hDlg, LOWORD(wParam));
                            return TRUE;
                        }
                        break;
                }
                return FALSE;
            }
```

实验六 扫雷

一、实验目的

掌握位图，扫雷游戏的创建破解。

二、实验环境

Vc++ 6.0、Windows10

三、实验内容

成功找出所有雷点。

四、实验步骤

1）样本基础信息

编写工具：从链接器版本判断为 VS2003。

类型：32 位如下图所示。

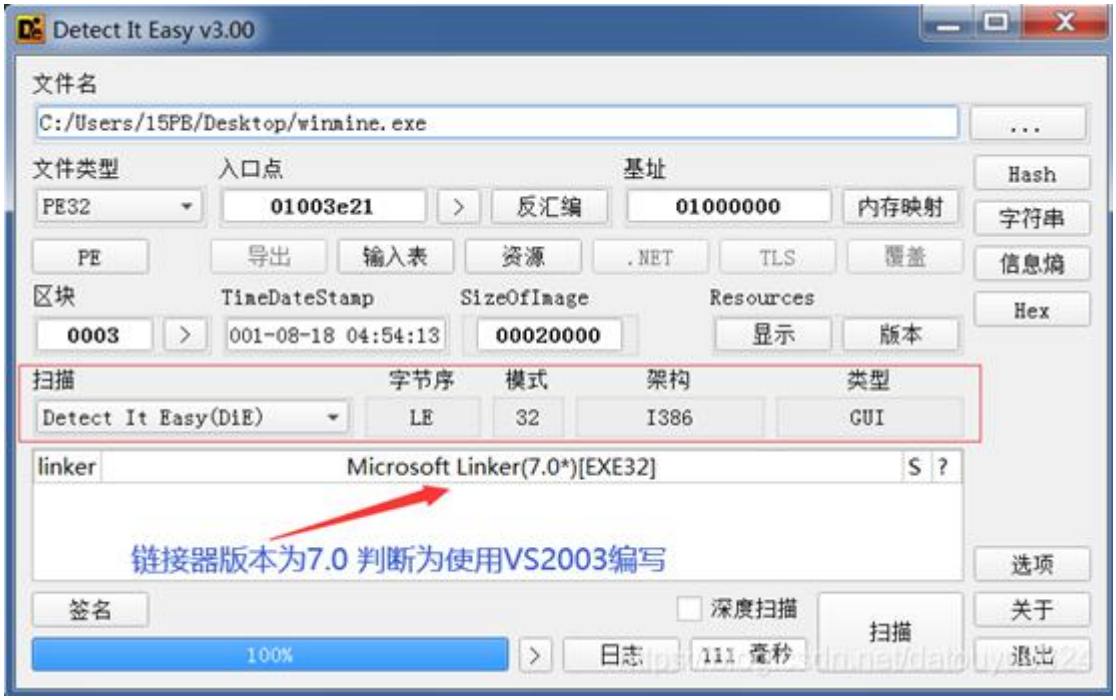


图 6-1. 链接器判读

2）CE 搜索数据

获取数据基本步骤，如下图所示。

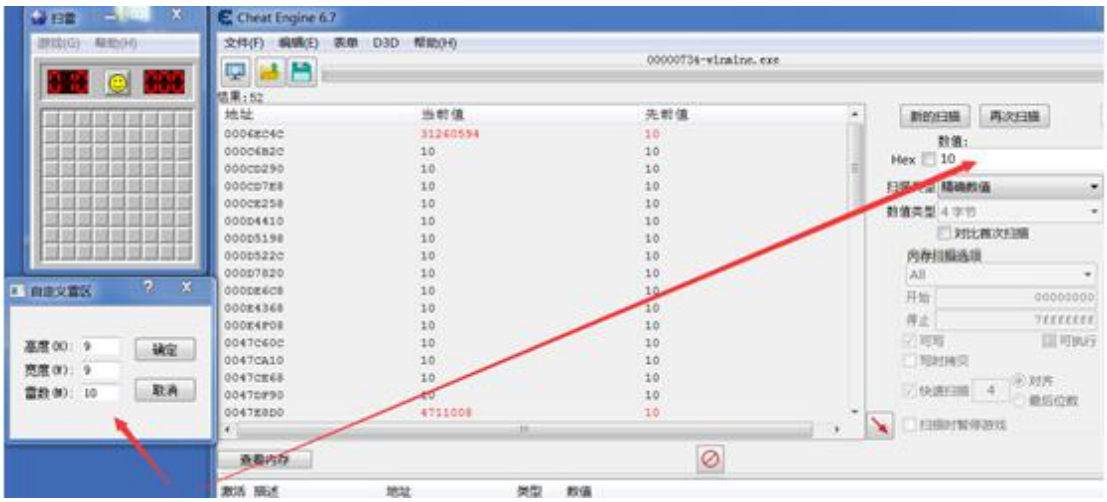


图 6-2. CE 搜索数据

多次修改获取到三个的高，宽，雷数的数据，且在 CE 中地址为绿色(静态地址，全局)，有多份可能为(代码冗余，实际显示需要)，如下图所示。

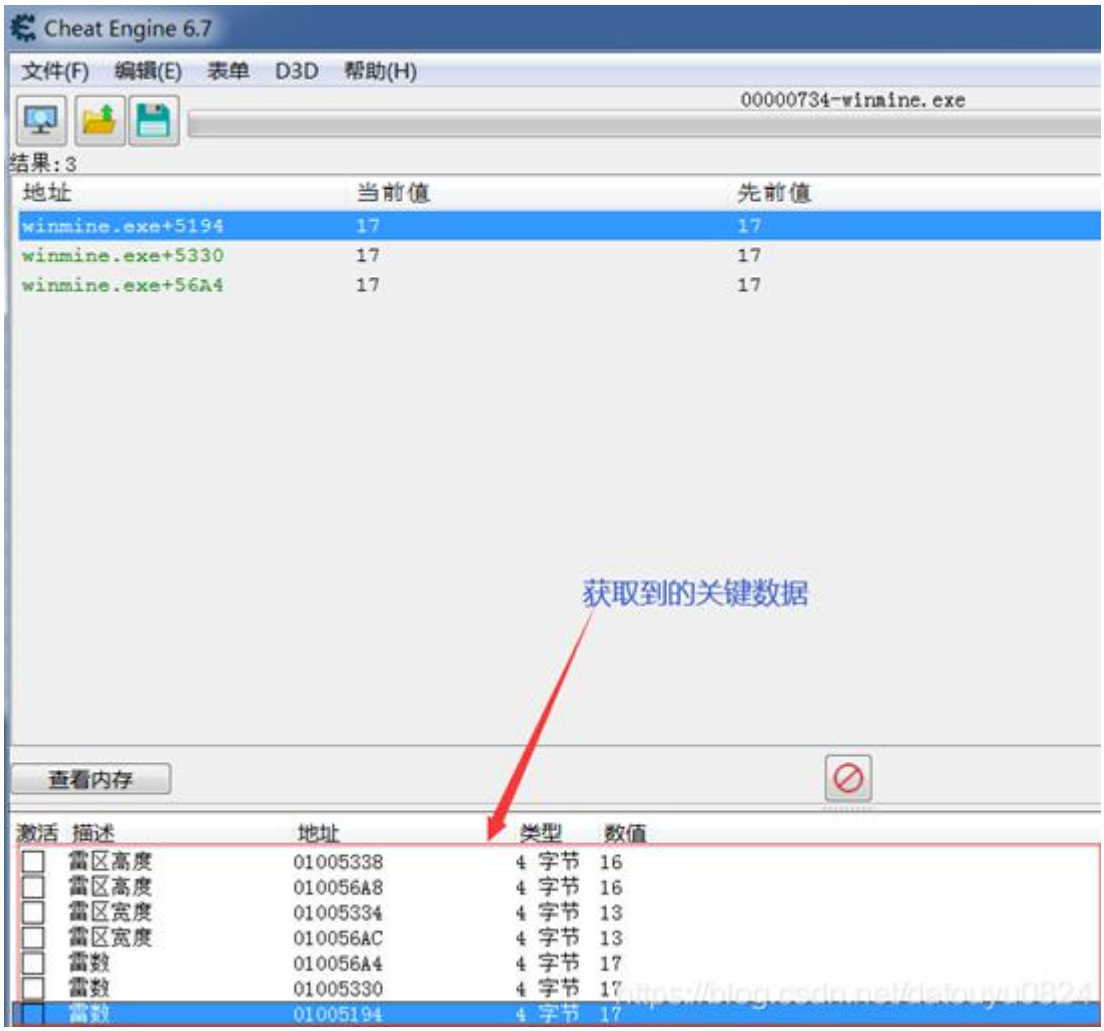


图 6-3. CE 获取雷数内存地址

3）验证数据有效性

在 VS2017 中编写 MFC DLL(静态编译，不依赖环境)，关键调试代码

```
//调式获取到的高度，宽度，雷数变量地址
PDWORD pHeigth = (PDWORD)0x01005338;
PDWORD pWidth = (PDWORD)0x01005334;
PDWORD pMineCount = (PDWORD)0x01005330;

//显示错误
BOOL CheckResult(BOOL result, WCHAR* str)
{
  if (!result)
  {
    OutputDebugString(str);
    return FALSE;
  }
  return TRUE;
}

//保存窗口句柄
HWND g_hWnd = 0;
//原来的回调函数
WNDPROC OldWinProc = 0;
//新的窗口回调函数
LRESULT CALLBACK NewWndProc(HWND hWnd, UINT uMsg, WPARAM wParam, LP
ARAM lParam)
{
  switch (uMsg)
  {
    //如果是键盘消息
    //一键扫雷思路：遍历扫雷数组中的元素，判断不为雷的区域，模拟鼠标点击
    case WM_KEYDOWN:
    {
      //f5
      if (wParam == VK_F5)
      {
        CString str;
        str.Format(L"高：%d，宽：%d，雷数：%d\n", *pHeigth, *pWidth, *pMineCount);
        OutputDebugString(str);
       }
    }
  }
  //CallWindowProc 不处理的消息让原来的回调函数处理
  return OldWinProc(hWnd, uMsg, wParam, lParam);
}
// CMFCmineApp 初始化
BOOL CMFCmineApp::InitInstance()
```

