

10 Lessons About

C++

**You Need To Learn
to Become a
Master Programmer**



Andrew Webb

10 Lessons About C++ You Need To Learn to Become a Master Programmer

Andrew Webb

A BERRY PUBLISHING HOUSE BOOK

First published in the UK
by Berry Publishing House
7 Kew Lodge
227 Kew Road
Richmond
Surrey TW9 3LQ

Copyright © Andrew Webb 2015 - Cover Getty Images (Rights Purchased)

All rights reserved. No part of this publication may be reproduced in any material form (including printing or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication) without the written permission of the copyright owner, except in accordance with the provisions of the Copyright, Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licencing Agency, 90 Tottenham Court Road, London W1P 9HE. Applications for the copyright owner's written permission to reproduce any part of this publication should be addressed to the publisher.

Your Free Gift

As a way of saying thank you for your purchase, I'm offering an exclusive free copy of all the coding scripts in full colour and indented that are included within this book. This enables readers to copy, paste and alter the coding to suit their own needs and also make the exact applications contained in the book fully functional without spending hours debugging the syntax, as I know how annoying and time consuming this is.

Click [here](#) to request your free downloadable copy of all the coding scripts contained in the book.

Chapter 1: Windows Programing

1.1 Getting Started: A simple window with event handling

The problem with teaching Windows programming — especially to those wishing to become master programmers — is that it is a vast subject, one that deserves a book or several books in its own right. In order to give an introduction I have been very selective, focusing on a few important basics with the intention that the reader can use these to build his or her own understanding.

The gist of Windows programming is this: the user initiates an event such as pressing a key or clicking a mouse, the Windows operating system recognizes and intercepts this event, preprocesses it, and dispatches it to the appropriate user program. When the user event is received by the program, it can decide on how it will act upon it, or otherwise leave it alone and let Windows handle it via its default processing.

There are a few things needed to get even the simplest of Windows applications up and running. Programmers who wish to create even a minimalist application will need to use the data types, classes, constants, functions and structures contained in the Microsoft Win32 base library. I have always preferred to try things out and learn about them in more depth later, so I will do the same here. Let's start with a bare bones Windows application with explanations to follow.

Every Windows program has an entry-point function that is named either `WinMain` or `wWinMain`:

```
int WINAPI wWinMain( HINSTANCE hInstance,  
HINSTANCE hPrevInstance,  
PWSTR pCmdLine,  
int nCmdShow);
```

1. **hInstance** - a handle to a module the Windows operating system uses to identify the executable (.exe) when loaded into memory. This handle instance is required for certain Windows operations such as loading icons and bitmaps.
2. **hPrevInstance** - is relevant to 16-bit Windows applications but is no longer used.

Now it is always set to zero.

3. **pCmdLine** - contains the command line arguments as a Unicode string.
4. **nCmdShow** - a flag indicating whether the main application window is to be minimized, maximized or displayed normally.

When creating a basic window programmatically the four minimum steps necessary are as follows:

1. Create the window class used to store information about things such as the type of Window being used, the Windows Procedure function used to control the window, icons, colours and other parameters. This is held in the **WNDCLASSEX** structure.

```
WNDCLASSEX windowClass;
```

```
windowClass.cbSize = sizeof( WNDCLASSEX );
```

```
windowClass.style = CS_HREDRAW | CS_VREDRAW;
```

```
windowClass.lpfnWndProc = WndProc;
```

```
windowClass.cbClsExtra = 0;
```

```
windowClass.cbWndExtra = 0;
```

```
windowClass.hInstance = hInstance;
```

```
windowClass.hIcon = LoadIcon( NULL, IDI_APPLICATION );
```

```
windowClass.hCursor = LoadCursor( NULL, IDC_ARROW );
```

```
windowClass.hbrBackground = (HBRUSH) GetStockObject( WHITE_BRUSH );
```

```
windowClass.lpszMenuName = NULL;
```

```
windowClass.lpszClassName = "MyClass";
```

```
windowClass.hIconSm = LoadIcon( NULL, IDI_WINLOGO );
```

2. Register the Windows class. Whenever a Windows program starts, it registers the windows class with the system, and gives the system a pointer to a callback function called the windows procedure. This windows procedure is called each time Windows wants to pass an event message back to the program. You need do this only once before using it as many times as necessary. The window attributes defined in step 1 can be changed at any time if desired.

Use *RegisterClassEx* to do the registering.

```
RegisterClassEx(&windowClass);
```

3. Create the window object using the **CreateWindowEx** function (See MSDN documentation for an explanation of **CreateWindowEx** and its parameters:

<https://msdn.microsoft.com/en-us/library/ms908193.aspx>)

```
CreateWindowEx(  
WS_EX_CLIENTEDGE,  
g_szClassName,  
“The title of my window”,  
WS_OVERLAPPEDWINDOW,  
CW_USEDEFAULT,  
CW_USEDEFAULT,  
240,  
120,  
NULL,  
NULL,  
hInstance,  
NULL);
```

See MSDN documentation for an explanation of **CreateWindowEx** and its parameters:

<https://msdn.microsoft.com/en-us/library/ms908193.aspx>

4. Enter the main message loop, which is where all event messages will get sent for processing. A bare minimum example is shown below:

```
LRESULT CALLBACK WndProc(HWND hwnd, UINT msg, WPARAM wParam,  
LPARAM lParam)  
{  
switch(msg)  
{  
case WM_CLOSE:
```



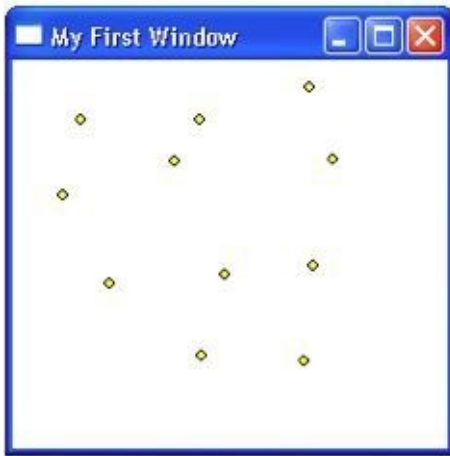
```
DestroyWindow(hwnd);  
break;  
case WM_DESTROY:  
PostQuitMessage(0);  
break;  
default:  
return DefWindowProc(hwnd, msg, wParam, lParam);  
}  
return 0;  
}
```

A mouse click, for example, gets dispatched to the program that created the window over which the mouse was positioned and clicked. There are literally dozens of possible messages that will get handled by Windows – every time you resize, minimize, maximize, press a key, and so forth.

Then the program responsible for that window will get a chance to intercept and process the message for that event in whatever way the programmer chooses. This example shows you not only how to create a basic window programmatically, but also add the capability to respond to mouse event notifications. Hardly earth-shattering, but it does help you to get the hang of handling messages. When the program is first run we are presented with the simple window:



The current mouse cursor position is tracked every time it moves, so that when the left mouse button has been clicked, a yellow circle is painted at the location of the mouse click:



The full code listing is as follows:

```
#include <windows.h>
```

```
#include <set>
```

```
#define WIN32_LEAN_AND_MEAN
```

```
const unsigned radius = 3;
```

```
int xPos = 0;
```

```
int yPos = 0;
```

```
typedef std::pair<int, int> coordinate;
```

```
typedef std::set<coordinate>::const_iterator coord_iter;
```

```
// Step 4: Process any message sent to the Window
```

```
LRESULT CALLBACK WndProc(HWND hwnd,
```

```
UINT message,
```

```
WPARAM wParam,
```

```
LPARAM lParam)
```

```
{
```

```
PAINTSTRUCT paintStruct;
```

```
HDC hDC;
```

```
HBRUSH hOldBrush, hNewBrush;
```

```
static std::set<coordinate> coords;
```

```
switch( message )
```

```

{
case WM_CREATE:
return 0;
break;

case WM_CLOSE:
PostQuitMessage(0);
return 0;
break;
case WM_PAINT:
// Paint the set of circles
hDC = BeginPaint( hwnd, &paintStruct );
hNewBrush = CreateSolidBrush( RGB(255, 255, 0) );
hOldBrush = (HBRUSH) SelectObject(hDC, hNewBrush);

for ( coord_iter it = coords.begin();
it != coords.end();
++it )
{
const int x = (*it).first;
const int y = (*it).second;
Ellipse( hDC, x - radius, y + radius, x + radius, y - radius );
}

SelectObject(hDC, hOldBrush);
DeleteObject(hNewBrush);
EndPaint(hwnd, &paintStruct);
return 0;
break;

case WM_MOUSEMOVE:
// Store the mouse cursor position at every movement

