

控件是在系统内部定义的能够完成特定功能的控制单元。在应用程序中,使用控件不仅可以简化编程,还能完成常用的各种功能。在所有的控件中,根据它们的使用及 Visual C++ 对其支持的情况,可以把控件分为 Windows 一般控件(即早期的如编辑框、列表框、组合框和按钮等)、通用控件(如列表视图、树视图等控件)、MFC 扩展控件和 ActiveX 控件。

控件的创建方式有两种。一种方式是在对话框模板中指定控件(既可使用对话框编辑器进行创建,也可自行定义),这样,当应用程序启动该对话框时,Windows 系统就会为对话框创建控件;当对话框清除时,控件也随之清除。另一种方式是将控件看作子窗口,通过调用 `CWnd::Create()` 函数来创建控件。

在本实验(实训)中,将用对话框模板来设计几个比较有趣的实例: `Ex_Cal` 能完成简单的计算功能并能扩展,使其计算平方、立方、平方根和倒数; `Ex_Hatch` 用来例显控件中图案(图形)绘制及其颜色调整的能力; `Ex_Person` 用作管理学生的个人信息(含照片)。

实验目的

- 熟悉对话框应用程序的创建。
- 熟悉对话框资源的添加和设计。
- 掌握控件消息和对话框消息的映射。
- 学会在控件上绘制图案(图像)。
- 学会在同一个消息函数中处理不同控件的消息。
- 熟悉常见问题的处理方法和技巧。

实验内容

- 简单计算器与功能扩展。
- 控件图案(图形)绘制。
- 管理学生的个人信息。

实验准备和说明

- 具备知识: 常用控件(教程第 3 章)、使用 `CImage`(教程 8.1.2 节)。
- 构思并准备上机所需要的程序 `Ex_Cal`、`Ex_Hatch` 和 `Ex_Person`。
- 创建本实验(实训)的工作文件夹“D:\Visual C++ 程序\LiMing\3”。

3.1 简单计算器与功能扩展

一个简单的计算器常常包含加、减、乘和除四则运算,由于乘除的运算等级要高于加减,因此显示结果时需要对输入的表达式按运算等级进行解析。单击上方的“>>”按钮,计算器显示扩展功能:平方、立方、平方根和倒数,如图 3.1(a)所示;再次单击此按钮,则恢复到开始的简单功能,如图 3.1(b)所示。当然,表达式的输入是通过数字和符号按钮进行的,单击“=”按钮,显示计算结果。



图 3.1 带功能扩展的计算器

本实验(实训)创建的项目为 Ex_Cal,具体过程如下。

- (1) 设计计算器对话框。
- (2) 扩展功能按钮的显示与隐藏。
- (3) 映射并控制输入。
- (4) 解析并输出结果。
- (5) 扩展功能的实现。

3.1.1 设计计算器对话框

具体步骤如下。

- (1) 启动 Microsoft Visual Studio 2010。
- (2) 选择“文件”→“新建”→“项目”菜单命令或按快捷键 Ctrl+Shift+N 或单击顶层菜单下的标准工具栏中的  按钮,弹出“新建项目”对话框。在“已安装的模板”栏下选中 Visual C++ 下的 MFC 结点,在中间的模板栏中选中  MFC 应用程序。

- (3) 单击“位置”编辑框右侧的“浏览”按钮 ,从弹出的“项目位置”对话框中指定项目所在的文件夹 ,单击  按钮,回到“新建项目”对话框中。

- (4) 在“新建项目”对话框的“名称”编辑框中输入名称“Ex_Cal”。同时,要取消勾选“为解决方案创建目录”复选框。

(5) 单击  按钮, 出现“MFC 应用程序向导”欢迎页面, 单击  按钮, 出现“应用程序类型”页面。选中“基于对话框”应用程序类型, 此时右侧的“项目类型”自动选定为“MFC 标准”, 取消勾选“使用 Unicode 库”复选框。

(6) 保留默认选项, 单击  按钮, 系统开始创建, 并又回到 Visual C++ 主界面, 同时还自动打开对话框资源(模板)编辑器。将项目工作区切窗口换到“解决方案管理器”页面, 双击头文件结点  `stdafx.h`, 打开 `stdafx.h` 文档, 滚动到最后代码行, 将“`#ifdef _UNICODE`”和最后一行的“`#endif`”删除(注释掉)。

(7) 将文档窗口切换到对话框资源模板页面, 单击对话框编辑器上的“网格切换”按钮 , 显示模板网格。删除“TODO: 在此 xx”静态文本控件和“确定”“取消”按钮。