```
{
    CWinApp::InitInstance();
    //1. 通过查找窗口获取窗口句柄 spy++
    //通过窗口名，窗口类名获取窗口句柄
    g_hWnd = ::FindWindow(L"扫雷", L"扫雷");
    //判断是否找到
    if (CheckResult(g_hWnd != 0, L"未找到扫雷窗口\n") == 0)
        return 0;

    //2. 设置窗口回调函数
     OldWinProc = WNDPROC(SetWindowLong(g_hWnd, GWL_WNDPROC, LONG(New
WndProc)));
    //判断是否修改成功
    if (CheckResult(OldWinProc != 0, L"修改回调函数失败\n") == 0)
        return 0;
    return TRUE;
}
```

4）OD 分析代码逻辑，CE 中找到访问数据位置。

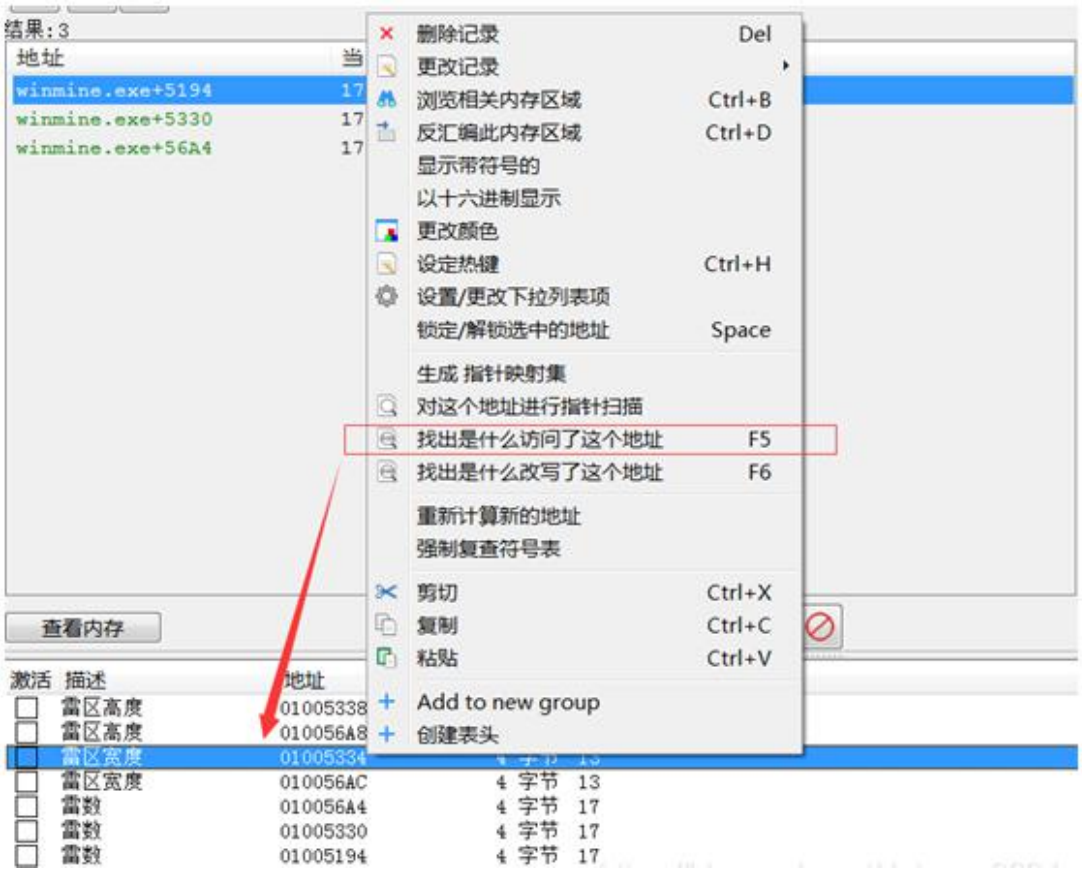


图 6-4. CE 获取雷区宽度内存地址

5）功能实现

在注入 DLL 中的消息回调响应鼠标移动，根据分析的转换算法将鼠标坐标转换成数组

坐标，判断当前雷区数组没区域是否为雷，在标题栏提示，主要代码如下：

```
//鼠标移动显示信息
    case WM_MOUSEMOVE:
    {
      CString sCaption;

      //x,y 坐标
      DWORD x = LOWORD(lParam);
      DWORD y = HIWORD(lParam);

      //转换为数组坐标
      x = (x + 4) >> 4;
      y = (y - 0x27) >> 4;

      //在雷区范围内
      if (0 < x && x <= *pWidth && 0 < y && y <= *pHeigth)
      {
        //如果判断为雷
        if (*(PBYTE(dwBaseAddr + y * 32 + x)) == MINE)
          sCaption.Format(L"扫雷 (x:%d,y:%d) 有雷", x, y);
        else
          sCaption.Format(L"扫雷 (x:%d,y:%d) 无雷", x, y);

        SetWindowText(g_hWnd, sCaption);
      }
    }
```

实验七 文件的读写

一、实验目的

了解 C++对文件读写操作。

二、实验环境

Visual Studio 2017、Windows10

三、实验内容

对文件进行读写。

文件打开方式：

ios::in 打开文件进行读操作，即读取文件中的数据

ios::out 打开文件进行写操作，即输出数据到文件中

ios::ate 打开文件时文件指针指向文件末尾，但是你可以在文件中的任何地方写数据

ios::app 打开文件不会清空数据，文件指针始终在文件末尾，因此只能在文件末尾写数据

ios:trunc 默认，若打开文件已存在，测清空文件的内容

ios::nocreate 若打开文件不存在则不建立，返回打开失败信息

ios::noreplace 打开文件时不能覆盖，若文件存在测返回打开失败信息

ios::binary 打开文件为二进制文件，否则为文本文件

四、实验步骤（描述详细过程）

1）创建项目

2）利用 IOS:IN 文件编写

3）实验结果

具体代码如下：

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

 // 输出空行
 void    OutPutAnEmptyLine()
 {
     cout<<"\n";
}
 // 读取方式: 逐词读取, 词之间用空格区分
void    ReadDataFromFileWBW()
 {
     ifstream fin("1.txt");
     string s;
     while( fin >> s )
     {
          cout << "Read from file: " << s << endl;
     }
}
 // 读取方式: 逐行读取, 将行读入字符数组, 行之间用回车换行区分
   void    ReadDataFromFileLBLIntoCharArray()
 {
     ifstream fin("data.txt");
     const int LINE_LENGTH = 100;
     char str[LINE_LENGTH];
```

```
    while( fin.getline(str,LINE_LENGTH) )
    {
        cout << "Read from file: " << str << endl;
    }
}
 // 读取方式: 逐行读取, 将行读入字符串, 行之间用回车换行区分
 // If you want to avoid reading into character arrays,
 // you can use the C++ string getline() function to read lines into strings
 void   ReadDataFromFileLBLIntoString()
 {
    ifstream fin("data.txt");
    string s;
    while( getline(fin,s) )
    {
        cout << "Read from file: " << s << endl;
    }
}

 // 带错误检测的读取方式
 void   ReadDataWithErrChecking()
 {
    string filename = "dataFUNNY.txt";
    ifstream fin( filename.c_str());
    if( !fin )
    {
        cout << "Error opening " << filename << " for input" << endl;
        exit(-1);
    }
}
 int   main()
 {
    ReadDataFromFileWBW(); //逐词读入字符串
    OutPutAnEmptyLine(); //输出空行
    ReadDataFromFileLBLIntoCharArray(); //逐词读入字符数组
    OutPutAnEmptyLine(); //输出空行
    ReadDataFromFileLBLIntoString(); //逐词读入字符串
    OutPutAnEmptyLine(); //输出空行
ReadDataWithErrChecking(); //带检测的读取
    return 0;
}
```

实验八 创建多窗口应用程序

一、实验目的

利用 C++编写两个窗口切换。

二、实验环境

Vc++ 6.0、Windows10

三、实验内容

实现多窗口的创建。

四、实验步骤（描述详细过程）

1）相关知识

由于对话框是一个特殊的窗口，所以该类是从 CWnd 类中派生出来的。对话框子层次结构包括通用对话框类 CDialog 以及支持文件选择、颜色选择、字体选择、打印、替换文本的公共对话框子类。