```

```
xPos = LOWORD(lParam);
```

```
yPos = HIWORD(lParam);
```

```
break;
```

```
case WM_LBUTTONDOWN:
```

```
// Store unique set of mouse coordinates and redraw
```

```
coords.insert( std::make_pair( xPos, yPos ) );
```

```
InvalidateRect(hwnd, NULL, TRUE);
```

```
break;
```

```
default:
```

```
break;
```

```
}
```

```
return DefWindowProc( hwnd, message, wParam, lParam );
```

```
}
```

```
int APIENTRY WinMain(HINSTANCE hInstance,
```

```
HINSTANCE hPrevInstance,
```

```
LPSTR lpCmdLine,
```

```
int nCmdShow)
```

```
{
```

```
WNDCLASSEX windowClass; // window class
```

```
HWND hwnd; // window handle
```

```
MSG msg; // message
```

```
// Step 1: Create the Windows class, exit if unsuccessful
```

```
windowClass.cbSize = sizeof( WNDCLASSEX );
```

```
windowClass.style = CS_HREDRAW | CS_VREDRAW;
```

```
windowClass.lpfnWndProc = WndProc;
```

```
windowClass.cbClsExtra = 0;
```

```
windowClass.cbWndExtra = 0;
```

```
windowClass.hInstance = hInstance;
```

```
windowClass.hIcon = LoadIcon( NULL, IDI_APPLICATION );
windowClass.hCursor = LoadCursor( NULL, IDC_ARROW );
windowClass.hbrBackground = (HBRUSH) GetStockObject( WHITE_BRUSH );
windowClass.lpszMenuName = NULL;
windowClass.lpszClassName = "MyClass";
windowClass.hIconSm = LoadIcon( NULL, IDI_WINLOGO );
// Step 2: Register the window, exit if unsuccessful
if ( !RegisterClassEx(&windowClass) ) return 0;
// Step 3: Create the window, exit if unsuccessful
if ( !CreateWindowEx(
    NULL, // extended style
    "MyClass", // class name
    "My First Window", // app name
    WS_OVERLAPPEDWINDOW | WS_VISIBLE | WS_SYSMENU, // window style
    50, // x coordinate
    50, // y coordinate
    250, // width
    250, // height
    NULL, // handle to parent
    NULL, // handle to menu
    hInstance, // application instance
    NULL ) ) return 0; // no extra parameter's
// Step 4: Enter the main message loop and event handling
while(GetMessage(&msg, NULL, 0, 0) > 0)
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
return msg.wParam;
}
```

1.2 Creating dialogs programmatically

Dialog boxes are Windows applications that can display messages and controls while allowing user interaction via events such as mouse clicks and key presses. Dialog boxes can be created as applications in their own right, or they can be used in addition to an existing application. A dialog box is created from a resource file, which is recognized as having the .rc extension. This file contains information about the dialog box and specifies parameters such as the dialog box size and location, its textual caption, and whether it contains controls like buttons, edit boxes, and so forth.

There are two main approaches to defining Microsoft Foundation Class (MFC) dialog boxes for User Interface-based applications. Either use the Resource Editor (a tool to define the dialog appearance and generate the code associated with the dialog) or create (and hence later modify) dialog boxes at run time by using using the Win32 API directly.

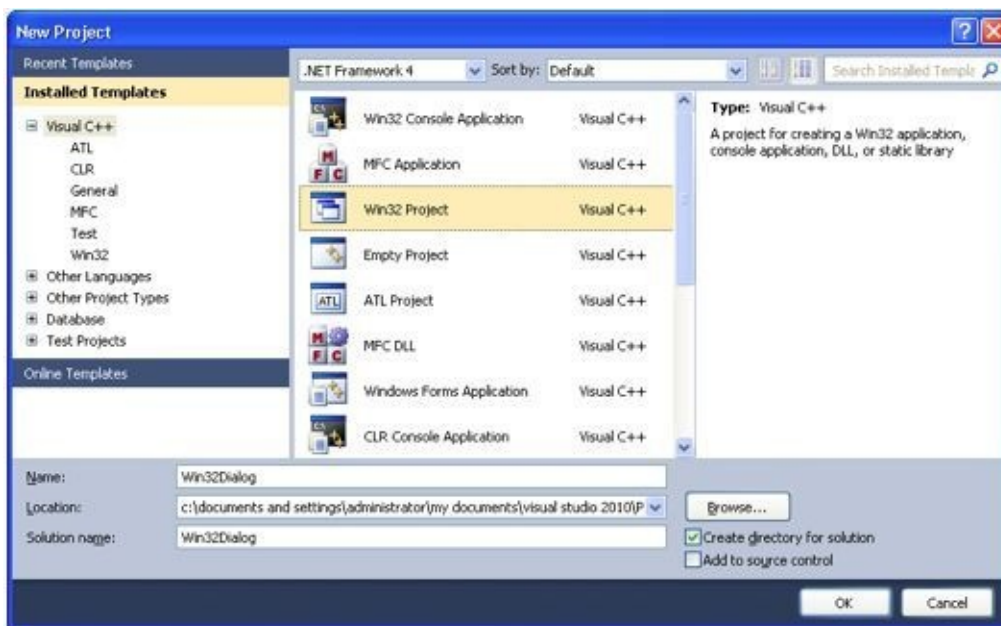
The Visual Studio Resource Editor can be used to position dialog components in conjunction with the Class Wizard to generate the corresponding C++ control code in separate .cpp and .h files. The Resource Editor is very convenient in that all dialog box layout information is stored in the resource (.rc) file, and all corresponding resource identifiers are stored in the resource.h file. To programmatically create a dialog box, use the **DialogBox** function. Its usage is:

```
INT_PTR DialogBox( HINSTANCE hInstance,  
LPCTSTR lpTemplate,  
HWND hWndParent,  
DLGPROC lpDialogFunc );
```

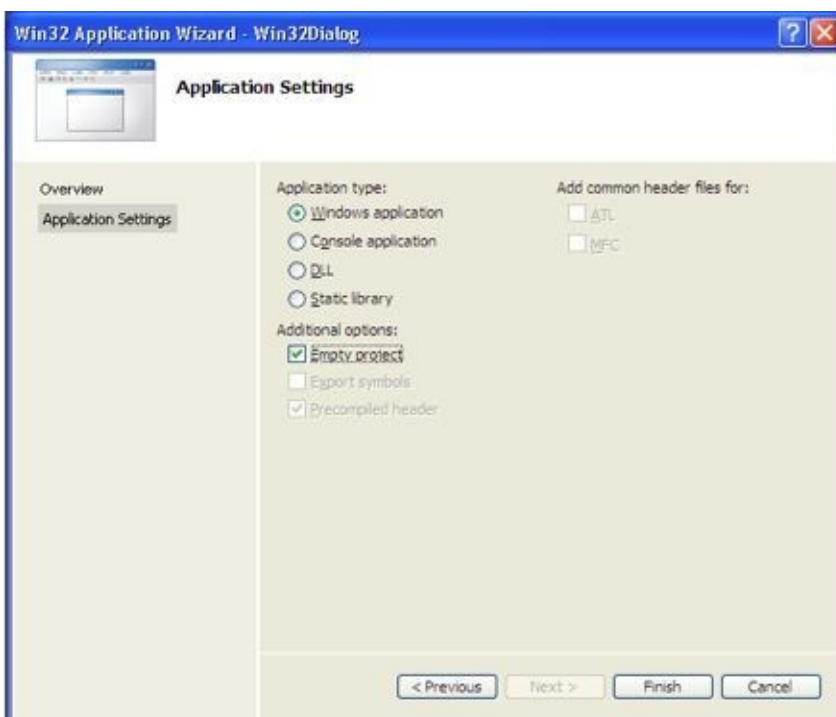
The first argument hInstance is the handle to the application using the dialog box. The lpTemplate argument specifies the dialog box template. The hWndParent argument is a handle to the parent window that owns the dialog box. The lpDialogFunc argument is a pointer to the dialog box procedure, implying the need to define a callback procedure whose usage is:

```
INT_PTR CALLBACK DialogProc( HWND hwndDlg,  
UINT uMsg,  
WPARAM wParam,  
LPARAM lParam );
```

To do this in Visual Studio first create a new Win32 application. Specify the desired location of your project and enter the project name, say Win32Dialog, and then click OK:



In the Application Wizard that appears make sure you create it as an Empty Project before clicking on Finish:

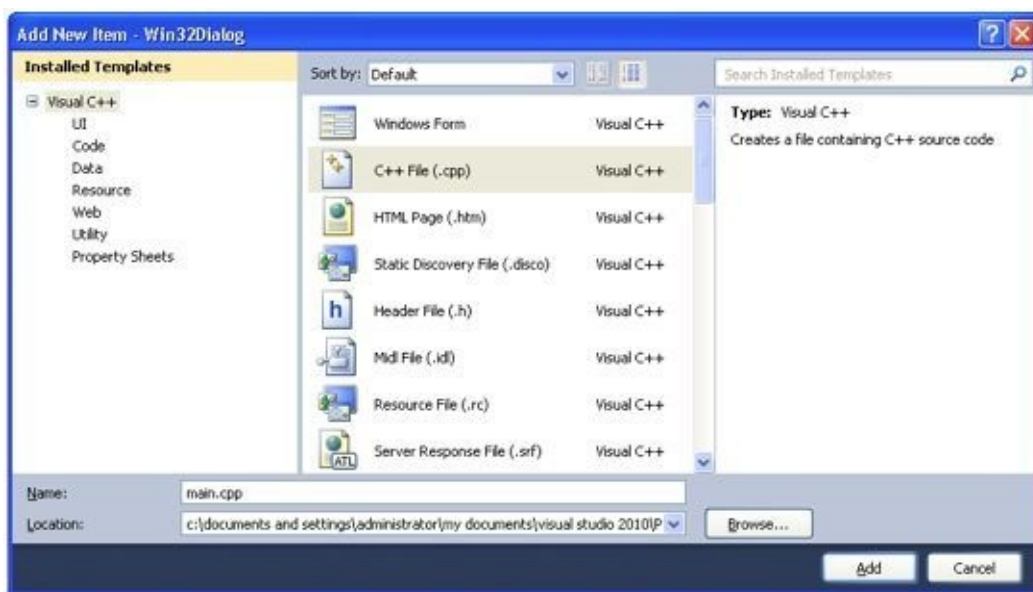


Now you need to actually create your dialog. Right-click the project folder that Visual Studio has created and select Add > Resource: Select Dialog and then select New:



This will generate the resource header and rc file used to define the dialog. In the Resource Editor, right-click on the newly created dialog box and select Properties. The name of the ID defaults to **IDD_DIALOG1**. Change this name if you wish.