(8) 右击对话框资源模板, 从弹出的快捷菜单中选择“属性”命令, 出现其“属性”窗口。将 Caption(标题)属性改为“简单计算器与功能扩展”, Font(Size)属性设为“Tahoma, 常规, 9(小五)”。

(9) 调整对话框的大小(大小调为 $258 \times 197\text{px}$), 先添加一个静态文本控件, 其属性 ID 设为 `IDC_STATIC_DISP`, Right Align Text、Sunken、Center Image 属性指定为 True。参看图 3.1 的控件布局为对话框添加如表 3.1 所示的其他一些控件。

表 3.1 对话框添加的控件

添加的控件	ID	标 题	其 他 属 性
按钮	IDC_BUTTON_EX	<<	默认
按钮	IDC_BUTTON_X2=901	X^2	默认
按钮	IDC_BUTTON_X3=902	X^3	默认
按钮	IDC_BUTTON_SQRT=903	$\sqrt{\quad}$	默认
按钮	IDC_BUTTON_1X=904	$1/X$	默认
按钮	IDC_BUTTON_7=607	7	默认
按钮	IDC_BUTTON_8=608	8	默认
按钮	IDC_BUTTON_9=609	9	默认
按钮	IDC_BUTTON_4=604	4	默认
按钮	IDC_BUTTON_5=605	5	默认
按钮	IDC_BUTTON_6=606	6	默认
按钮	IDC_BUTTON_1=601	1	默认
按钮	IDC_BUTTON_2=602	2	默认
按钮	IDC_BUTTON_3=603	3	默认
按钮	IDC_BUTTON_0=600	0	默认
按钮	IDC_BUTTON_00=599	00	默认
按钮	IDC_BUTTON_DOT	.	默认
按钮	IDC_BUTTON_BACK	←	默认

续表

添加的控件	ID	标 题	其 他 属 性
按钮	IDC_BUTTON_CLEAR	C	默认
按钮	IDC_BUTTON_MUL=701	x	默认
按钮	IDC_BUTTON_DIV=702	÷	默认
按钮	IDC_BUTTON_ADD=703	+	默认
按钮	IDC_BUTTON_SUB=704	—	默认
按钮	IDC_BUTTON_RES	=	默认

说明：表中控件 ID 中的“=”用来重新为其指定一个值,输入时全部输入。例如,当输入“IDC_BUTTON_0=600”后表示将按钮 ID“IDC_BUTTON_0”的值设为 600。

(10) 选择“编译”→“资源符号”菜单命令,弹出如图 3.2 所示的“资源符号”对话框,从中核对其值是否与表 3.1 所设一致。若 ID 值不对,则关闭“资源符号”对话框后,在该控件的“属性”窗口,重新在 ID 属性的标识符后用“=”赋值。

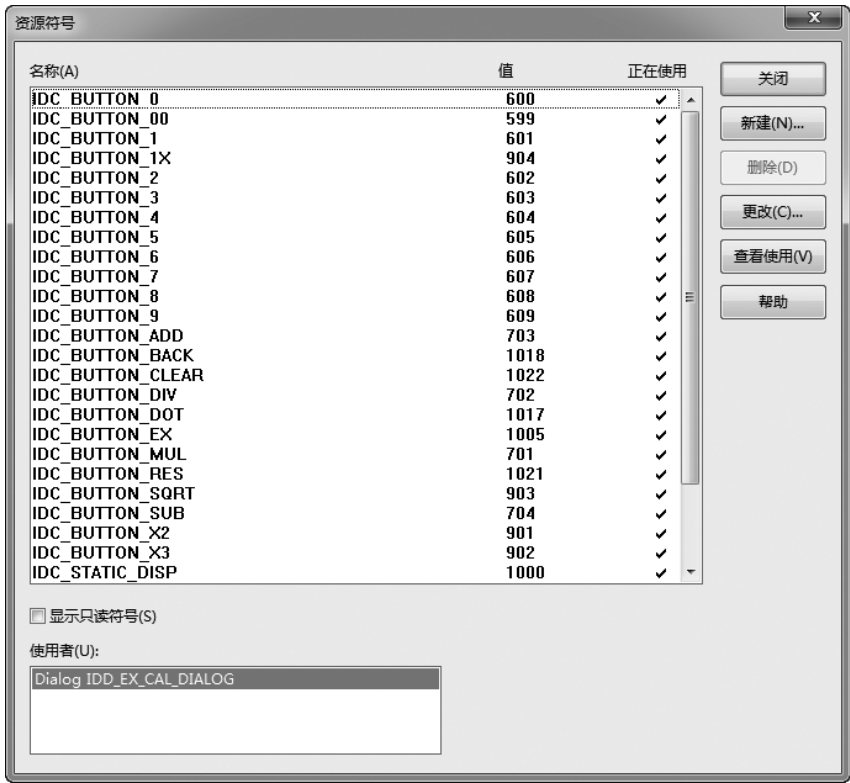


图 3.2 “资源符号”对话框

(11) 选择“格式”→“Tab 键顺序”菜单命令或按快捷键 Ctrl+D,此时每个控件的左上方都有一个数字,表明当前 Tab 键顺序,如图 3.3 所示。单击“=”按钮,则其 Tab 顺序值被重置为 1,这意味着该控件在对话框显示后第 1 个被选定的对象。按 Enter 键或在其他区域