2）编程要求

实现多窗口的创建。

3）测试说明，如下图。

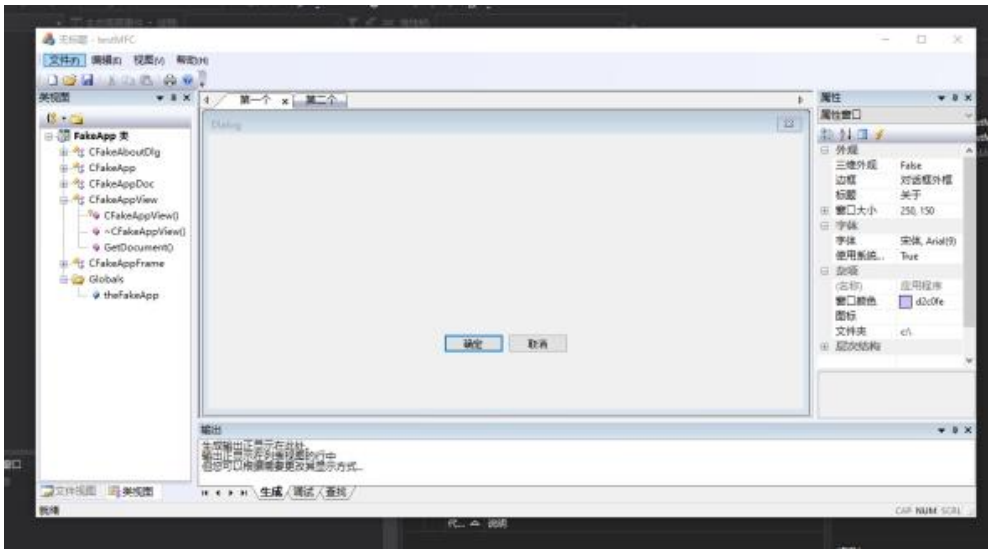


图 8-1. 窗口切换

4）具体代码

```
#include<windows.h>
LRESULT CALLBACK WndProc(HWND,UINT,WPARAM,LPARAM);

Void create(WNDCLASSEX* wcex,HINSTANCE hInstance,char IpzeClassName[])
{
 wcex->cbSize = sizeof(WNDCLASSEX); //窗口类的大小
 wcex->style= 0;                    //窗口类型为默认类型
 wcex->lpfnWndProc = WndProc; //窗口处理函数为 WndProc
 wcex->cbClsExtra      = 0;         //窗口类无扩展
 wcex->cbWndExtra      = 0;   //窗口实例无扩展
 wcex->hInstance       = hInstance;   //当前实例句柄
 wcex->hIcon= LoadIcon(hInstance, MAKEINTRESOURCE (IDI_APPLICATION));//窗口的图标为默认图标
 wcex->hCursor    = LoadCursor(NULL, IDC_ARROW);
 wcex->hbrBackground    = (HBRUSH)GetStockObject(WHITE_BRUSH); //窗口背景为白色
 wcex->lpszMenuName     = NULL;   //窗口中无菜单
 wcex->lpszClassName    = IpzeClassName;//窗口类名为“窗口示例”
 wcex->hIconSm          = LoadIcon(wcex->hInstance, MAKEINTRESOURCE(IDI_APPLICATION));
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInst,LPSTR lpszCmdLine,int nCmdShow)
{
 WNDCLASSEX wcex;
 HWND hWnd;
 MSG msg;
 char IpzeClassName[] ="window";
 char IpszTitle[]="My_Windows";
Creat(&wcex,hInstance,IpzeClassName);

 //-------以下进行窗口类的注册  -------
 if(!RegisterClassEx(&wcex))
 {
 MessageBox(NULL, "窗口类注册失败!","窗口注册", NULL);
 return 1;
 }
 //hWnd=create(IpzeClassName,IpszTitle,hInstance);

 hWnd=CreateWindow
 (
```

```
    IpzeClassName,  //窗口类名
    IpszTitle,  //窗口实例的标题名
   WS_OVERLAPPEDWINDOW,  //窗口的风格
   CW_USEDEFAULT,
   CW_USEDEFAULT,  //窗口左上角坐标为缺省值
   CW_USEDEFAULT,
   CW_USEDEFAULT,,  //窗口的高和宽为缺省值
   NULL,  //此窗口无父窗口
   NULL,  //此窗口无主菜单
   hInstance,  //创建此窗口的应用程序的当前句柄
   NULL  //不使用该值
   );


   ShowWindow(hWnd, nCmdShow) ;
   UpdateWindow(hWnd);
   while( GetMessage(&msg, NULL, 0, 0))
   {
   TranslateMessage( &msg) ;
   DispatchMessage( &msg) ;
   }
   return msg.wParam; //消息循环结束即程序终止时将信息返回系统
  }

  LRESULT CALLBACK WndProc( HWND hwnd, UINT message,WPARAM wParam,LPARAM lParam)
  {
   switch(message)
     {
   case WM_DESTROY:
   PostQuitMessage(0);
   default:  //缺省时采用系统消息缺省处理函数
   return DefWindowProc(hwnd,message,wParam,lParam);
   }
   return(0);
  }
```

实验九 窗口控件

一、实验目的

利用 C++编写一个简易计算器。

二、实验环境

Visual Studio2017、Windows10

三、实验内容

在窗口中实现加减乘除。

四、实验步骤（描述详细过程）

1）创建一个 MFC 工程，基于对话框即可。

2）如下测试。

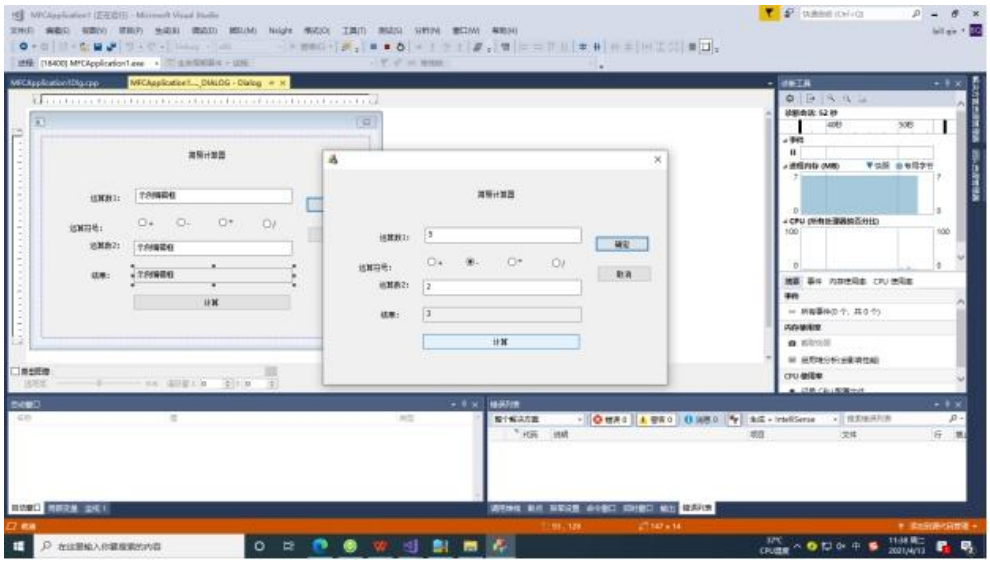


图 9-1. 简易计算器

3）代码如下：

```
// MFCApplication3Dlg.cpp: 实现文件
//

#include "pch.h"
#include "framework.h"
#include "MFCApplication3.h"
#include "MFCApplication3Dlg.h"
```

```
#include "afxdialogex.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif

// 用于应用程序“关于”菜单项的 CAboutDlg 对话框

class CAboutDlg : public CDialogEx
{
public:
	CAboutDlg();

// 对话框数据
#ifdef AFX_DESIGN_TIME
	enum { IDD = IDD_ABOUTBOX };
#endif

	protected:
	virtual void DoDataExchange(CDataExchange* pDX);	// DDX/DDV 支持

// 实现
protected:
	DECLARE_MESSAGE_MAP()
};

CAboutDlg::CAboutDlg() : CDialogEx(IDD_ABOUTBOX)
{
}

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()
// CMFCApplication3Dlg 对话框
CMFCApplication3Dlg::CMFCApplication3Dlg(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_MFCAPPLICATION3_DIALOG, pParent)
	, m_num1(0)
	, m_operator(0)
	, m_num2(0)
	, m_result(0)
{
```

```
        m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
    }
    void CMFCApplication3Dlg::DoDataExchange(CDataExchange* pDX)
    {
        CDialogEx::DoDataExchange(pDX);
        DDX_Text(pDX, IDC_NUM1, m_num1);
        DDX_Text(pDX, IDC_ADD, m_operator);
        DDX_Text(pDX, IDC_NUM2, m_num2);
        DDX_Text(pDX, IDC_RESULT, m_result);
        DDX_Radio(pDX, IDC_ADD, m_operator);
        DDX_Text(pDX, IDC_RESULT, m_result);
    }
    BEGIN_MESSAGE_MAP(CMFCApplication3Dlg, CDialogEx)
        ON_WM_SYSCOMMAND()
        ON_WM_PAINT()
        ON_WM_QUERYDRAGICON()
        ON_BN_CLICKED(IDC_BUTTON1,
&CMFCApplication3Dlg::OnBnClickedButton1)
        ON_BN_CLICKED(IDC_RADIO1, &CMFCApplication3Dlg::OnBnClickedRadio1)
        ON_EN_CHANGE(IDC_EDIT1, &CMFCApplication3Dlg::OnEnChangeEdit1)
        ON_EN_CHANGE(IDC_NUM1, &CMFCApplication3Dlg::OnEnChangeNum1)
        ON_BN_CLICKED(IDC_ADD, &CMFCApplication3Dlg::OnBnClickedAdd)
        ON_BN_CLICKED(IDC_SUB, &CMFCApplication3Dlg::OnBnClickedSub)
        ON_BN_CLICKED(IDC_MUL, &CMFCApplication3Dlg::OnBnClickedMul)
        ON_BN_CLICKED(IDC_DIV, &CMFCApplication3Dlg::OnBnClickedDiv)
        ON_EN_CHANGE(IDC_NUM2, &CMFCApplication3Dlg::OnEnChangeNum2)
        ON_BN_CLICKED(IDC_RESET, &CMFCApplication3Dlg::OnBnClickedReset)
    END_MESSAGE_MAP()

    // CMFCApplication3Dlg 消息处理程序
    BOOL CMFCApplication3Dlg::OnInitDialog()
    {
        CDialogEx::OnInitDialog();
        // 将“关于...”菜单项添加到系统菜单中。
        // IDM_ABOUTBOX 必须在系统命令范围内。
        ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
        ASSERT(IDM_ABOUTBOX < 0xF000);
        CMenu* pSysMenu = GetSystemMenu(FALSE);
        if (pSysMenu != nullptr)
        {
            BOOL bNameValid;
            CString strAboutMenu;
            bNameValid = strAboutMenu.LoadString(IDS_ABOUTBOX);
            ASSERT(bNameValid);
```

```
            if (!strAboutMenu.IsEmpty())
            {
                pSysMenu->AppendMenu(MF_SEPARATOR);
                pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
            }
        }
        // 设置此对话框的图标。  当应用程序主窗口不是对话框时，框架将自动
        //  执行此操作
        SetIcon(m_hIcon, TRUE);            // 设置大图标
        SetIcon(m_hIcon, FALSE);        // 设置小图标
        // TODO: 在此添加额外的初始化代码
        return TRUE;  // 除非将焦点设置到控件，否则返回 TRUE
    }
    void CMFCApplication3Dlg::OnSysCommand(UINT nID, LPARAM lParam)
    {
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
        {
            CDialogEx::OnSysCommand(nID, lParam);
        }
    }
    // 如果向对话框添加最小化按钮，则需要下面的代码
    //  来绘制该图标。  对于使用文档/视图模型的 MFC 应用程序，
    //  这将由框架自动完成。
    void CMFCApplication3Dlg::OnPaint()
    {
        if (IsIconic())
        {
            CPaintDC dc(this); // 用于绘制的设备上下文
            SendMessage(WM_ICONERASEBKGND,
reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

            // 使图标在工作区矩形中居中
            int cxIcon = GetSystemMetrics(SM_CXICON);
            int cyIcon = GetSystemMetrics(SM_CYICON);
            CRect rect;
            GetClientRect(&rect);
            int x = (rect.Width() - cxIcon + 1) / 2;
            int y = (rect.Height() - cyIcon + 1) / 2;
```

```
        // 绘制图标
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        CDialogEx::OnPaint();
    }
}

//当用户拖动最小化窗口时系统调用此函数取得光标
//显示。
HCURSOR CMFCApplication3Dlg::OnQueryDragIcon()
{
    return static_cast<HCURSOR>(m_hIcon);
}

void CMFCApplication3Dlg::OnBnClickedButton1()
{
    // TODO: 在此添加控件通知处理程序代码
}

void CMFCApplication3Dlg::OnBnClickedRadio1()
{
    // TODO: 在此添加控件通知处理程序代码
}

void CMFCApplication3Dlg::OnEnChangeEdit1()
{
}
```