Then create the main source file by selecting Add > New Item. In the Name edit box type the name of the source file, such as main.cpp, and press Add:



In the main.cpp file, insert the following code:

```
#include <windows.h>
```

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

```
HWND hWnd;
```

```
LRESULT CALLBACK DlgProc(HWND hWnd,
```

```
UINT Msg,
```



```
WPARAM wParam,  
LPARAM lParam);
```

```
INT WINAPI WinMain( HINSTANCE hInstance,  
HINSTANCE hPrevInstance,  
LPSTR lpCmdLine,  
int nCmdShow )  
{  
    DialogBox( hInstance,  
MAKEINTRESOURCE( IDD_DIALOG1 ),  
hWnd,  
reinterpret_cast<DLGPROC>( DlgProc ) );  
  
    return FALSE;  
}
```

```
LRESULT CALLBACK DlgProc(HWND hWndDlg,  
UINT Msg,  
WPARAM wParam,  
LPARAM lParam)  
{  
    switch(Msg)  
    {  
        case WM_INITDIALOG:  
            return TRUE;  
  
        case WM_COMMAND:  
            switch(wParam)  
            {  
                case IDOK:  
                    EndDialog(hWndDlg, 0);  
                    return TRUE;
```

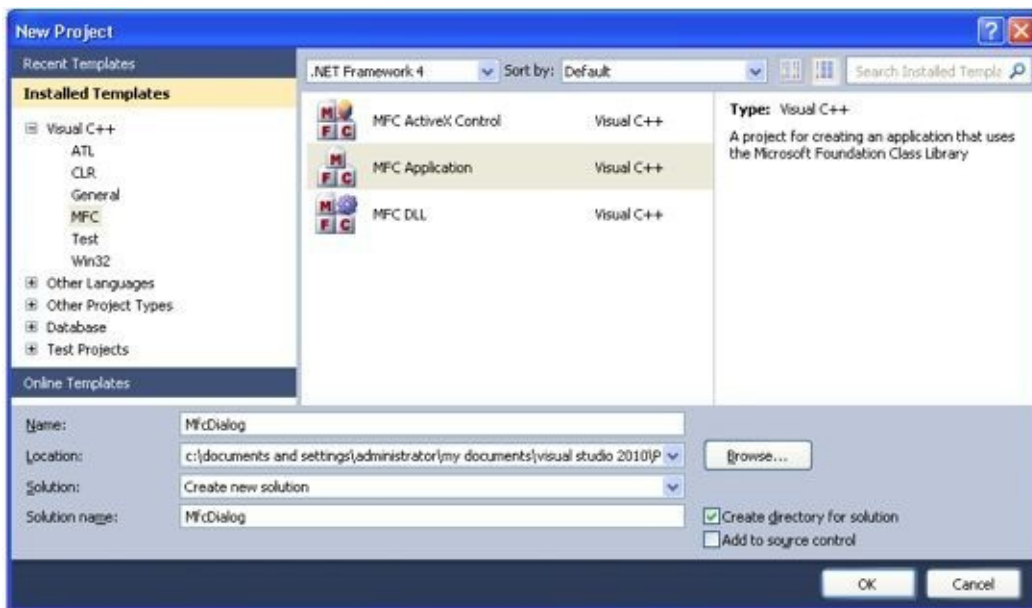
```
}  
break;  
  
return FALSE;  
}
```

This will produce the following dialog when you run the application:

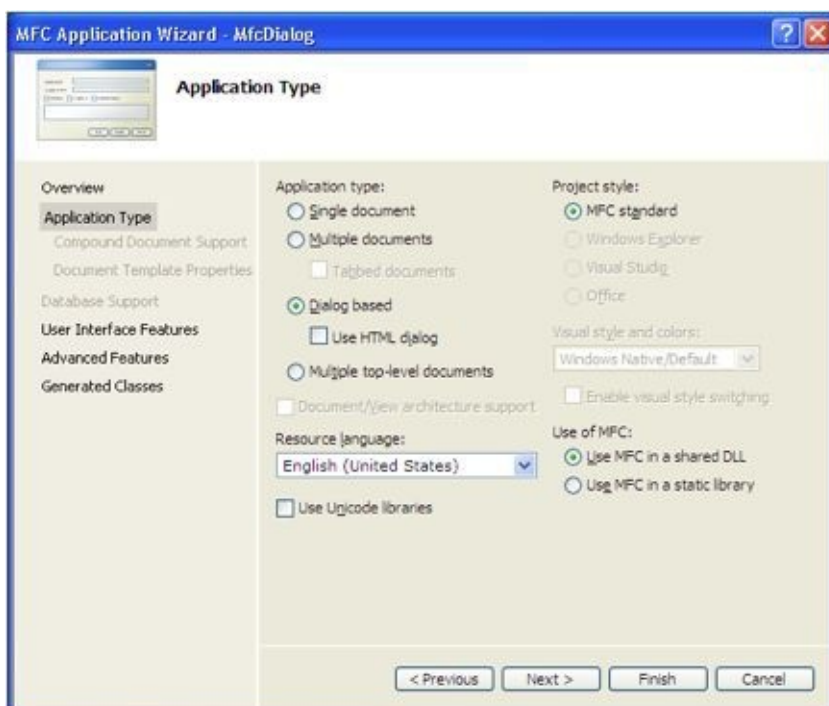


1.3 Creating Dialog Applications Using the MFC Application Wizard

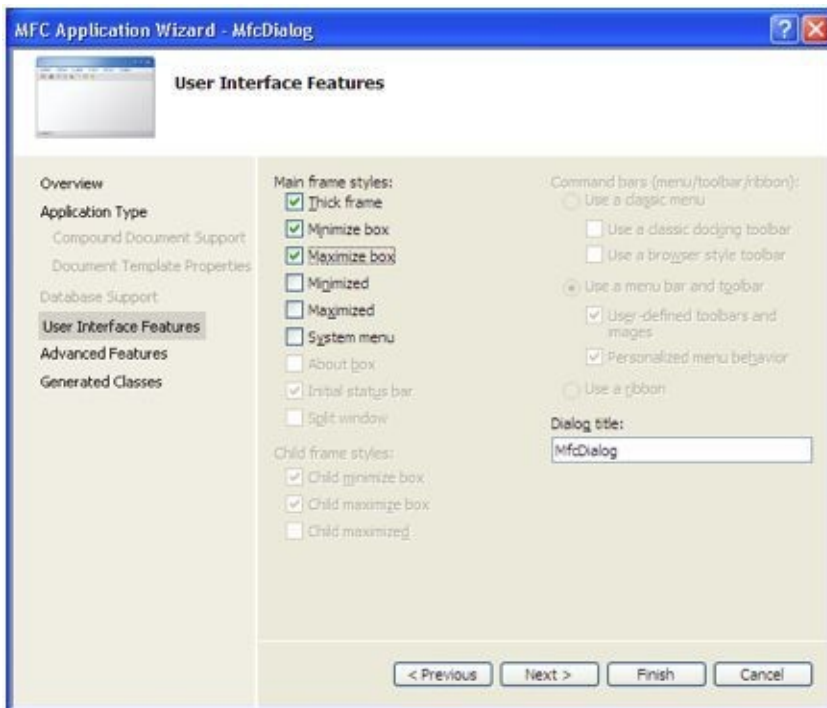
Alternatively you can create a Dialog application using the MFC Application Wizard in just a few steps. From the File menu select New > Project, and in the Installed Templates section of the New Project dialog select the MFC > MFC Application option. Give the project a name and select the location of the project folder, then click OK:



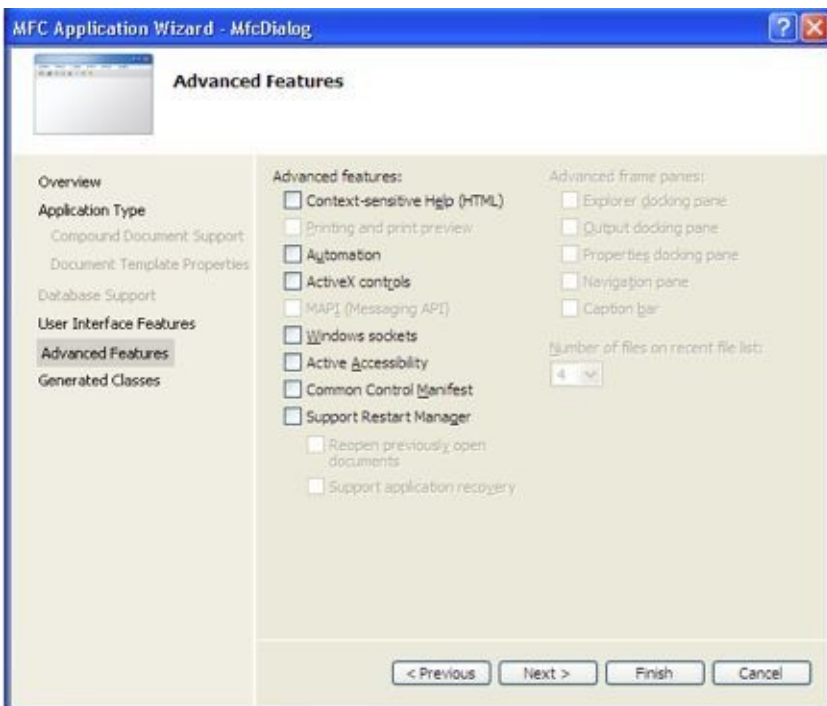
In the MFC Application Wizard that appears choose the Dialog option for the Application type and click Next:



Select the particular Main Frame styles you prefer and click Next:



Finally for our minimalist application we will uncheck any remaining options before clicking on Finish:



The Application Wizard will automatically generate the necessary resource files and the application whose user interface is a dialog box, the same as in the previous section:

MfcDialog

TODO: Place dialog controls here.