单击鼠标,退出 Tab 键顺序设置状态。

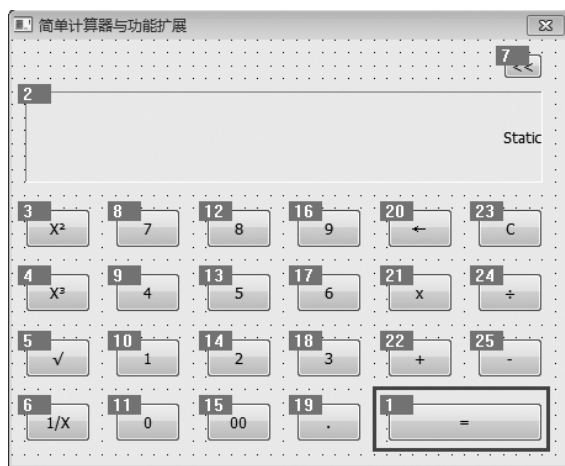


图 3.3 设置控件 Tab 键顺序

(12) 右击 IDC_STATIC_DISP 静态文本控件,从弹出的快捷菜单中选择“添加变量”命令,弹出“添加成员变量向导”对话框,指定“类别”为 Value,输入“变量名”为“m_strDisp”,如图 3.4 所示,单击 **完成** 按钮。

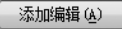


图 3.4 添加控件变量

3.1.2 扩展功能按钮的显示与隐藏

具体步骤如下。

(1) 右击“>>”按钮控件,从弹出的快捷菜单中选择“添加事件处理程序”命令,弹出“事

件处理程序向导”对话框,保留默认选项,单击  按钮,退出向导对话框。这样就为 CEx_CalDlg 类添加 IDC_BUTTON_EX 按钮控件的 BN_CLICKED 消息的默认映射函数,添加下列代码。

```
void CEx_CalDlg::OnBnClickedButtonEx()
{
    CRect    rcAllDlg;
    CWnd*    pExWnd        = GetDlgItem(IDC_BUTTON_EX);
    CString  strEx;
    pExWnd->GetWindowText(strEx);
    GetWindowRect(rcAllDlg);

    int      nOffset        = 0;

    //根据按钮的标题文本来判断
    if(strEx.Find("<<") >= 0)
    {    //收缩
        //设置扩展按钮标题
        pExWnd->SetWindowText(">>");
        nOffset        = -m_nExDist;
    }
    else
    {    //扩展
        //设置扩展按钮标题
        pExWnd->SetWindowText("<<");
        nOffset        = m_nExDist;
    }

    //对话框变化
    rcAllDlg.right    += nOffset;
    SetWindowPos(NULL, 0, 0, rcAllDlg.Width(), rcAllDlg.Height(),
        SWP_NOMOVE|SWP_NOZORDER);

    //所有控件移动 nOffset
    CWnd*    pWndCtrl    = GetWindow(GW_CHILD);
    CRect    rcChild;
    while(pWndCtrl != NULL)
    {
        pWndCtrl->GetWindowRect(rcChild);
        rcChild.OffsetRect(nOffset, 0);
        ScreenToClient(rcChild);
        pWndCtrl->MoveWindow(rcChild);
        pWndCtrl = pWndCtrl->GetWindow(GW_HWNDNEXT);
    }
}
```

```

//最后处理显示控件 IDC_STATIC_DISP
CWnd*      pDispWnd    = GetDlgItem(IDC_STATIC_DISP);
pDispWnd->GetWindowRect(rcChild);
rcChild.left      -= nOffset;
ScreenToClient(rcChild);
pDispWnd->MoveWindow(rcChild);
pDispWnd->Invalidate();          //强制刷新
}

```

(2) 将项目工作区窗口切换到“类视图”页面,双击 CEx_CalDlg 类名结点,打开 Ex_CalDlg.h,在类 CEx_CalDlg 中添加下列成员变量定义代码。

```

class CEx_CalDlg : public CDialogEx
{
public:
    int          m_nExDist;
    int          m_nCurKeyType;
    bool         m_bHasDot;
    bool         m_bHasNum;
//构造
public:
    CEx_CalDlg(CWnd* pParent = NULL);    //标准构造函数