关键逻辑代码

```
void CMFCApplication3Dlg::OnEnChangeNum1()
{
    UpdateData(TRUE);
    switch (m_operator)
    {
    case 0:
        m_result = m_num1 + m_num2;             break;
    case 1:
        m_result = m_num1 - m_num2;         break;
    case 2:
        m_result = m_num1 * m_num2;         break;
    case 3:
```

```
            m_result = m_num1 / m_num2;
        }
        UpdateData(FALSE);
}


void CMFCApplication3Dlg::OnBnClickedAdd()
{
        // TODO: 在此添加控件通知处理程序代码
        OnEnChangeNum1();
}

void CMFCApplication3Dlg::OnBnClickedSub()
{
        // TODO: 在此添加控件通知处理程序代码
        OnEnChangeNum1();
}

void CMFCApplication3Dlg::OnBnClickedMul()
{
        // TODO: 在此添加控件通知处理程序代码
        OnEnChangeNum1();
}

void CMFCApplication3Dlg::OnBnClickedDiv()
{
        OnEnChangeNum1();
}

void CMFCApplication3Dlg::OnEnChangeNum2()
{
        // TODO:  在此添加控件通知处理程序代码
        OnEnChangeNum1();
}
void CMFCApplication3Dlg::OnBnClickedReset()
{
        m_result = m_num1 = m_num2 = m_operator = 0;
        UpdateData(FALSE);
}
```