OK

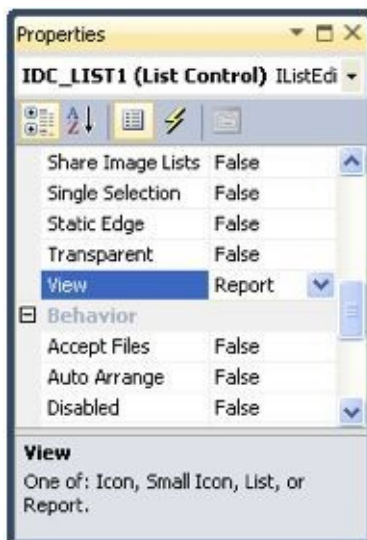
Cancel

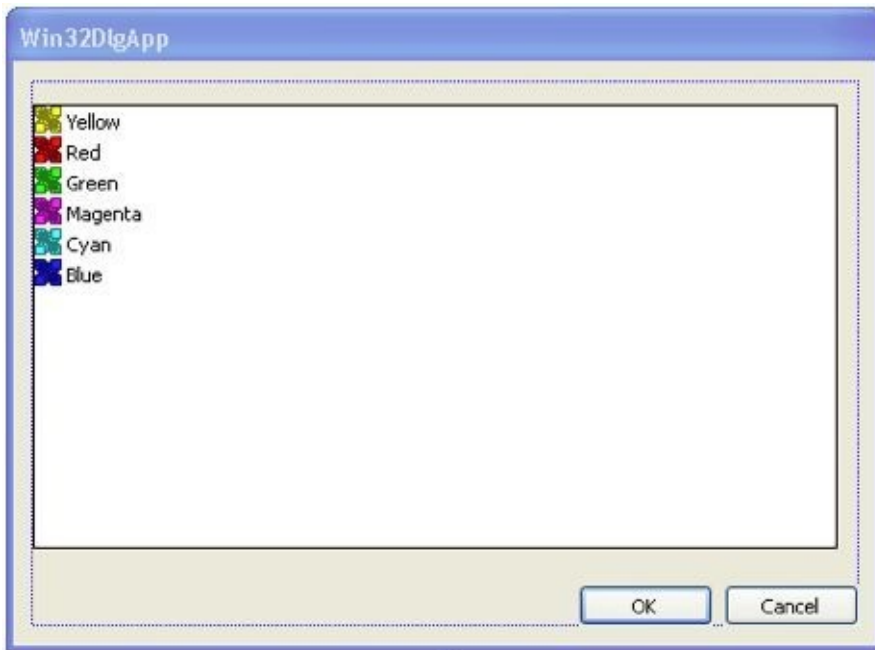
1.4 Adding Controls to Dialog Boxes: List Control

To add controls to your dialog, select the control you want from the Toolbox Window and drag it to the location on your dialog box. As an example, in the Dialog Application you created using the MFC Application Wizard, you want to select the List Control and drag this onto your Dialog:

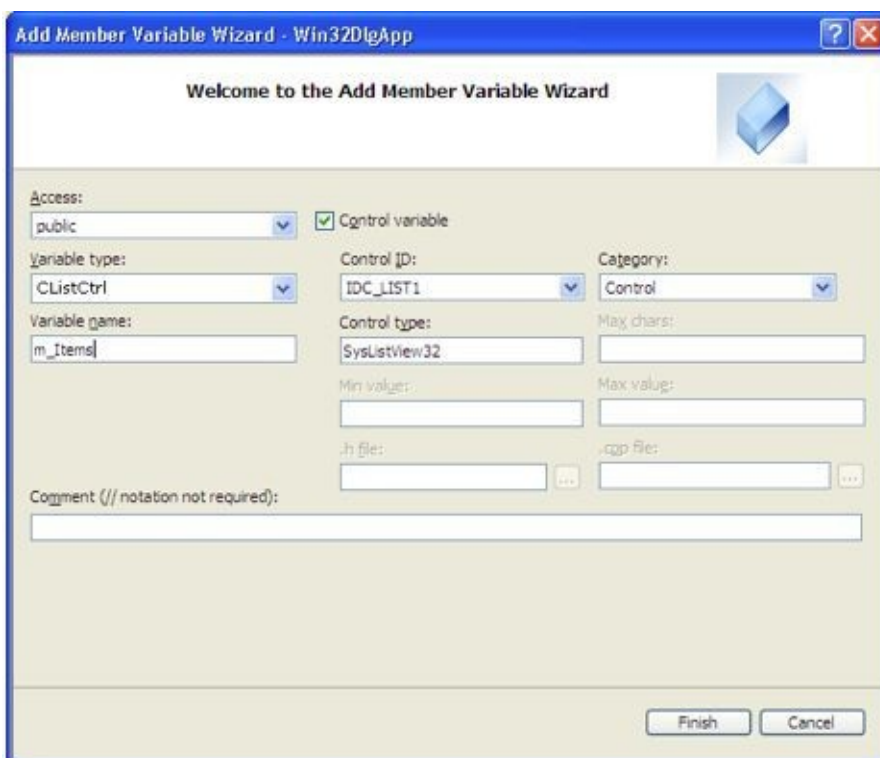


Right click on your List Control and select Properties. In the Appearance section set the View property to List. You can keep its ID property as **IDC_LIST1**:





Right-click the list control and select Add Variable. Set the Variable Name to **m_Items**. Click Finish.



Now populate the list control with items using the **CListCtrl::InsertItem** method. Use it as follows:

```
int InsertItem(const LVITEM* pItem );
```

This method requires an **LVITEM** pointer as an argument. The **LVITEM** structure is defined as follows:

```
typedef struct _LVITEM
{
    UINT mask;
    int iItem;
    int iSubItem;
    UINT state;
    UINT stateMask;
    LPTSTR pszText;
    int cchTextMax;
    int iImage;
    LPARAM lParam;
    #if (_WIN32_IE >= 0x0300)
    int iIndent;
    #endif
} LVITEM, FAR *LPLVITEM;
```

The mask parameter specifies the types of values you want to set for the current item. **iItem** specifies the zero-based index of the item being changed. **iSubItem** is the index of the sub item for the current value. If the current item will be the leader, the **iSubItem** is stored in a 0-based array. If it is a sub item, then it is stored in a 1-based array. The **pszText** variable is the item string displayed, the length of which can be specified by assigning a value to **cchTextMax**.

After initializing the **LVITEM**, it is passed to the **InsertItem** method to add it as a new list item. Here is an example that creates items and displays as a List view:

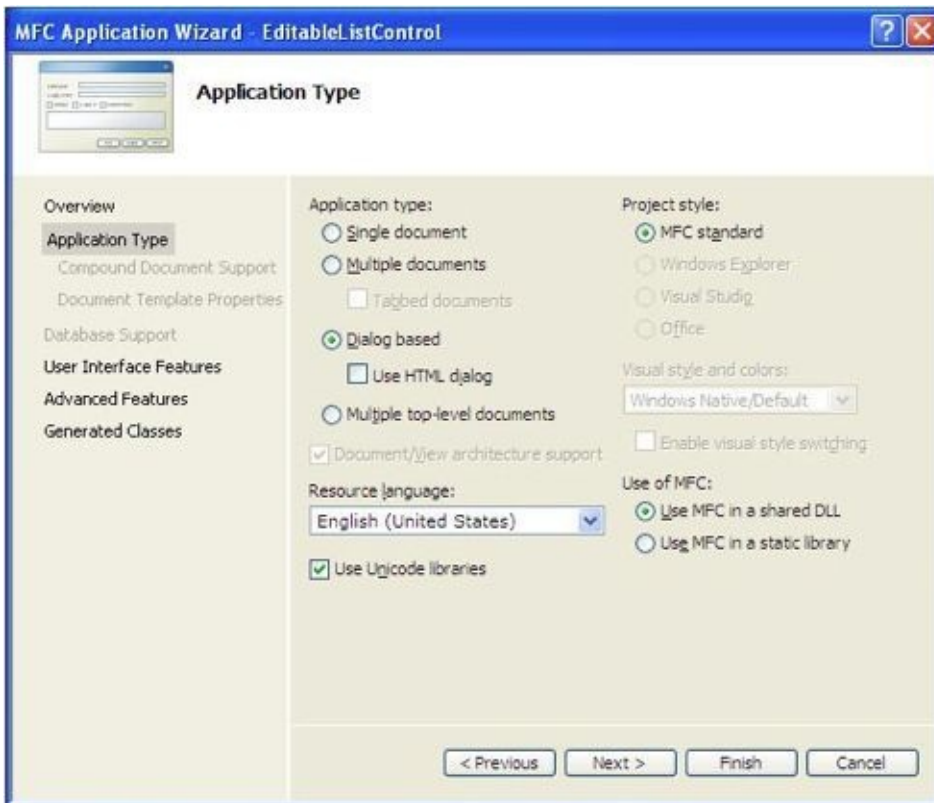


Creating Custom MFC Controls – an Editable List Control

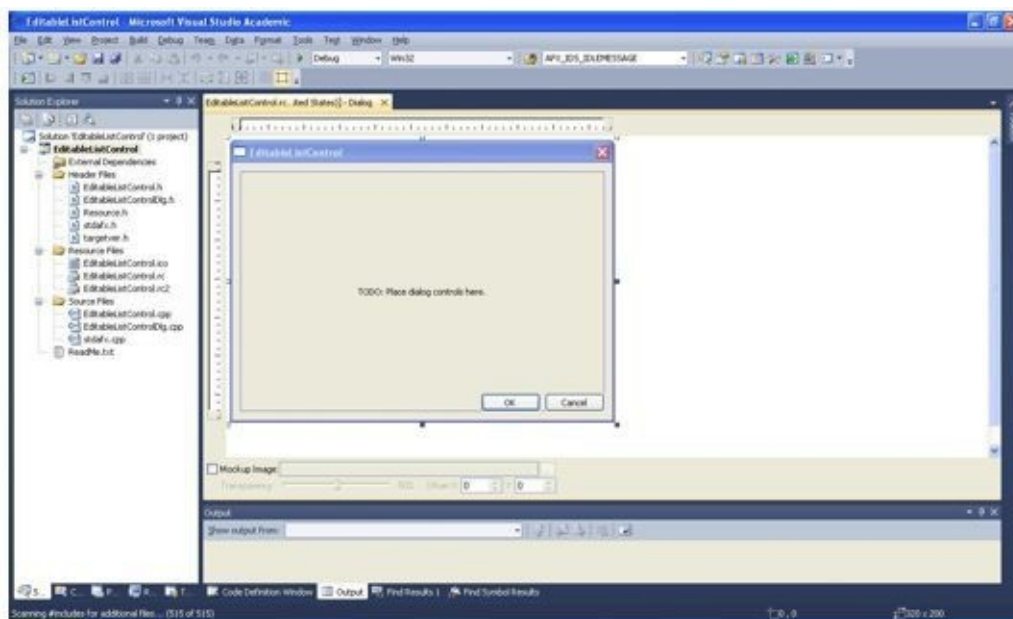
Sometimes you may need more than what the Microsoft Foundation Classes provide. The List Control, for example, is great for displaying lists of records in various modes. But what if you wish to make individual grid items in the list control editable, for example? The answer is to override the MFC class and provide custom methods to implement the behaviour you desire. This example describes the steps necessary to do just that.

Step 1: Create the MFC Project

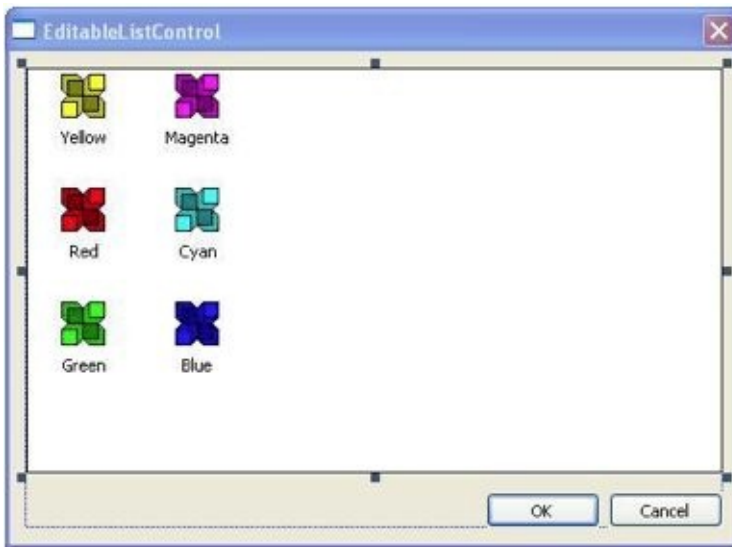
Start by creating a dialog-based application in Visual Studio. Select File > New > Project > MFC Application. When the Application Wizard is launched, select Dialog based as the Application Type:



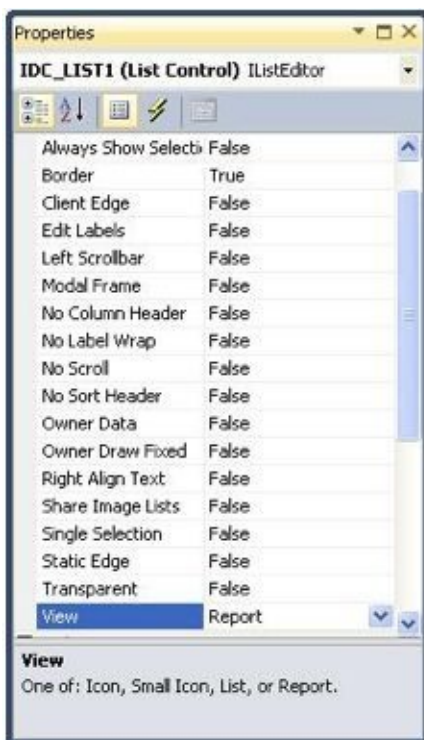
And then select Finish. In the Resources View, notice that a new dialog is created:



To get started, first delete the static control “TODO: Place dialog controls here” that gets created automatically. Then in the Toolbox, select the “List Control” and in the Resource View place this within your dialog area:

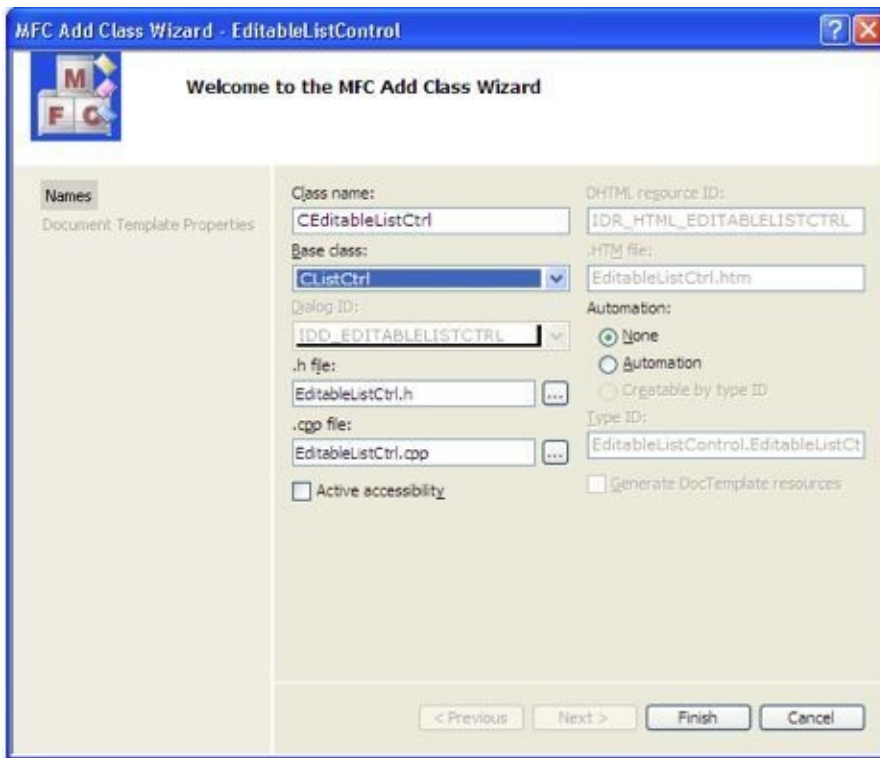


Right-click the List Control you just added and select Properties. In the Properties window, make sure the View section of Appearance is set to 'Report' style:



Step 2: Derive a class from CListCtrl

Create a new List Control class **CEditableListCtrl**, that publicly inherits from the standard **CListCtrl**. An efficient way to do this is to use the Class Wizard. Right click your project folder and select Class Wizard... Then press the Add Class button that appears in the MFC Class Wizard dialog:



In our new derived **CEditableListCtrl** class, we need to define six new methods in order to make the List Control grid items editable:

```
class CEditableListCtrl : public CListCtrl
{
public:
    int GetRowFromPoint( CPoint &point, int *col ) const;
    CEdit* EditSubLabel( int nItem, int nCol );
    void OnHScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    void OnVScroll(UINT nSBCode, UINT nPos, CScrollBar* pScrollBar);
    void OnEndLabelEdit(NMHDR* pNMHDR, LRESULT* pResult);
    void OnLButtonDown(UINT nFlags, CPoint point);
};
```

GetRowFromPoint – to determine the row and column number that the cursor falls on, if any:

```
int CEditableListCtrl::GetRowFromPoint( CPoint &point, int *col ) const
{
    int column = 0;
```

```

int row = HitTest( point, NULL );
if( col ) *col = 0;
// Make sure that the ListView is in LVS_REPORT
if( ( GetWindowLong( m_hWnd, GWL_STYLE ) & LVS_TYPEMASK ) !=
LVS_REPORT )
{
return row;
}
// Get the top and bottom row visible
row = GetTopIndex();
int bottom = row + GetCountPerPage();
if( bottom > GetItemCount() )
{
bottom = GetItemCount();
}
// Get the number of columns
CHeaderCtrl* pHeader = (CHeaderCtrl*)GetDlgItem( 0 );
int nColumnCount = pHeader->GetItemCount();
// Loop through the visible rows
for( ; row <= bottom; row++ )
{
// Get bounding rectangle of item and check whether point falls in it.
CRect rect;
GetItemRect( row, &rect, LVIR_BOUNDS );

if( rect.PtInRect(point) )
{
// Find the column
for( column = 0; column < nColumnCount; column++ )
{
int colwidth = GetColumnWidth( column );
if(point.x >= rect.left && point.x <= (rect.left + colwidth ))
{

```

```

if( col ) *col = column;
return row;
}
rect.left += colwidth;
}
}
}
return -1;
}