```

(3) 在“类视图”CEx_CalDlg 类的成员页面中,双击 OnInitDialog 结点,打开并在该函数的最后(return true;语句前)添加下列初始化代码。

```

BOOL CEx_CalDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();
    ...
    //TODO: 在此添加额外的初始化代码
    CWnd*      pD7Wnd      = GetDlgItem(IDC_BUTTON_7);
    CWnd*      pD8Wnd      = GetDlgItem(IDC_BUTTON_8);

    CRect      rcD7, rcD8;
    pD7Wnd->GetWindowRect(rcD7);
    pD8Wnd->GetWindowRect(rcD8);

    m_nExDist   = rcD8.left - rcD7.left;
    if(m_nExDist < 0) m_nExDist   = -m_nExDist;

    m_strDisp   = "0";          //开始显示为 0
    m_bHasDot   = false;

```

```

m_bHasNum        = false;
m_nCurKeyType    = -1;           //没有输入

//设置显示字体
CWnd *    pWnd    = GetDlgItem(IDC_STATIC_DISP);
CFont *    pFont   = pWnd->GetFont();
LOGFONT lf;
pFont->GetLogFont(&lf);
lf.lfHeight      = -24;           //指定为 24px 高度
static CFont    font;             //须将其指定为全局, 否则后面的字体设置无效
font.CreateFontIndirect(&lf);     //创建字体
pWnd->SetFont(&font);
UpdateData(false);
return TRUE;                      //除非将焦点设置到控件, 否则返回 TRUE
}

```

(4) 编译运行并测试, 结果如图 3.5 所示。



图 3.5 扩展功能按钮的显示与隐藏

3.1.3 映射并控制输入

为了区分简单计算器中 0、00、1~9 数字按钮与小数点、加、减、乘、除等按钮, 这里用前面已定义的成员变量 `m_nCurKeyType` (当前按钮类型, -1 表示当前没有输入, 1 表示当前输入数字, 2 表示当前输入小数点, 5 表示当前输入运算符)、`m_bHasDot` (是否已有小数点)、`m_bHasNum` (是否已有非 0 数值) 来控制输入。具体步骤如下。

(1) 在“类视图”页面中, 右击 `CEx_CalDlg` 类结点, 从弹出的快捷菜单中选择“属性”命令, 弹出其“属性”窗口, 在窗口上部单击“重写”按钮 , 将其切换到“重写”页面, 找到 `OnCommand`, 在其右侧栏单击鼠标, 然后单击右侧的下拉按钮 , 从弹出的下拉项中选择“添加 `OnCommand`”, 这样 `OnCommand()` 虚函数重写 (重载) 函数就添加完成。

(2) 此时自动转向文档窗口, 并定位到 `CEx_CalDlg::OnCommand()` 函数实现的源代码处。关闭“属性”窗口, 添加下列代码。


```
BOOL CEx_CalDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    //TODO: 在此添加专用代码和/或调用基类
    WORD nID = LOWORD(wParam);           //获取控件 ID
    if((nID >= 599) && (nID <= 609))     //数字按钮
    {
        //处理 0、00 按钮
        if((nID == 600) || (nID == 599))
        {
            if(m_bHasDot || m_bHasNum)    //前有小数点或非 0 数值
            {
                m_strDisp = m_strDisp + "0";
                if(nID == 599) m_strDisp = m_strDisp + "0";
                UpdateData(false);
                m_nCurKeyType = 1;
            }
        }
        else
        {
            if(m_nCurKeyType < 0) m_strDisp.Empty();
            m_bHasNum = true;
            CString strNum;
            strNum.Format("%d", nID - 600);
            m_strDisp = m_strDisp + strNum;
            UpdateData(false);
            m_nCurKeyType = 1;
        }
    }
    else if(nID == IDC_BUTTON_DOT)       //处理小数点按钮
    {
        if(!m_bHasDot)
        {
            m_bHasDot = true;
            m_strDisp = m_strDisp + ".";
            UpdateData(false);
            m_nCurKeyType = 2;
        }
    }
    else if((nID >= 701) && (nID <= 704)) //处理 * /+-
    {
        int nLength = m_strDisp.GetLength();
        if((m_nCurKeyType > 0) && (nLength > 0))
        {
```