实验十 动态链接库

一、实验目的

添加动态链接库，利用三方库函数

二、实验环境

Visual Studio2017、Windows10

三、实验内容

创建窗口利用匹配函数实现匹配。

四、实验步骤（描述详细过程）

1）创建工程

2）添加下图中资源视图

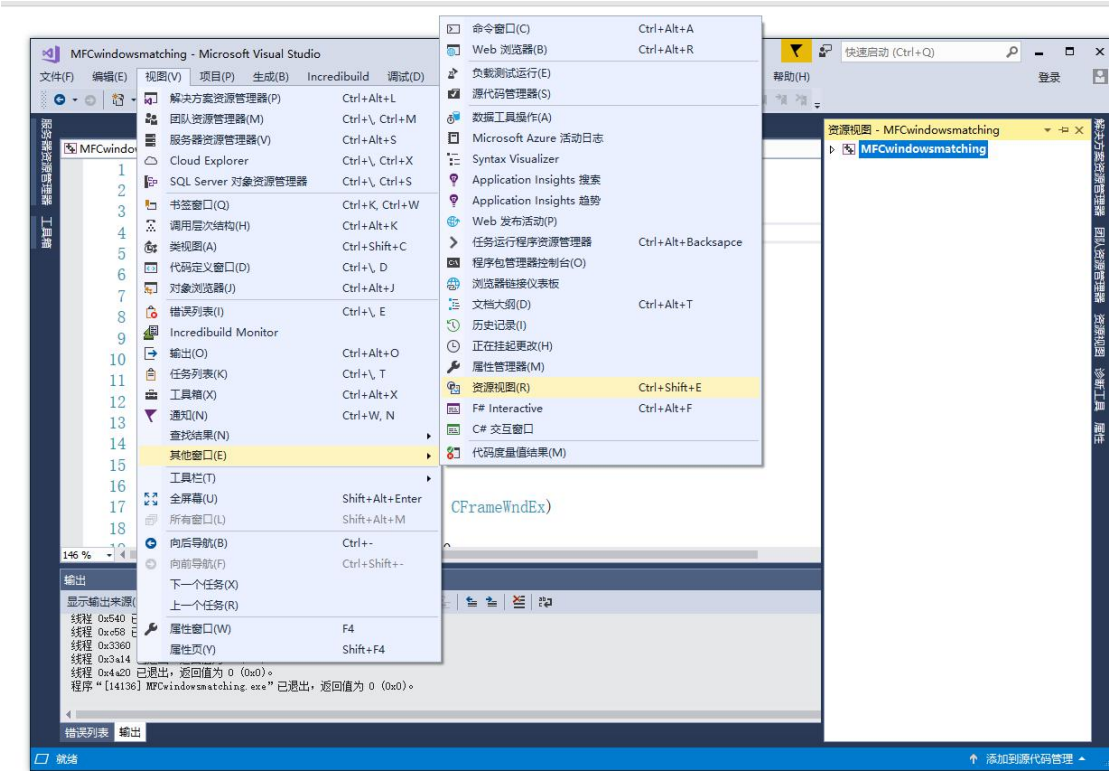


图 10-1. 资源视图中的菜单

3）添加对话框完成数据导入

图 10-2. 数据导入对话框

4）实现结果

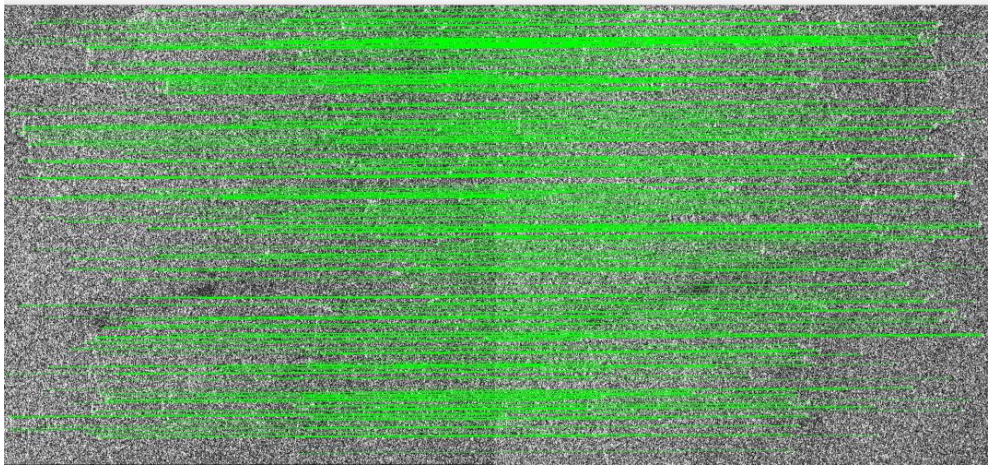


图 10-3. 匹配结果

代码如下：

```
#include "pch.h"
#include "framework.h"
#include "afxwinappex.h"
#include "afxdialogex.h"
#include "MFCApplication6.h"
#include "MainFrm.h"

#include "MFCApplication6Doc.h"
#include "MFCApplication6View.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#endif
// CMFCApplication6App
```

```
BEGIN_MESSAGE_MAP(CMFCApplication6App, CWinAppEx)
    ON_COMMAND(ID_APP_ABOUT, &CMFCApplication6App::OnAppAbout)
    // 基于文件的标准文档命令
    ON_COMMAND(ID_FILE_NEW, &CWinAppEx::OnFileNew)
    ON_COMMAND(ID_FILE_OPEN, &CWinAppEx::OnFileOpen)
    // 标准打印设置命令
    ON_COMMAND(ID_FILE_PRINT_SETUP, &CWinAppEx::OnFilePrintSetup)
END_MESSAGE_MAP()
// CMFCApplication6App 构造
CMFCApplication6App::CMFCApplication6App() noexcept
{
    m_bHiColorIcons = TRUE;
    // 支持重新启动管理器
    m_dwRestartManagerSupportFlags =
AFX_RESTART_MANAGER_SUPPORT_ALL_ASPECTS;
#ifdef _MANAGED
    // 如果应用程序是利用公共语言运行时支持(/clr)构建的，则:
    //      1) 必须有此附加设置，“重新启动管理器”支持才能正常工作。
    //      2) 在您的项目中，您必须按照生成顺序向 System.Windows.Forms 添加引
用。
    System::Windows::Forms::Application::SetUnhandledExceptionMode(System::Windo
ws::Forms::UnhandledExceptionMode::ThrowException);
#endif
    // TODO: 将以下应用程序 ID 字符串替换为唯一的 ID 字符串；建议的字符串格
式
    //为 CompanyName.ProductName.SubProduct.VersionInformation
    SetAppID(_T("MFCApplication6.AppID.NoVersion"));

    // TODO: 在此处添加构造代码，
    // 将所有重要的初始化放置在 InitInstance 中
}

// 唯一的 CMFCApplication6App 对象

CMFCApplication6App theApp;


// CMFCApplication6App 初始化

BOOL CMFCApplication6App::InitInstance()
{
    // 如果一个运行在 Windows XP 上的应用程序清单指定要
    // 使用 ComCtl32.dll 版本 6 或更高版本来启用可视化方式，
    //则需要 InitCommonControlsEx()。  否则，将无法创建窗口。
```

```
INITCOMMONCONTROLSEX InitCtrls;
InitCtrls.dwSize = sizeof(InitCtrls);
// 将它设置为包括所有要在应用程序中使用的
// 公共控件类。
InitCtrls.dwICC = ICC_WIN95_CLASSES;
InitCommonControlsEx(&InitCtrls);

CWinAppEx::InitInstance();


// 初始化 OLE 库
if (!AfxOleInit())
{
	AfxMessageBox(IDP_OLE_INIT_FAILED);
	return FALSE;
}
AfxEnableControlContainer();
EnableTaskbarInteraction(FALSE);

// 使用 RichEdit 控件需要 AfxInitRichEdit2()
// AfxInitRichEdit2();

// 标准初始化
// 如果未使用这些功能并希望减小
// 最终可执行文件的大小，则应移除下列
// 不需要的特定初始化例程
// 更改用于存储设置的注册表项
// TODO: 应适当修改该字符串，
// 例如修改为公司或组织名
SetRegistryKey(_T("应用程序向导生成的本地应用程序"));
LoadStdProfileSettings(4);  // 加载标准 INI 文件选项(包括 MRU)
InitContextMenuManager();
InitKeyboardManager();
InitTooltipManager();
CMFCToolTipInfo ttParams;
ttParams.m_bVislManagerTheme = TRUE;
theApp.GetTooltipManager()->SetTooltipParams(AFX_TOOLTIP_TYPE_ALL,
	RUNTIME_CLASS(CMFCToolTipCtrl), &ttParams);
// 注册应用程序的文档模板。  文档模板
// 将用作文档、框架窗口和视图之间的连接
CSingleDocTemplate* pDocTemplate;
pDocTemplate = new CSingleDocTemplate(
	IDR_MAINFRAME,
	RUNTIME_CLASS(CMFCApplication6Doc),
```

```
        RUNTIME_CLASS(CMainFrame),          // 主 SDI 框架窗口
        RUNTIME_CLASS(CMFCApplication6View));
    if (!pDocTemplate)
        return FALSE;
    AddDocTemplate(pDocTemplate);
    // 分析标准 shell 命令、DDE、打开文件操作的命令行
    CCommandLineInfo cmdInfo;
    ParseCommandLine(cmdInfo);
    // 调度在命令行中指定的命令。  如果
    // 用 /RegServer、/Register、/Unregserver 或 /Unregister 启动应用程序，则返回
FALSE。
    if (!ProcessShellCommand(cmdInfo))
        return FALSE;
    // 唯一的一个窗口已初始化，因此显示它并对其进行更新
    m_pMainWnd->ShowWindow(SW_SHOW);
    m_pMainWnd->UpdateWindow();
    return TRUE;
}
int CMFCApplication6App::ExitInstance()
{
    //TODO: 处理可能已添加的附加资源
    AfxOleTerm(FALSE);
    return CWinAppEx::ExitInstance();
}
// CMFCApplication6App 消息处理程序
// 用于应用程序“关于”菜单项的 CAboutDlg 对话框
class CAboutDlg : public CDialogEx
{
public:
    CAboutDlg() noexcept;

// 对话框数据
#ifdef AFX_DESIGN_TIME
    enum { IDD = IDD_ABOUTBOX };
#endif
protected:
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV 支持
// 实现
protected:
    DECLARE_MESSAGE_MAP()
};
CAboutDlg::CAboutDlg() noexcept : CDialogEx(IDD_ABOUTBOX)
{
}
```

```
void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
}
BEGIN_MESSAGE_MAP(CAboutDlg, CDialogEx)
END_MESSAGE_MAP()

// 用于运行对话框的应用程序命令
void CMFCApplication6App::OnAppAbout()
{
	CAboutDlg aboutDlg;
	aboutDlg.DoModal();
}
// CMFCApplication6App 自定义加载/保存方法
void CMFCApplication6App::PreLoadState()
{
	BOOL bNameValid;
	CString strName;
	bNameValid = strName.LoadString(IDS_EDIT_MENU);
	ASSERT(bNameValid);
	GetContextMenuManager()->AddMenu(strName, IDR_POPUP_EDIT);
	bNameValid = strName.LoadString(IDS_EXPLORER);
	ASSERT(bNameValid);
	GetContextMenuManager()->AddMenu(strName, IDR_POPUP_EXPLORER);
}
void CMFCApplication6App::LoadCustomState()
{
}
void CMFCApplication6App::SaveCustomState()
{
}
// CMFCApplication6App 消息处理程序
// Cloaddialog.cpp: 实现文件
//
#include "pch.h"
#include "MFCApplication6.h"
#include "Cloaddialog.h"
#include "afxdialogex.h"
// Cloaddialog 对话框
IMPLEMENT_DYNAMIC(Cloaddialog, CDialogEx)
Cloaddialog::Cloaddialog(CWnd* pParent /*=nullptr*/)
	: CDialogEx(IDD_DIALOG1, pParent)
{
}
```

```
Cloaddialog::~Cloaddialog()
{
}
void Cloaddialog::DoDataExchange(CDataExchange* pDX)
{
	CDialogEx::DoDataExchange(pDX);
	DDX_Control(pDX, IDC_EDIT1, my_searchEdit);
}
BEGIN_MESSAGE_MAP(Cloaddialog, CDialogEx)
	ON_BN_CLICKED(IDC_BUTTON1, &Cloaddialog::OnClickedButton1)
END_MESSAGE_MAP()
// Cloaddialog 消息处理程序
void Cloaddialog::OnClickedButton1()
{
	// TODO: 在此添加控件通知处理程序代码
	char searchPath[MAX_PATH];
	char *pass = NULL;
	CString searchname;
	ZeroMemory(searchPath, sizeof(searchPath));
	BROWSEINFO search;
	search.hwndOwner = NULL;
	search.pidlRoot = NULL;
	search.pszDisplayName = LPUWSTR(searchPath);
	search.lpszTitle = _T("请选择需要搜索的目录:");
	search.ulFlags = 0;
	search.lpfn = NULL;
	search.lParam = 0;
	search.iImage = 0;
	LPITEMIDLIST lp = SHBrowseForFolder(&search);
	if (lp && SHGetPathFromIDList(lp, LPUWSTR(searchPath)))
	{
		searchname = LPWSTR(searchPath);
		USES_CONVERSION;
		pass = W2A(searchname.GetBuffer(0));
		searchname.ReleaseBuffer();
	}
	my_searchEdit.SetWindowTextW(searchname);
}
```

实验十一 多线程编程技术

一、实验目的

利用多个线程同时工作，初步了解多线程。

二、实验环境

Visual Studio 2017、Windows10

三、实验内容

1、创建线程

2、使用 join()函数

3、使用 detach()函数

4、使用 joinable()函数

四、实验步骤（描述详细过程）

1）创建 win32 控制台项目如下图所示。

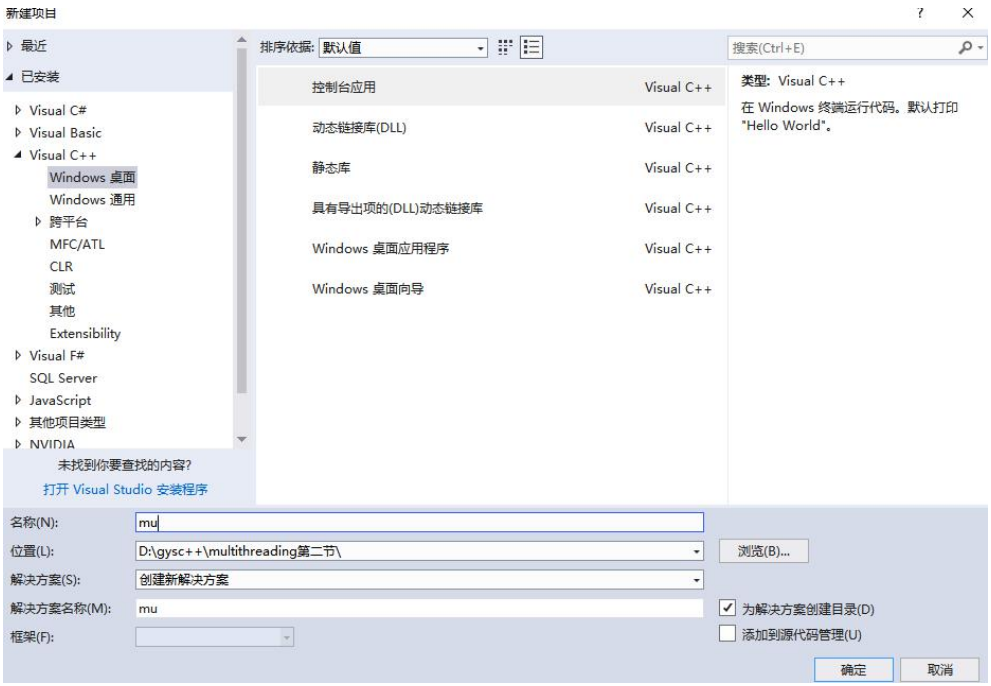


图 11-1.控制台

2）添加多线程的头文件#include<thread>；

3）更改命名空间 using namespace std;

4）定义一个线程函数如图 2 所示；

```
void myprint()
{
    cout << "我的线程开始" << endl;
    cout << "我的线程结束了" << endl;
}
```

5）将 myprint 函数作为变量传入线程中。

```
thread mytobj(myprint);//创建了线程，线程执行起点（入口）myprint（myprint）线程开始执行
```

6）使用 join()函数

创建线程以后，需要和主线程汇合，主线程必须等待子线程运行结束，join 函数会让主线程等待子线程运行结束。

7）使用 detach（）函数

创建线程以后，如果不需要主线程等待子线程运行结束，那么就可以使用 detach（）函数，在使用 detach()函数以后，主线程将不会等待子线程是否结束。

8）使用 joinable()