```

EditSubLabel – is a method to edit the individual cells of the List Control. Taking the row and column integers as arguments, **EditSubLabel** creates and makes visible an edit control of the appropriate size and text justification. (This also requires another edit control class, **CInPlaceEdit** which we derived from the standard **CEdit** class and is described in the next section.)

```

CEdit* CEditableListCtrl::EditSubLabel( int nItem, int nCol )
{
// The returned pointer should not be saved, make sure item visible
if( !EnsureVisible( nItem, TRUE ) ) return NULL;
// Make sure that column number is valid
CHeaderCtrl* pHeader = (CHeaderCtrl*)GetDlgItem(0);
int nColumnCount = pHeader->GetItemCount();
if( nCol >= nColumnCount || GetColumnWidth(nCol) < 5 ) return NULL;
// Get the column offset
int offset = 0;
for( int i = 0; i < nCol; i++ ) {
offset += GetColumnWidth( i );
}
CRect rect;
GetItemRect( nItem, &rect, LVIR_BOUNDS );
// Scroll horizontally if we need to expose the column
CRect rcClient;

```

```

GetClientRect( &rcClient );
if( offset + rect.left < 0 || offset + rect.left > rcClient.right )
{
    CSize size;
    size.cx = offset + rect.left;
    size.cy = 0;
    Scroll( size );
    rect.left -= size.cx;
}
// Get Column alignment
LV_COLUMN lvcol;
lvcol.mask = LVCF_FMT;
GetColumn( nCol, &lvcol );
DWORD dwStyle ;
if( (lvcol.fmt&LVCFMT_JUSTIFYMASK) == LVCFMT_LEFT ) {
    dwStyle = ES_LEFT;
}
else if( (lvcol.fmt&LVCFMT_JUSTIFYMASK) == LVCFMT_RIGHT ) {
    dwStyle = ES_RIGHT;
}
else {
    dwStyle = ES_CENTER;
}
rect.left += offset+4; ¬
rect.right = rect.left + GetColumnWidth( nCol ) - 3 ;
if( rect.right > rcClient.right) {
    rect.right = rcClient.right;
}
dwStyle |= WS_BORDER | WS_CHILD | WS_VISIBLE | ES_AUTOHSCROLL;
CEdit *pEdit = new CInPlaceEdit(nItem, nCol,.GetItemText( nItem, nCol ));
pEdit->Create( dwStyle, rect, this, IDC_LIST1 );
return pEdit;

```

```
}
```

Another essential modification is to add the means by which the user can initiate an edit of the selected List Control cell by modifying the OnLButtonDown method:

```
void CEditableListCtrl::OnLButtonDown(UINT nFlags, CPoint point)
{
    int index;
    CListCtrl::OnLButtonDown(nFlags, point);
    ModifyStyle(0, LVS_EDITLABELS);
    int colnum;
    if( ( index = GetRowFromPoint( point, &colnum ) ) != -1 )
    {
        UINT flag = LVIS_FOCUSED;
        if( (GetItemState( index, flag ) & flag) == flag )
        {
            // Add check for LVS_EDITLABELS
            if( GetWindowLong(m_hWnd, GWL_STYLE) & LVS_EDITLABELS )
            {
                EditSubLabel( index, colnum );
            }
        }
        else
        {
            SetItemState(
                index,
                LVIS_SELECTED | LVIS_FOCUSED,
                LVIS_SELECTED | LVIS_FOCUSED);
        }
    }
}
```

Step 3: Derive a class from CEdit

We create a derived instance of the **CEdit** class to satisfy a number of requirements: it needs to send the **LVN_ENDLABELEDIT** message and self-destruct upon completion of editing; expand a little to accommodate the text; and also terminate upon pressing the Escape or Enter keys or when the edit control loses focus.

```
class CInPlaceEdit : public CEdit
{
public:
    CInPlaceEdit(int iItem, int iSubItem, CString sInitText);
    // ClassWizard generated virtual function overrides
    //{{AFX_VIRTUAL(CInPlaceEdit)
public: virtual BOOL PreTranslateMessage(MSG* pMsg);
    //}}AFX_VIRTUAL
public: virtual ~CInPlaceEdit();
    // Generated message map functions
protected:
    //{{AFX_MSG(CInPlaceEdit)
afx_msg void OnKillFocus(CWnd* pNewWnd);
afx_msg void OnNcDestroy();
afx_msg void OnChar(UINT nChar, UINT nRepCnt, UINT nFlags);
afx_msg int OnCreate(LPCREATESTRUCT lpCreateStruct);
    //}}AFX_MSG
DECLARE_MESSAGE_MAP()
private:
    int m_iItem;
    int m_iSubItem;
    CString m_sInitText;
    BOOL m_bESC;
};
```

The derived **CEdit** control, **CInPlaceEdit**, is described as follows. The overridden **PreTranslateMessage()** determines whether certain key strokes make it to the edit control. **OnKillFocus()** sends the **LVN_ENDLABELEDIT** notification and destroys the edit control. The notification is sent to the parent of the list view control and not to the list

view control itself. When sending the notification, the **m_bESC** member variable is used to determine whether to send a NULL string. **OnNcDestroy()** is the appropriate place to destroy the C++ object. The **OnChar()** function terminates editing if the Escape or the Enter key is pressed.

Otherwise for any other character, it lets the base class function take care of it before it tries to determine if the control needs to be resized. The function gets the extent of the new string and then compares it to the dimension of the edit control. If the string is too long to fit within the edit control, it resizes the edit control after checking if there is sufficient space for it to expand.

OnCreate() creates the edit control, initialising it with the proper values.

Full code listing of the **CInPlaceEdit** control:

```
CInPlaceEdit::CInPlaceEdit(int iItem, int iSubItem, CString sInitText):m_sInitText(
sInitText )
{
m_iItem = iItem;
m_iSubItem = iSubItem;
m_bESC = FALSE;
}
CInPlaceEdit::~~CInPlaceEdit(){}
BEGIN_MESSAGE_MAP(CInPlaceEdit, CEdit)
//{{AFX_MSG_MAP(CInPlaceEdit)
ON_WM_KILLFOCUS()
ON_WM_NCDESTROY()
ON_WM_CHAR()
ON_WM_CREATE()
//}}AFX_MSG_MAP
END_MESSAGE_MAP()
//CInPlaceEdit message handlers
// Translate window messages before they are dispatched to the TranslateMessage
and DispatchMessage Windows functions.
BOOL CInPlaceEdit::PreTranslateMessage(MSG* pMsg)
{
```

```

if( pMsg->message == WM_KEYDOWN )
{
if(pMsg->wParam == VK_RETURN ||
pMsg->wParam == VK_DELETE ||
pMsg->wParam == VK_ESCAPE ||
GetKeyState( VK_CONTROL ) )
{
::TranslateMessage(pMsg);
::DispatchMessage(pMsg);
return TRUE;
// DO NOT process further
}
}
return CEdit::PreTranslateMessage(pMsg);
}

// Called immediately before losing the input focus
void CInPlaceEdit::OnKillFocus(CWnd* pNewWnd)
{
CEdit::OnKillFocus(pNewWnd);
CString str;
GetWindowText(str);
DestroyWindow();
}

// Called when nonclient area is being destroyed
void CInPlaceEdit::OnNcDestroy()
{
CEdit::OnNcDestroy();
delete this;
}

// Called for nonsystem character keystrokes
void CInPlaceEdit::OnChar(UINT nChar, UINT nRepCnt, UINT nFlags)
{

```

```

if( nChar == VK_ESCAPE || nChar == VK_RETURN)
{
if( nChar == VK_ESCAPE )
{
m_bESC = TRUE;
}
GetParent()->SetFocus();
return;
}
CEdit::OnChar(nChar, nRepCnt, nFlags);
// Resize edit control if needed
CString str;
GetWindowText( str );
CWindowDC dc(this);
CFont *pFont = GetParent()->GetFont();
CFont *pFontDC = dc.SelectObject( pFont );
CSize size = dc.GetTextExtent( str );
dc.SelectObject( pFontDC );
size.cx += 5;
// Get the client rectangle
CRect rect, parentrect;
GetClientRect( &rect );
GetParent()->GetClientRect( &parentrect );
// Transform rectangle to parent coordinates
ClientToScreen( &rect );
GetParent()->ScreenToClient( &rect );
// Check whether control needs resizing and if sufficient space to grow
if( size.cx > rect.Width() )
{
if( size.cx + rect.left < parentrect.right )
{
rect.right = rect.left + size.cx;

```

```

}
else
{
    rect.right = parentrect.right;
}
MoveWindow( &rect );
}

// Construct list control item data
LV_DISPINFO dispinfo;
dispinfo.hdr.hwndFrom = GetParent()->m_hWnd;
dispinfo.hdr.idFrom = GetDlgCtrlID();
dispinfo.hdr.code = LVN_ENDLABELEDIT;
dispinfo.item.mask = LVIF_TEXT;
dispinfo.item.iItem = m_iItem;
dispinfo.item.iSubItem = m_iSubItem;
dispinfo.item.pszText = m_bESC ? NULL : LPTSTR((LPCTSTR)str);
dispinfo.item.cchTextMax = str.GetLength();
// Send this Notification to parent of ListView ctrl
CWnd* pWndViewAttachmentsDlg = GetParent()->GetParent();
if ( pWndViewAttachmentsDlg )
{
    pWndViewAttachmentsDlg->SendMessage(
        WM_NOTIFY_DESCRIPTION_EDITED,
        GetParent()->GetDlgCtrlID(),
        (LPARAM)&dispinfo );
}
}

// Called when application requests the Windows window be created by calling
// Create/CreateEx member function.
int CInPlaceEdit::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CEdit::OnCreate(lpCreateStruct) == -1)

```

```

{
return -1;
}

// Set the proper font
CFont* font = GetParent()->GetFont();
SetFont( font );
SetWindowText( m_sInitText );
SetFocus();
SetSel( 0, -1 );
return 0;
}

```

*Step 4: Add the **CEditableListCtrl** as a control variable*

In the main **CEditableListControlDlg** class, add **CEditableListCtrl** as a control variable:

```

CEditableListCtrl m_EditableList;

```

And modify **DoDataExchange** accordingly:

```

void CEditableListControlDlg::DoDataExchange(CDataExchange* pDX)
{
CDialogEx::DoDataExchange(pDX);

DDX_Control(pDX, IDC_LIST1, m_EditableList);
}

```

And in **OnInitDialog** let's add a few sample list control entries:

```

BOOL CEditableListControlDlg::OnInitDialog()
{
CDialogEx::OnInitDialog();

```

// Add “About...” menu item to system menu.