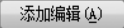
```

        m_strDisp.Replace("×", "*");
        m_strDisp.Replace("÷", "/");
        CString strOP[] = {"*", "/", "+", "-"};
        if(m_nCurKeyType >= 5)
        {
            nLength = m_strDisp.GetLength();
            m_strDisp = m_strDisp.Left(nLength-1);
        }
        m_strDisp = m_strDisp + strOP[nID - 701];
        m_strDisp.Replace("*", "×");
        m_strDisp.Replace("/", "÷");
        UpdateData(false);
        m_nCurKeyType = 5;
        m_bHasDot = false;
        m_bHasNum = false;
    }
}

return CDialogEx::OnCommand(wParam, lParam);
}

```

(3) 将项目工作区窗口切换到“资源视图”页面(若没有此页面,则选择“视图”→“资源视图”菜单命令显示),展开所有结点,双击 Dialog 下的 IDD_EX_CAL_DIALOG 结点,打开对话框资源模板。

(4) 右击 C 按钮控件(IDC_BUTTON_CLEAR),从弹出的快捷菜单中选择“添加事件处理程序”命令,弹出“事件处理程序向导”对话框,保留默认选项,单击  按钮,退出向导对话框。这样,就添加了该按钮控件的 BN_CLICKED“事件”消息的默认映射函数,添加下列代码。

```

void CEx_CalDlg::OnBnClickedButtonClear()
{
    //TODO: 在此添加控件通知处理程序代码
    m_strDisp = "0"; //开始显示为 0
    m_bHasDot = false;
    m_bHasNum = false;
    m_nCurKeyType = -1; //没有输入
    UpdateData(false);
}

```

(5) 类似地,为按钮 IDC_BUTTON_BACK 添加 BN_CLICKED“事件”消息的默认映射函数,并添加下列代码。

```

void CEx_CalDlg::OnBnClickedButtonBack()
{

```

```

//TODO: 在此添加控件通知处理程序代码
if(m_nCurKeyType < 0) return;
m_strDisp.Replace("×", "*");
m_strDisp.Replace("÷", "/");
int nLength      = m_strDisp.GetLength();
if(nLength <= 1)
{
    OnBnClickedButtonClear();    //则相当于单击 C 按钮
    return;
}
m_strDisp        = m_strDisp.Left(nLength - 1);
char chLast      = m_strDisp[nLength - 2];
if((chLast >= '0') && (chLast <= '9'))
{
    m_bHasNum      = true;
    m_nCurKeyType = 1;
}
else if(chLast == '.')
{
    m_bHasDot      = true;
    m_nCurKeyType = 2;
}
else
{
    //剩下来应是+-*/运算符
    m_nCurKeyType = 5;
    m_bHasDot      = false;
    m_bHasNum      = false;
}
m_strDisp.Replace("*", "×");
m_strDisp.Replace("/", "÷");
UpdateData(false);
}

```

(6) 编译运行并测试。

3.1.4 解析并输出结果

由于乘除的运算等级高于加减,因此解析输入的表达式时需要先按乘除进行运算。由于乘除是二元运算符,因此可先找出运算符左、右两边操作数,运算后的结果“替换”原来的表达式,然后循环解析直到没有乘除运算符为止,最后解析加减运算。同级别相邻运算符需按从左到右的顺序来计算结果。具体步骤如下。

(1) 将项目工作区窗口切换到“类视图”页面,右击 CEx_CalDlg 类结点,从弹出的快捷菜单中选择“添加”→“添加函数”命令,弹出“添加成员函数向导”对话框,将“返回类型”选为

CString, 在“函数名”框中输入“GetLeftData”;“参数类型”为 CString, 指定“参数名”为 str, 单击 **添加(A)** 按钮; 再次选择“参数类型”为 double&, 指定“参数名”为 fRes, 如图 3.6 所示, 单击 **添加(A)** 按钮。



图 3.6 添加成员函数

(2) 保留其他默认选项, 单击 **完成** 按钮。此时, 对话框退出, 并自动定位到添加的函数实现代码处, 在这里可以为该函数添加下列代码。

```
CString CEx_CalDlg::GetLeftData(CString str, double& fRes)
{
    int nIndex          = str.GetLength() - 1;
    //从最后往前查找
    while(nIndex>= 0)
    {
        char ch          = str[nIndex];
        if((ch == '+') || (ch == '-') || (ch == '*') || (ch == '/'))
            break;
        nIndex--;
    }
    CString strLeft;      //数据拿掉后还剩下的字符串
    if(nIndex >= 0)      {
        strLeft          = str.Left(nIndex + 1);
        fRes              = atof(str.Mid(nIndex + 1));
    }
}
```

```

        else
        {
            fRes          = atof(str);
            strLeft.Empty();
        }
        return strLeft;
    }
}