该函数是确定是否使用 join 函数。

9）部分参考代码

```
#include <iostream>
#include<string>
#include<map>
#include<thread>
using namespace std;
void myprint()
{
    cout << "我的线程开始" << endl;
    cout << "我的线程结束了" << endl;
}
int main()
{
    //一：范例演示线程运行的开始和结束
    //程序运行起来，生成一个进程，该进程所属的主线程开始自动运行

    //主线程从main（）开始执行，那么我们自己创建的线程，也需要从一个函数开始运行（初始函数），一旦这个函数运行完毕，就代表着我们这个线程结束）
    //整个进程是否执行完毕标志是主线程是否执行完毕，如果主线程执行完毕了，就代表这整个进程执行完毕了；
```

```
        //此时，如果其他子线程还没有执行完毕，那么这些子线程也会被操作系统强行终止
        //所以，一般情况下，我们得到一个结论：如果想保持子线程的运行状态，那么需要保持主线程一直运行
        //这条规律有例外
        //a）包含头文件  thread
        //b)
        //大家必须明确一点：有两个线程再跑，整个程序有两条线同时走，所以即使一条线被堵住了，另外一条线也可以继续走
        //1.1thread:标准库的类
        //1.2 jion（）：加入，阻塞主线程等待子线程执行忘，让主线程等待子线程执行完毕，然后子线程和主线程汇合。
        //如果主线程执行完毕了，但子线程没有执行完成，写出来的程序不稳定
        //一个良好的程序吗，应该是主线程等待子线程执行完毕
        //（1.3）detach（）:传统多线程主线程要等待子线程执行，然后自己再退出
        //detach：分离，主线程不和子线程汇合，主线程和子线程各自不影响
        //为什么引入detach：创建很多子线程，让主线程卓哥等待子线程结束，这种编程方法不太好，所以引入了detach（）；
        //建议等待
        //一旦detach（）,与主线程关联的thread对象就会失去与这个主线程的关联关系，此时这个子线程就会驻留再后台运行，这个子线程就被C++实行时刻接管，、
        //1.4jionable（）判断线程是否成功使用jion，或者detach，jion为True ，detach是False
        mytobj.join();//主线程阻塞到这里，等待子线程执行完毕，当子线程执行完毕，这个jion就执行完毕，主线程继续往后走
        mytobj.detach();    //这个子线程执行完毕，由运行时库负责清理该线程相关的资源（守护线程）
        //detach失去控制，一旦调用detach（）无法使用jion（）,否则系统异常
        cout << "SHOU" << endl;//实际上是主线程在执行，主线程从main函数返回，则整个进程执行完毕
        return 0;
    }
```

实验十二 CPU 并行计算

一、实验目的

利用多个线程同时工作，初步了解多线程。

二、实验环境

Visual Studio 2017、Windows10

三、实验内容

1）vector 容器创建多个线程

2）调用类的成员函数

3）互斥量保护共享资源

4）多个互斥量产生死锁

5）解决多个互斥量引起的死锁

四、实验步骤（描述详细过程）

1）创建一个 win32 控制台项目

2）添加#include<vector>头文件

3）在主函数中添加下如下代码，用来调用 A 类中的输入成员函数和输出成员函数，join 函数用来主线程等待子线程结束。

```
int main()
{

    cout << "Hello World!\n";//整个进程结束


    A myobja;
    thread myOutnnMsgObj(&A::outMsgRecvQueue, &myobja);//第二个参数时引用，才能保证线程里用的是同一个对象
    thread myInMsgObj(&A::inMsgRevQueue, &myobja);
    myOutnnMsgObj.join();
    myInMsgObj.join();
    //保护共享数据，用代码把共享数据锁住，其他想操作共享数据的线程必须等待解锁
```

```
        //其他想操作数据的线程必须等待数据解锁
        //”互斥量“
        //一：互斥量（mutex）的基本概念
        //互斥量是个类对象，多个线程尝试用lock()成员函数来枷锁这把锁头，只有一个线程能锁定成功（锁定成功的标志：lock()函数返回）
        //如果没锁成功，那么流程就卡在lock（）；
        //互斥量的只保护需要保护的数据
        //互斥量使用的方法#include<mutex>
        //lock（） ，unlock（）
        //先lock 后unlock()
        //使用规则要成对使用有lock必然调用unlock
        //有lock，忘记unlock问题很难排查
        //为了防止大家忘记unlock（）引入了一个std：：lock_guard；自动unlock
        //类似智能指针
        //2.2std：：lock_guard类模板，直接取代lock unlock
        //三：死锁

        //c++中
        //两个互斥量会产生死锁，两个锁才会死锁
        //两个线程A,B
        //线程A执行先锁资源A把lock成功，然后去锁资源B，正要去锁B时
        //线程B执行了先锁资源B，因为资源B没有被锁，资源被lock成功，线程B去lock资源A
        //现在死锁产生
        //线程A锁不住资源B，线程B锁不了资源A
        //死锁演示
        //死锁解决方案
        //只要保持两个互斥量保持锁的顺序一致
        //lock函数：一次可以锁住多个互斥量
        //lock不存在在多个线程中，因为锁的顺序问题导致死锁的风险问题
        //lock：可以锁住多个互斥量
        //3.4 lcok_guard<mutex>name (mutex,adopt_lock)
        //adopt_lock结构体变量，起一个标记作用，不需要在std::lock
        //总结lock一次锁住多个互斥量变量，谨慎使用，一个一个锁
    }
```

4、创建类的成员函数添加如下代码

```
class A {
public:
        //把收到的消息送到队列线程中
        void inMsgRevQueue()//unloack()
        {
            for (int i = 0; i < 1000000; ++i)
            {
                cout << "inMsgRevQueue()执行，插入一个元素" << i << endl;
```

```
                lock_guard<mutex>mguard2(my_mutex2);
                lock_guard<mutex>mguard(my_mutex);
                //my_mutex.lock();//实际工作中可能执行了很多其他代码，锁资源，这两个资源不一定挨着
                //my_mutex2.lock();
                msgRecvQueue.push_back(i);//假设这个数字就是命令，直接放到消息队列中
                //my_mutex2.unlock();
                //my_mutex.unlock();

            }
            return;

        }
        bool outMsgLULProc(int &command)
        {
            lock_guard<mutex>mguard(my_mutex);
            lock_guard<mutex>mguard2(my_mutex2);

                                                    //lock_guard执行了lock函数
                                                    //lock_guard析构函数执行了unlock函数
            //my_mutex2.lock();
            //my_mutex.lock();
            if (!msgRecvQueue.empty())
            {
                //消息队列不为空
                command = msgRecvQueue.front();//返回第一个元素，但是不检查第一个元素是否存在；
                msgRecvQueue.pop_front();//移除第一个元素，但不返回
                //my_mutex.unlock();
                //my_mutex2.unlock();
                //考虑匹配函数
                return true;
            }
            //my_mutex.unlock();
            //my_mutex2.unlock();
            return false;
        }
        //数据从队列中取出的线程
        void outMsgRecvQueue()
        {
            int command = 0;
```

```
            for (int i = 0; i < 1000000; ++i)
            {
                if (!msgRecvQueue.empty())
                {
                    bool result = outMsgLULProc(command);
                    if (result == true) {
                        cout << "outMsgRecvQueue()函数执行，取出一个元素" <<
command << endl;
                    }
                    //消息队列不为空
                    //command = msgRecvQueue.front();//返回第一个元素，但是不检查第
一个元素是否存在；
                    //msgRecvQueue.pop_front();//移除第一个元素，但不返回
                    //考虑匹配函数
                }
                else
                {
                    //消息队列为空
                    cout << "outMsgRecvQueue()函数执行，但消息队列为空" << i <<
endl;
                }
            }
        }
    private:
        list<int>msgRecvQueue;
        mutex my_mutex;//创建了互斥量
        mutex my_mutex2;//创建了互斥量
    };
```

在私有变量中，创建了两个互斥量，如果资源发生死锁那就资源锁住的顺序的不一样，如果两把锁都按照顺序去锁资源，那么就不会产生死锁的问题。

实验十三 矩阵并行计算

一、 实验目的

1、用 OpenMP 实现最基本的数值算法“矩阵乘法”

2、掌握 for 编译制导语句

3、对并行程序进行简单的性能调优

二、实验环境

Visual Studio2017

Windows10

三、实验内容

四、实验步骤

1）创建 win32 控制台项目

2）通过输入的矩阵的行列数、线程数和矩阵乘法运行次数来控制计算量，如下图所示：

图 13-1. 矩阵设置

3）观察不同线程数的，CPU 运行的时间如下图所示

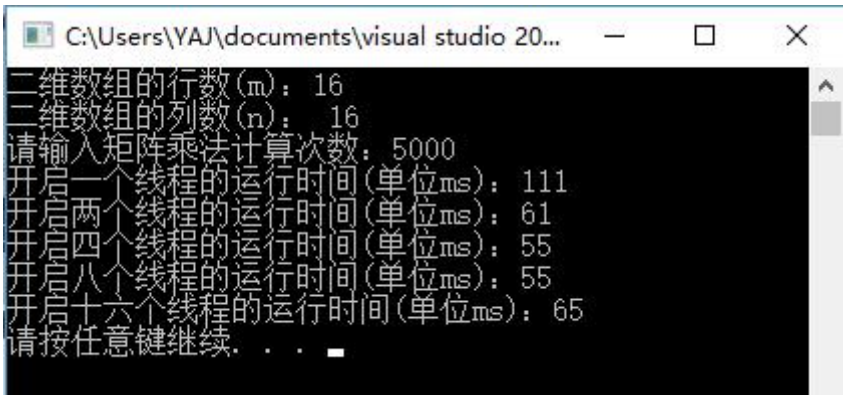


图 13-2. 线程比较结果

4）保持矩阵规模不变，martix_order=16（小规模矩阵）

观察在不同计算量（compute_scale=10,100,1000,5000,10000），加速比随线程数变化而变化（thread_num=1,2,4,8,16）的情况。