// IDM_ABOUTBOX must be in the system command range.

ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);

ASSERT(IDM_ABOUTBOX < 0xF000);

CMenu* pSysMenu = GetSystemMenu(FALSE);

if (pSysMenu != NULL)

{

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);

}

}

// Set the icon for this dialog. The framework does this automatically

// when the application's main window is not a dialog

SetIcon(m_hIcon, TRUE); // Set big icon

SetIcon(m_hIcon, FALSE); // Set small icon

// TODO: Add extra initialization here

CRect rect;

CWnd *pWnd = this->GetDlgItem(IDC_LIST1);

pWnd->GetWindowRect(&rect);

this->ScreenToClient(&rect); //optional step - see below

m_xoffset = rect.left;

```
m_yoffset = rect.top;
```

```
LVCOLUMN lvColumn;
```

```
int nCol;
```

```
lvColumn.mask = LVCF_FMT | LVCF_TEXT | LVCF_WIDTH;
```

```
lvColumn.fmt = LVCFMT_LEFT;
```

```
lvColumn.cx = 150;
```

```
lvColumn.pszText = "Name";
```

```
nCol = m_EditableList.InsertColumn(0, &lvColumn);
```

```
lvColumn.mask = LVCF_FMT | LVCF_TEXT | LVCF_WIDTH;
```

```
lvColumn.fmt = LVCFMT_CENTER;
```

```
lvColumn.cx = 150;
```

```
lvColumn.pszText = "Occupation";
```

```
m_EditableList.InsertColumn(1, &lvColumn);
```

```
lvColumn.mask = LVCF_FMT | LVCF_TEXT | LVCF_WIDTH;
```

```
lvColumn.fmt = LVCFMT_LEFT;
```

```
lvColumn.cx = 150;
```

```
lvColumn.pszText = "Country";
```

```
m_EditableList.InsertColumn(2, &lvColumn);
```

```
m_EditableList.SetExtendedStyle(m_EditableList.GetExtendedStyle() |  
LVS_EX_FULLROWSELECT | LVS_EDITLABELS);
```

```
// Insert a few example list items
```

```
int l_iItem = m_EditableList.InsertItem(LVIF_TEXT|LVIF_STATE, 0, "Andrew", 0,  
LVIS_SELECTED, 0, 0);
```

```
m_EditableList.SetItemText( l_iItem, 1, "Bricklayer" );
```

```
m_EditableList.SetItemText( l_iItem, 2, "Australia" );
```

```
l_iItem = m_EditableList.InsertItem(LVIF_TEXT|LVIF_STATE, 0, "Peter", 0,  
LVIS_SELECTED, 0, 0);
```

```
m_EditableList.SetItemText( l_iItem, 1, "Lecturer" );
```



```
m_EditableList.SetItemText( l_iItem, 2, "New Zealand" );
```

```
l_iItem = m_EditableList.InsertItem(LVIF_TEXT|LVIF_STATE, 0, "Richard", 0,  
LVIS_SELECTED, 0, 0);
```

```
m_EditableList.SetItemText( l_iItem, 1, "Dentist" );
```

```
m_EditableList.SetItemText( l_iItem, 2, "Botswana" );
```

```
return TRUE; // return TRUE unless you set the focus to a control  
}
```

Step 5: Introduce notifiers for handling updates and Windows messages

Beginning with the Windows message map:

```
BEGIN_MESSAGE_MAP(CEditableListControlDlg, CDialogEx)  
ON_WM_SYSCOMMAND()  
ON_WM_PAINT()  
ON_WM_QUERYDRAGICON()  
ON_NOTIFY(NM_CLICK, IDC_LIST1, OnNMClickList)  
ON_MESSAGE(WM_NOTIFY_DESCRIPTION_EDITED,  
OnNotifyDescriptionEdited)  
END_MESSAGE_MAP()
```

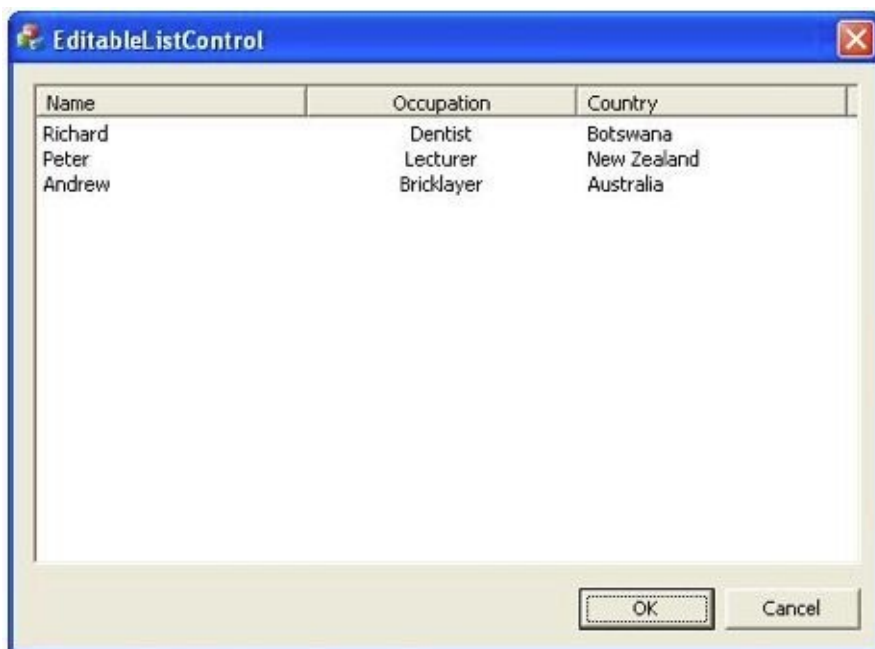
For when the user clicks on the editable list control in particular:

```
void CEditableListControlDlg::OnNMClickList(NMHDR *pNMHDR, LRESULT  
*pResult)  
{  
m_fClickedList = true;  
m_EditableList.OnLButtonDown( MK_LBUTTON, InterviewListCursorPosition() );  
*pResult = 0;  
}
```

Also add a notifier for handling when the edit control has been updated:

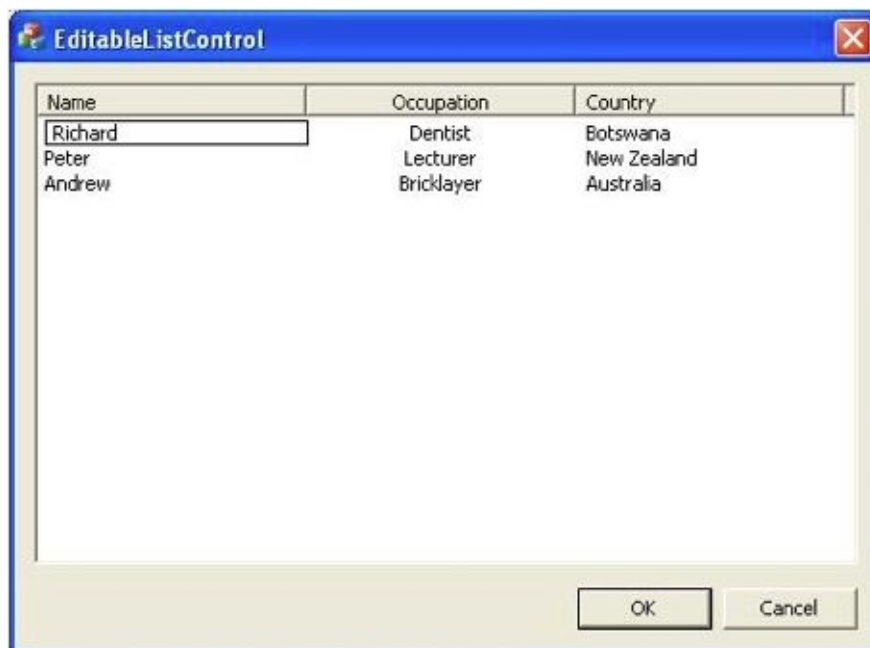
```
LRESULT CEditableListControlDlg::OnNotifyDescriptionEdited(  
WPARAM wParam,  
LPARAM lParam)  
{  
// Get the changed Description field text via the callback  
LV_DISPINFO* dispinfo = reinterpret_cast<LV_DISPINFO*>(lParam);  
  
// Persist the selected attachment details upon updating its text  
m_EditableList.SetItemText(  
dispinfo->item.iItem,  
dispinfo->item.iSubItem,  
dispinfo->item.pszText );  
  
return 0;  
}
```

Run the sample project to see the default List Control entries displayed:



And then see how upon left-clicking individual cells, they become editable, an outline