```

此函数的目的是将外部运算符左边子串 str 的最右边转换成浮点数并由引用参数 fRes 返回,而将剩下的字符串由函数返回。

(3) 类似地,为 CEx_CalDlg 类添加成员函数 GetRightData(),用来取外部运算符右边子串 str 的最左边转换成浮点数,结果保存到引用 fRes 中,剩下的字符串由函数返回。添加的函数代码如下。

```

CString CEx_CalDlg::GetRightData(CString str, double& fRes)
{
    int nIndex          = 0;
    int nLength          = str.GetLength();
    while(nIndex < nLength)
    {
        char ch          = str[nIndex];
        if((ch == '+' ) || (ch == '-' ) || (ch == '*' ) || (ch == '/'))
            break;
        nIndex++;
    }
    CString strRight;      //数据拿掉后还剩下的字符串
    if(nIndex >= 0)    {
        strRight        = str.Mid(nIndex);
        fRes              = atof(str.Left(nIndex));
    }
    else
    {
        fRes              = atof(str);
        strRight.Empty();
    }
    return strRight;
}

```

(4) 为 CEx_CalDlg 类添加用于处理运算符的成员函数 DoParseOP(),函数返回解析后的字符串。具体的函数代码如下。

```

CString CEx_CalDlg::DoParseOP(CString strData, CString strOP)
{

```

```

int nPos;
while((nPos = strData.FindOneOf(strOP)) >= 0)
{
    //处理首字符是运算符的情况
    if(nPos == 0)    {
        CString strTemp = strData.Mid(1);
        nPos = strTemp.FindOneOf(strOP);
        if(nPos < 0)    break;
        else            nPos++;
    }
    char    chOP        = strData[nPos];
    //以当前运算符为界,将字符串分成左、右两段
    CString strL        = strData.Left(nPos);
    CString strR        = strData.Mid(nPos+1);
    double fLData, fRData, fData;
    CString strLL        = GetLeftData(strL, fLData);
    CString strRR        = GetRightData(strR, fRData);
    if(chOP == '*' )        fData = fLData * fRData;
    else if(chOP == '/')
    {
        //处理除数为 0.0 的情况
        if((fabs(fRData)) > 0.0)
            fData = fLData/fRData;
        else
            return "E";
    }
    else if(chOP == '+')        fData = fLData + fRData;
    else if(chOP == '-')        fData = fLData - fRData;
    CString strRes;
    strRes.Format("%f", fData);
    strRes.TrimRight('0');
    strRes.TrimRight('.');
    //将左右剩下来和当前计算结果合成一个表达式
    strData.Format("%s%s%s", strLL, strRes, strRR);
}
return strData;
}

```

(5) 为“=”按钮 IDC_BUTTON_RES 添加 BN_CLICKED“事件”消息的默认映射函数,并添加下列代码。

```

void CEx_CalDlg::OnBnClickedButtonRes()
{

```

```

//TODO: 在此添加控件通知处理程序代码
//无输入或输入为空
if((m_nCurKeyType < 0) || (m_strDisp.IsEmpty()))
{
    OnBnClickedButtonClear();        return;
}
m_strDisp.Replace("×", "*");
m_strDisp.Replace("÷", "/");
//先处理"*/",再处理"+-"
m_strDisp    = DoParseOP(m_strDisp, "* /");
m_strDisp    = DoParseOP(m_strDisp, "+ -");
//处理结果
m_strDisp.TrimRight('0');
m_strDisp.TrimRight('.');

UpdateData(false);
m_bHasNum    = true;
if(m_strDisp.Find('.') >= 0)
    m_bHasDot = true;
else
    m_bHasDot = false;

m_nCurKeyType = 1;
}

```

(6) 在 Ex_CalDlg.cpp 文件的前面添加 cmath 头文件包含指令。

```

#include "afxdialogex.h"
#include <cmath>

```

(7) 编译运行并测试,结果如图 3.7 所示。



图 3.7 普通四则运算功能

3.1.5 扩展功能的实现

对于平方、立方、求平方根和倒数功能的实现代码,不必添加在相应按钮的映射函数中,而只须添加在 CEx_CalDlg::OnCommand()中即可。具体添加的代码如下。

```
BOOL CEx_CalDlg::OnCommand(WPARAM wParam, LPARAM lParam)
{
    //...
    else if ((nID >= 901) && (nID <= 904))    //扩展功能
    {
        if (!m_strDisp.IsEmpty())    {
            double fRes;
            bool bOK = true;
            if (m_strDisp.FindOneOf("×÷+-") >= 0)    {
                m_strDisp = "E";
                bOK = false;
            } else
                fRes = atof(m_strDisp);
            if (bOK)    {
                if (nID == 901) fRes = fRes * fRes;
                else if (nID == 902) fRes = fRes * fRes * fRes;
                else if (nID == 903)    {
                    if (fRes >= 0.0) fRes = sqrt(fRes);
                    else bOK = false;
                }
                else if (nID == 904)    {
                    if (fabs(fRes) > 0.0) fRes = 1.0/fRes;
                    else bOK = false;
                }
            }
        }
        if (bOK)    {
            //处理结果
            m_strDisp.Format("%f", fRes);
            m_strDisp.TrimRight('0');
            m_strDisp.TrimRight('.');
            m_bHasNum = true;
            if (m_strDisp.Find('.') >= 0)
                m_bHasDot = true;
            else
                m_bHasDot = false;
            m_nCurKeyType = 1;
        }
    }
    else
    {