5）具体代码如下：

```
#include "stdafx.h"
#include<iostream>
#include <omp.h>
using namespace std;
float *generate_martix(int m, int n);//声明矩阵生成函数
void show_martix(int *arr, int m, int n);//声明矩阵显示函数
void multi_martix(float *arr1, float *arr2, int m, int n);//声明矩阵乘法函数
int main(int argc, char *argv[]) {
	int row, column;//矩阵行、列
	cout << "二维数组的行数(m)： " << endl;
	cin >> row ;
	cout << "二维数组的列数(n)：  " << endl;;
	cin >> column;
	float *martix1, *martix2;
	martix1 = generate_martix(row, column);//创建矩阵
	martix2 = generate_martix(column, row);//创建矩阵
	//show_martix(martix1,row,column);
	//printf("\n");
	//show_martix(martix2,column,row)
	int compute_scale;//矩阵乘法计算次数
	cout << ("请输入矩阵乘法计算次数： ") << endl;;
	cin >> compute_scale;
	//omp_set_num_threads(8);
	int start = 0;
	int end = 0;
	//开启一个线程
	omp_set_num_threads(1);
	start = omp_get_wtime() * 1000;
	int count=0;
	//printf("运行多线程前时间(单位ms)：%d\n",start);
#pragma omp parallel for
	for (int i = 0; i < compute_scale; i++) //根据for循环次数确定运算规模
	{
		multi_martix(martix1, martix2, row, column);//进行矩阵乘法
		//count++;
	}
	//printf("矩阵计算次数=%d\n",count);
	end = omp_get_wtime() * 1000;
```

```
    //printf("运行多线程后时间(单位ms)：%d\n",end);
    cout << "开启一个线程的运行时间(单位ms)" << end - start << endl;
    //开启两个线程
    omp_set_num_threads(2);
    start = omp_get_wtime() * 1000;
#pragma omp parallel for
    for (int i = 0; i < compute_scale; i++) //根据for循环次数确定运算规模
    {
        multi_martix(martix1, martix2, row, column);//进行矩阵乘法
    }
    end = omp_get_wtime() * 1000;
    cout << "开启两个线程的运行时间(单位ms)" << end - start << endl;
    //开启四个线程
    //omp_set_num_threads(4);
    start = omp_get_wtime() * 1000;
#pragma omp parallel for
    for (int i = 0; i < compute_scale; i++) //根据for循环次数确定运算规模
    {
        multi_martix(martix1, martix2, row, column);//进行矩阵乘法
    }
    end = omp_get_wtime() * 1000;
    cout << "开启四个线程的运行时间(单位ms)" << end - start << endl;
    //开启八个线程
    //omp_set_num_threads(8);
    start = omp_get_wtime() * 1000;
#pragma omp parallel for
    for (int i = 0; i < compute_scale; i++) //根据for循环次数确定运算规模
    {
        multi_martix(martix1, martix2, row, column);//进行矩阵乘法
    }
    end = omp_get_wtime() * 1000;
    cout << "开启八个线程的运行时间(单位ms)" << end - start << endl;
    //开启十六个线程
    //omp_set_num_threads(16);
    start = omp_get_wtime() * 1000;
#pragma omp parallel for
    for (int i = 0; i < compute_scale; i++) //根据for循环次数确定运算规模
    {
        multi_martix(martix1, martix2, row, column);//进行矩阵乘法
    }
    end = omp_get_wtime() * 1000;
    cout << "开启十六个线程的运行时间(单位ms)" << end - start << endl;
    system("pause");
    return 0;
```

```
}
//矩阵生成函数
float *generate_martix(int m, int n) {
	float *tmp;
	tmp = (float *)malloc(sizeof(float)*m*n);
	for (int i = 0; i < m; i++) {
		for (int j = 0; j < n; j++) {
			tmp[i*n + j] = i * (j + 1) + 1;//按此规则生成二维数组的各项
		}
	}
	return tmp;
}
//矩阵乘法函数
void multi_martix(float *arr1, float *arr2, int m, int n) {
	double temp = 0;
	float c[256][256];//最大运算256*256规模的矩阵

	for (int i = 0; i < m; ++i) {
		for (int j = 0; j < m; ++j) {
			for (int k = 0; k < n; k++) {
				temp += arr1[i*n + k] +    arr2[k*m + j];
			}
			//printf("temp=%d\n",temp);
			c[i][j] = temp;
			//printf("%5d", c[i][j]);
			temp = 0;//清理temp值
		}
		//printf("\n");
	}
}
//矩阵显示函数
void show_martix(int *arr, int m, int n) {
	for (int i = 0; i < m; i++) {
		for (int j = 0; j < n; j++)
		{
			cout << arr[i*n + j]<<"          " ;//按此规则输出二维数组的各项
		   // cout << "%5d" << arr[i*n + j];
		}
		cout << endl;
	}
	}
```