```



```

        m_nCurKeyType    = -1;
        m_bHasDot         = false;
        m_bHasNum         = false;
    }
    UpdateData(false);
}
}
return CDialogEx::OnCommand(wParam, lParam);
}

```

3.2 控件图案绘制

在控件中绘制图案的关键在于控制对话框自动刷新控件。图案绘制后还可通过“图案”组合框和颜色分量调节器来改变绘制的图案类型和颜色,创建的项目为 Ex_Hatch,结果如图 3.8(a)所示。



图 3.8 Ex_MFC 运行结果

具体实验(实训)过程如下。

- (1) 设计图案绘制对话框。
- (2) WM_PAINT 和控件绘制。
- (3) 图案及其颜色调整。

3.2.1 设计图案绘制对话框

具体步骤如下。

- (1) 选择“文件”→“关闭解决方案”菜单命令,关闭原来的项目。
- (2) 选择“文件”→“新建”→“项目”菜单命令或按快捷键 Ctrl+Shift+N 或单击顶层菜单下的标准工具栏中的  按钮,弹出“新建项目”对话框。保留默认选项,直接在“名称”编辑框中输入名称“Ex_Hatch”。
- (3) 单击  按钮,出现“MFC 应用程序向导”欢迎页面,单击  按钮,出现“应

用程序类型”页面。选中“基于对话框”应用程序类型,此时右侧的“项目类型”自动选定为“MFC 标准”,取消勾选“使用 Unicode 库”复选框。

(4) 保留默认选项,单击  按钮,系统开始创建,并又回到了 Visual C++ 主界面,同时还自动打开对话框资源(模板)编辑器。将项目工作区切窗口换到“解决方案管理器”页面,打开 stdafx.h 文档,滚动到最后代码行,将“#ifdef _UNICODE”和最后一行的“#endif”删除(注释掉)。

(5) 将文档窗口切换到对话框资源模板页面,删除“TODO: 在此 xx”静态文本控件和“取消”按钮。将对话框 Caption(标题)属性改为“图案绘制”,将“确定”按钮 Caption(标题)属性改为“退出”。

(6) 单击对话框编辑器上的“网格切换”按钮,显示模板网格。调整对话框大小(调为 282×185px),并将“退出”移至对话框的右下角。参看图 3.8(b)控件布局为对话框添加如表 3.2 所示的一些控件并微调(其中将旋转按钮控件的 Alignment 属性选定为 Right Align,Auto buddy 和 Set buddy integer 设为 True)。

表 3.2 对话框添加的控件

添加的控件	ID	标题	其他属性
组合框	IDC_COMBO_HATCH	—	默认
静态文本	IDC_STATIC_DRAW	默认	Sunken
编辑框	IDC_EDIT_R	—	默认
旋转按钮控件	IDC_SPIN_R	—	见上面提示
水平滚动条	IDC_SCROLLBAR_G	—	默认
滑动条	IDC_SLIDER_B	—	默认

需要说明的是。

- ① 当添加组合框后,一定要单击右侧的下拉箭头,增大下拉框的大小。
- ② 当“红色分量”编辑框添加之后,紧接着要添加旋转按钮,这样它们才能组成伙伴窗口。

(7) 选择“项目”→“类向导”菜单或按快捷键 Ctrl+Shift+X,弹出“MFC 类向导”对话框。查看“类名”组合框中是否已选择了 CEx_HatchDlg,切换到“成员变量”页面。在“控件变量”列表中,选中所需的控件 ID,双击鼠标或单击  按钮。依次为表 3.3 控件增加成员变量。单击  按钮,退出“MFC 类向导”对话框。

表 3.3 控件变量

控件 ID	变量类别	变量类型	变量名	范围和大小
IDC_COMBO_HATCH	Control	CComboBox	m_cbHatch	—
IDC_EDIT_R	Value	UINT	m_nRValue	0~255
IDC_SPIN_R	Control	CSpinButtonCtrl	m_spinRValue	—
IDC_SCROLLBAR_G	Control	ScrollBar	m_sbGValue	—
IDC_SLIDER_B	Control	CSliderCtrl	m_sdBValue	—

(8) 编译。

3.2.2 WM_PAINT 和控件绘制

在 MFC 框架中,当需要更新或重新绘制窗口外观时,应用程序就会发送 WM_PAINT 消息,通常需要在对话框中映射 WM_PAINT 消息以便执行自己的绘制代码。不过,在基于对话框应用程序中,WM_PAINT 消息映射已自动添加。所以,控件绘制的代码只要添加到 OnPaint()映射函数就可以了。

只是,在对话框中的控件进行绘画时,为了防止 Windows 用系统默认的颜色向对话框进行重复绘制,须调用 UpdateWindow() (更新窗口) 函数来达到这一效果。UpdateWindow() 是 CWnd 的一个无参数的成员函数,其目的是绕过系统的消息列队,而直接发送或停止发送 WM_PAINT 消息。当窗口没有需要更新区域时,就停止发送。这样,当图形绘制完时,由于没有 WM_PAINT 消息的发送,系统也就不会用默认的颜色对窗口进行重复绘制。

当然,也可在程序中通过调用 Invalidate()函数来通知系统此时的窗口状态已变为无效,强制系统调用 WM_PAINT 消息函数 OnPaint()重新绘制。

具体步骤如下。

(1) 将项目工作区窗口切换到“类视图”页面,双击 `CEx_HatchDlg` 类名结点,打开 `Ex_HatchDlg.h`,在类 `CEx_HatchDlg` 中添加下列成员变量定义代码。

```
public:
    CComboBox          m_cbHatch;
    UINT               m_nRValue;
    UINT               m_nGValue;
    UINT               m_nBValue;
    UINT               m_nHatch;
    CScrollBar         m_sbGValue;
    CSliderCtrl        m_sdBValue;
    CSpinButtonCtrl    m_spinRValue;
```

(2) 右击 CEx_HatchDlg 类结点,从弹出的快捷菜单中选择“添加”→“添加函数”命令,弹出“添加成员函数向导”对话框,将“返回类型”选为 void,在“函数名”框中输入“DrawHatch”。保留其他默认选项,单击  按钮。此时,对话框退出,并自动定位到添加的函数实现代码处,在这里可以为该函数添加下列代码。

[illegible]

```

//创建一个图案画刷。RGB 是一个颜色宏,用来将指定的红、绿、蓝三种
//颜色分量转换成一个 32 位的 RGB 颜色值
CBrush* pOldBrush = pDC->SelectObject(&drawBrush);
CRect rcClient;
pWnd->GetClientRect(rcClient);           //获取当前控件的客户区大小
pDC->Rectangle(rcClient);                //用当前画刷填充指定的矩形框
pDC->SelectObject(pOldBrush);            //恢复原来的画刷
}

```

(3) 在 CEx_HatchDlg::OnInitDialog() 函数中添加下列初始化代码。

```

BOOL CEx_HatchDlg::OnInitDialog()
{
    CDialogEx::OnInitDialog();
    ...
    //TODO: 在此添加额外的初始化代码
    //填充并设定图案组合框选项
    int nIndex;
    nIndex = m_cbHatch.AddString("向上 45 度斜线");
    m_cbHatch.SetItemData(nIndex, HS_BDIAGONAL);
    nIndex = m_cbHatch.AddString("交叉十字线");
    m_cbHatch.SetItemData(nIndex, HS_CROSS);
    nIndex = m_cbHatch.AddString("斜十字交叉线");
    m_cbHatch.SetItemData(nIndex, HS_DIAGCROSS);
    nIndex = m_cbHatch.AddString("向下 45 度斜线");
    m_cbHatch.SetItemData(nIndex, HS_FDIAGONAL);
    nIndex = m_cbHatch.AddString("水平线");
    m_cbHatch.SetItemData(nIndex, HS_HORIZONTAL);
    nIndex = m_cbHatch.AddString("垂直线");
    m_cbHatch.SetItemData(nIndex, HS_VERTICAL);
    m_cbHatch.SetCurSel(0);
    m_nHatch = m_cbHatch.GetItemData(0);
    //设置上下转换按钮范围和当前值
    m_spinRValue.SetRange(0, 255);
    m_nRValue = 0;
    m_spinRValue.SetPos(m_nRValue);
    //设置滚动条范围和当前值
    m_sbGValue.SetScrollRange(0, 255);
    m_nGValue = 0;
    m_sbGValue.SetScrollPos(m_nGValue);
    //设置滑动条范围和当前值
    m_sdBValue.SetRange(0, 255);
    m_nBValue = 0;
}

```