实验十四 GPU 并行计算

一、实验目的

了解简单的 GPU 并行计算

二、实验环境

Visual Studio2017

Windows10

三、实验内容

1）了解 GPU 环境配置

2）查看电脑 CUDA 的配置

3）在 Visual Studio2017 写一个.cu 函数查看电脑的配置

四、实验步骤（描述详细过程）

1）配置 GPU 环境，在 cmd 控制台中输入命令 nvcc -v 敲回车键如下图所示。

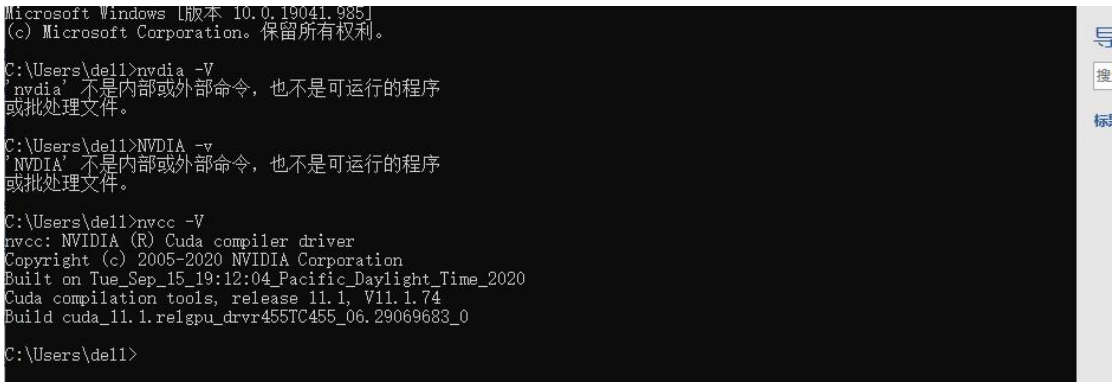


图 14-1. GPU 并行 CUDA 版本

上图中会显示 cuda 版本，则代表电脑有此环境。

2）创建一个 win32 项目起一个名字为 VS.CU

3）然后打开工程属性，并勾选相应的 CUDA 版本，如下图所示。

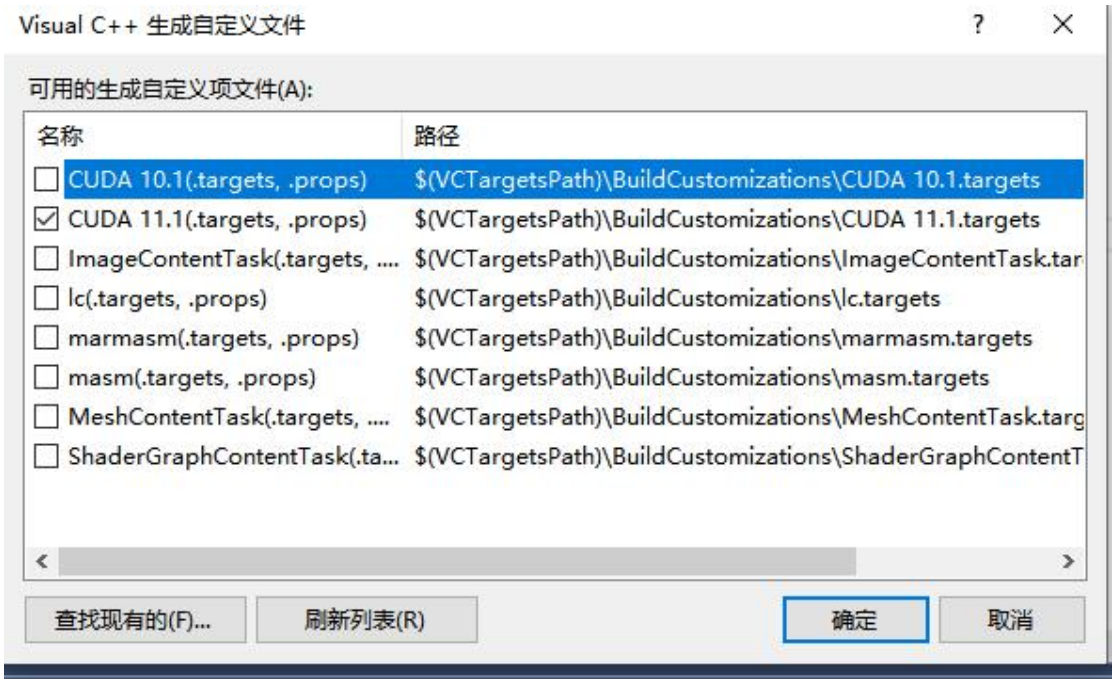


图 14-2. CUDA 与 VS 配置

4）然后在.cu 文件中添加如下代码

```
#include <omp.h>
#include <stdio.h>          // stdio functions are used since C++ streams aren't necessarily thread safe
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include<stdlib.h>
// a simple kernel that simply increments each array element by b
__global__ void kernelAddConstant(int *g_a, const int b)
{
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    g_a[idx] += b;
}

// a predicate that checks whether each array elemen is set to its index plus b
int correctResult(int *data, const int n, const int b)
{
    for (int i = 0; i < n; i++)
        if (data[i] != i + b)
            return 0;
    return 1;
}

int main(int argc, char *argv[])
{
```

```
int num_gpus = 0;           // number of CUDA GPUs

////////////////////////////////////////////////////////////////
// determine the number of CUDA capable GPUs
//
cudaGetDeviceCount(&num_gpus);
if (num_gpus < 1)
{
    printf("no CUDA capable devices were detected\n");
    return 1;
}

////////////////////////////////////////////////////////////////
// display CPU and GPU configuration
//
printf("number of host CPUs:\t%d\n", omp_get_num_procs());
printf("number of CUDA devices:\t%d\n", num_gpus);
for (int i = 0; i < num_gpus; i++)
{
    cudaDeviceProp dprop;
    cudaGetDeviceProperties(&dprop, i);
    printf("    %d: %s\n", i, dprop.name);
}
printf("-------------------------\n");

////////////////////////////////////////////////////////////////
// initialize data
    //
unsigned int n = num_gpus * 8192;
unsigned int nbytes = n * sizeof(int);
int *a = 0;                 // pointer to data on the CPU
int b = 3;                  // value by which the array is incremented
a = (int*)malloc(nbytes);
if (0 == a)
{
    printf("couldn't allocate CPU memory\n");
    return 1;
}
for (unsigned int i = 0; i < n; i++)
    a[i] = i;


////////////////////////////////////////////////////////////////
    // run as many CPU threads as there are CUDA devices
```

```
            //      each CPU thread controls a different device, processing its
            //      portion of the data.    It's possible to use more CPU threads
            //      than there are CUDA devices, in which case several CPU
            //      threads will be allocating resources and launching kernels
            //      on the same device.    For example, try omp_set_num_threads(2*num_gpus);
            //      Recall that all variables declared inside an "omp parallel" scope are
            //      local to each CPU thread
            //
        omp_set_num_threads(num_gpus);    // create as many CPU threads as there are CUDA devices
    //omp_set_num_threads(2*num_gpus);// create twice as many CPU threads as there are CUDA devices
    #pragma omp parallel
        {
            unsigned int cpu_thread_id = omp_get_thread_num();
            unsigned int num_cpu_threads = omp_get_num_threads();

            // set and check the CUDA device for this CPU thread
            int gpu_id = -1;
            cudaSetDevice(cpu_thread_id % num_gpus);            // "% num_gpus" allows more CPU threads than GPU devices
            cudaGetDevice(&gpu_id);

            printf("CPU thread %d (of %d) uses CUDA device %d\n", cpu_thread_id, num_cpu_threads, gpu_id);

            int *d_a = 0;     // pointer to memory on the device associated with this CPU thread
            int *sub_a = a + cpu_thread_id * n / num_cpu_threads;     // pointer to this CPU thread's portion of data
            unsigned int nbytes_per_kernel = nbytes / num_cpu_threads;
            dim3 gpu_threads(128);    // 128 threads per block
            dim3 gpu_blocks(n / (gpu_threads.x * num_cpu_threads));

            cudaMalloc((void**)&d_a, nbytes_per_kernel);
            cudaMemset(d_a, 0, nbytes_per_kernel);
            cudaMemcpy(d_a, sub_a, nbytes_per_kernel, cudaMemcpyHostToDevice);
            kernelAddConstant << <gpu_blocks, gpu_threads >> > (d_a, b);

            cudaMemcpy(sub_a, d_a, nbytes_per_kernel, cudaMemcpyDeviceToHost);
            cudaFree(d_a);


        }
        printf("---------------------------\n");
```

```
    if (cudaSuccess != cudaGetLastError())
        printf("%s\n", cudaGetErrorString(cudaGetLastError()));

    ////////////////////////////////////////////////////////////
    // check the result
    //
    if (correctResult(a, n, b))
        printf("Test PASSED\n");
    else
        printf("Test FAILED\n");

    free(a);       // free CPU memory

    cudaThreadExit();

    return 0;
}
```

实验十五 GPU 并行计算 2

一、实验目的

学习结合.cpp 和.cu 文件一起使用

二、实验环境

Visual Studio2017、Windows10

三、实验内容

1）创建.cpp 文件

2）创建点.cu 文件

3）在.cpp 文件中调用.cu 文件

四、实验步骤

1）首先创建一个 win32 控制台项目

2）然后在源文件中新建一个.cu 文件如下图所示，在 NVDIAD CUDA 中创建一个文件点击确定。

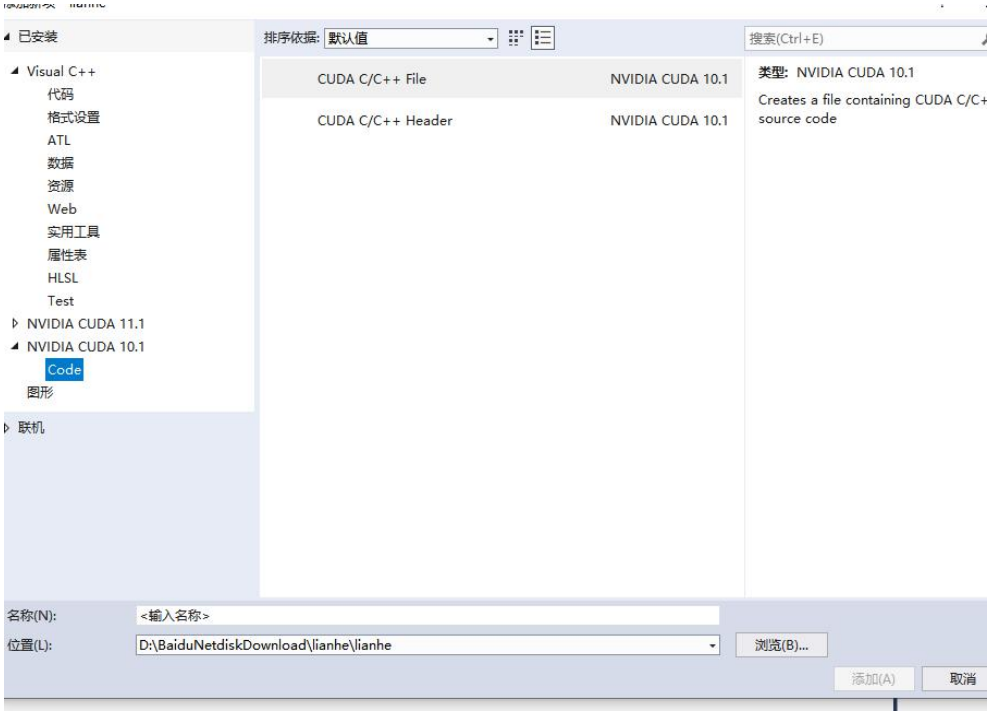


图 15-1. 创建 CUDA 文件

在.CPP 文件中创建如下代码：

```
#include <stdio.h>
#include "cuda_runtime.h"
#include "device_launch_parameters.h"

extern "C"
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size);
int main()
{
    const int arraySize = 5;
    const int a[arraySize] = { 1, 2, 3, 4, 5 };
    const int b[arraySize] = { 10, 20, 30, 40, 50 };
    int c[arraySize] = { 0 };
    // Add vectors in parallel.
    cudaError_t cudaStatus = addWithCuda(c, a, b, arraySize);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addWithCuda failed!");
        return 1;
    }
    printf("{1,2,3,4,5} + {10,20,30,40,50} = {%d,%d,%d,%d,%d}\n",
        c[0], c[1], c[2], c[3], c[4]);
    printf("cuda工程中调用cpp成功！ \n");

    // cudaDeviceReset must be called before exiting in order for profiling and
    // tracing tools such as Nsight and Visual Profiler to show complete traces.
    cudaStatus = cudaDeviceReset();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceReset failed!");
        return 1;
    }
    getchar(); //here we want the console to hold for a while
    return 0;
}
```

在.cu 文件中添加如下代码：

```
#include "cuda_runtime.h"
#include "device_launch_parameters.h"
#include <stdio.h>
__global__ void addKernel(int *c, const int *a, const int *b)
{
    int i = threadIdx.x;
    c[i] = a[i] + b[i];
}
// Helper function for using CUDA to add vectors in parallel.
extern "C"
```

```
cudaError_t addWithCuda(int *c, const int *a, const int *b, unsigned int size)
{
    int *dev_a = 0;
    int *dev_b = 0;
    int *dev_c = 0;
    cudaError_t cudaStatus;
    // Choose which GPU to run on, change this on a multi-GPU system.
    cudaStatus = cudaSetDevice(0);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaSetDevice failed!  Do you have a CUDA-capable GPU
installed?");
        goto Error;
    }
    // Allocate GPU buffers for three vectors (two input, one output)    .
    cudaStatus = cudaMalloc((void**)&dev_c, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_a, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    cudaStatus = cudaMalloc((void**)&dev_b, size * sizeof(int));
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMalloc failed!");
        goto Error;
    }

    // Copy input vectors from host memory to GPU buffers.
    cudaStatus = cudaMemcpy(dev_a, a, size * sizeof(int), cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

    cudaStatus = cudaMemcpy(dev_b, b, size * sizeof(int), cudaMemcpyHostToDevice);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }
```

```
    // Launch a kernel on the GPU with one thread for each element.
    addKernel << <1, size >> > (dev_c, dev_a, dev_b);

    // Check for any errors launching the kernel
    cudaStatus = cudaGetLastError();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "addKernel launch failed: %s\n", cudaGetErrorString(cudaStatus));
        goto Error;
    }

    // cudaDeviceSynchronize waits for the kernel to finish, and returns
    // any errors encountered during the launch.
    cudaStatus = cudaDeviceSynchronize();
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaDeviceSynchronize returned error code %d after launching
addKernel!\n", cudaStatus);
        goto Error;
    }

    // Copy output vector from GPU buffer to host memory.
    cudaStatus = cudaMemcpy(c, dev_c, size * sizeof(int), cudaMemcpyDeviceToHost);
    if (cudaStatus != cudaSuccess) {
        fprintf(stderr, "cudaMemcpy failed!");
        goto Error;
    }

Error:
    cudaFree(dev_c);
    cudaFree(dev_a);
    cudaFree(dev_b);
    return cudaStatus;
}
```