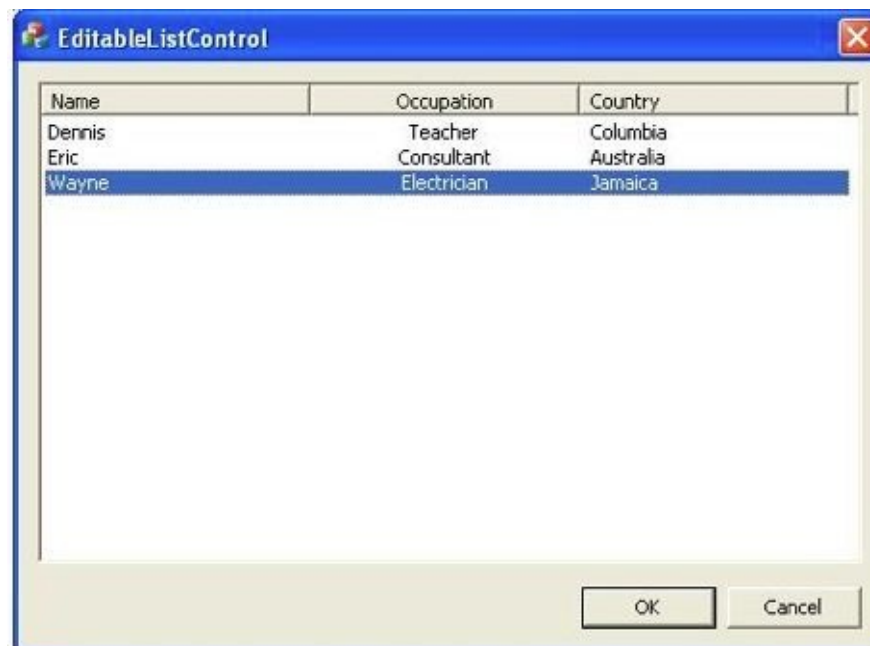
appearing around the editable control:



The screenshot shows a dialog box titled "EditableListControl". It contains a table with three columns: "Name", "Occupation", and "Country". The first row is highlighted, and the "Name" field is in edit mode, showing "Richard". The other rows are "Peter" (Lecturer, New Zealand) and "Andrew" (Bricklayer, Australia). At the bottom, there are "OK" and "Cancel" buttons.

Name	Occupation	Country
Richard	Dentist	Botswana
Peter	Lecturer	New Zealand
Andrew	Bricklayer	Australia

And that any of the fields can be edited if the user chooses:



The screenshot shows the same dialog box, but now the third row is highlighted, and the "Name" field is in edit mode, showing "Wayne". The other rows are "Dennis" (Teacher, Columbia) and "Eric" (Consultant, Australia). At the bottom, there are "OK" and "Cancel" buttons.

Name	Occupation	Country
Dennis	Teacher	Columbia
Eric	Consultant	Australia
Wayne	Electrician	Jamaica

Chapter Summary

To summarize in the first of these 10 master lessons, we have learned how to:

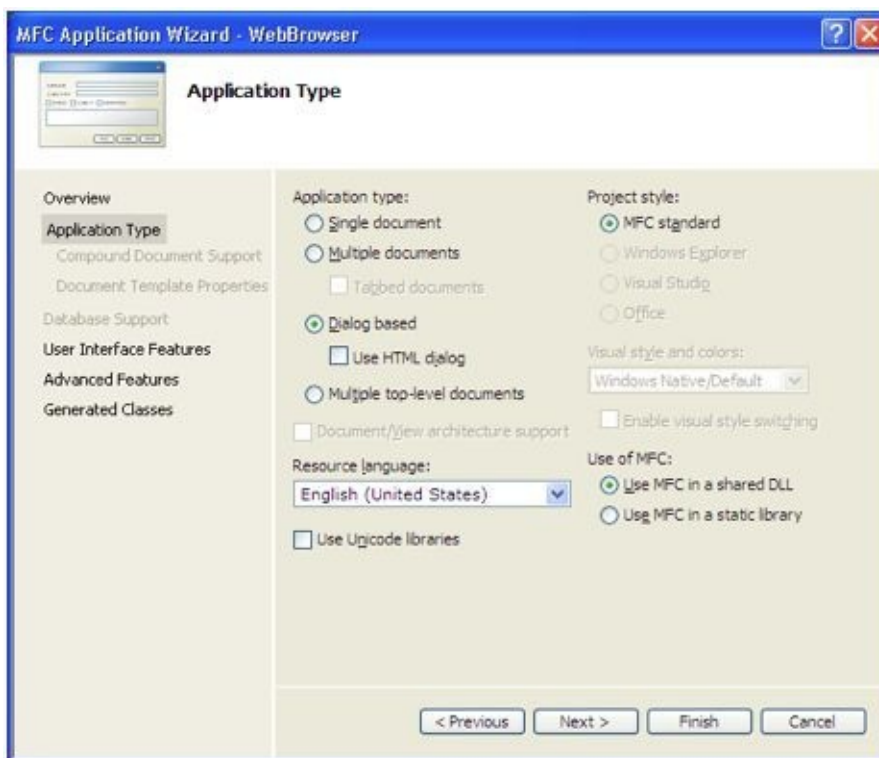
- Create our first Windows application with event handling.
- Create a dialog application both programmatically and by using the MFC Application Wizard.
- Add controls to dialog applications.
- Create customized MFC controls, such as the CListCtrl (ordinarily read-only), so that its individual grid items become modifiable.

Chapter 2: Further Windows Programming

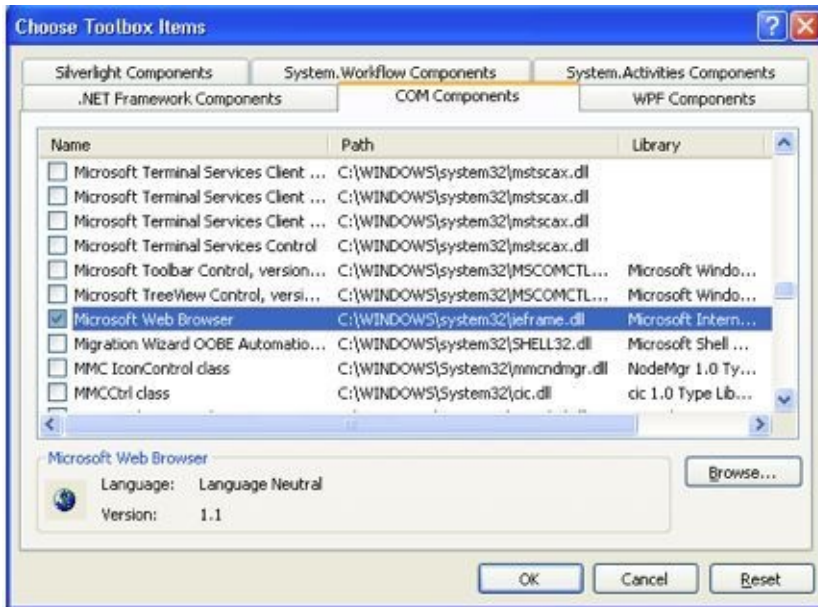
2.1 Building a web browser

Creating a Web browser in Visual C++ is remarkably easy. That's because Microsoft Internet Explorer comes with Visual C++ and by using a few simple steps, it can be used as a Windows control similar to any other. The process of creating a simple Web browser can be summarized as: add a Microsoft Web Browser control to your Windows program, create a member control variable for it, and use the **Navigate** method to browse the online location of your choice. Then add an edit control to specify the URL of the web site you wish to browse.

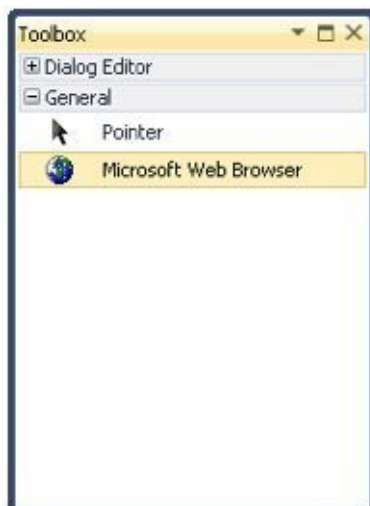
Use the AppWizard to create a new dialog-based application called WebBrowser.



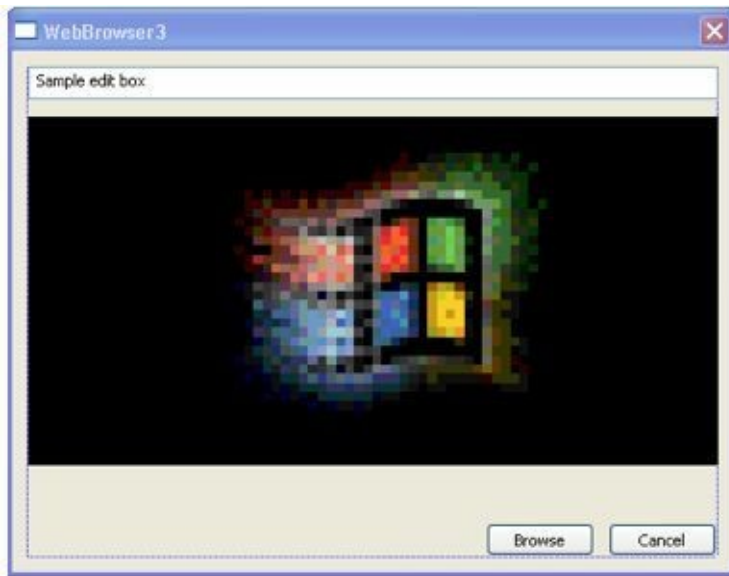
Make sure the Microsoft Web Browser is available for use in your application. From the Toolbox window, right-click and select Choose Items... (For older versions of Microsoft Visual Studio you may need to access this option by instead selecting Project > Add to Project > Components and Controls and then double-clicking the entry marked Registered ActiveX Controls.) When the dialog to choose Toolbox items appears, select the COM Components tab, and select the Microsoft Web Browser control:



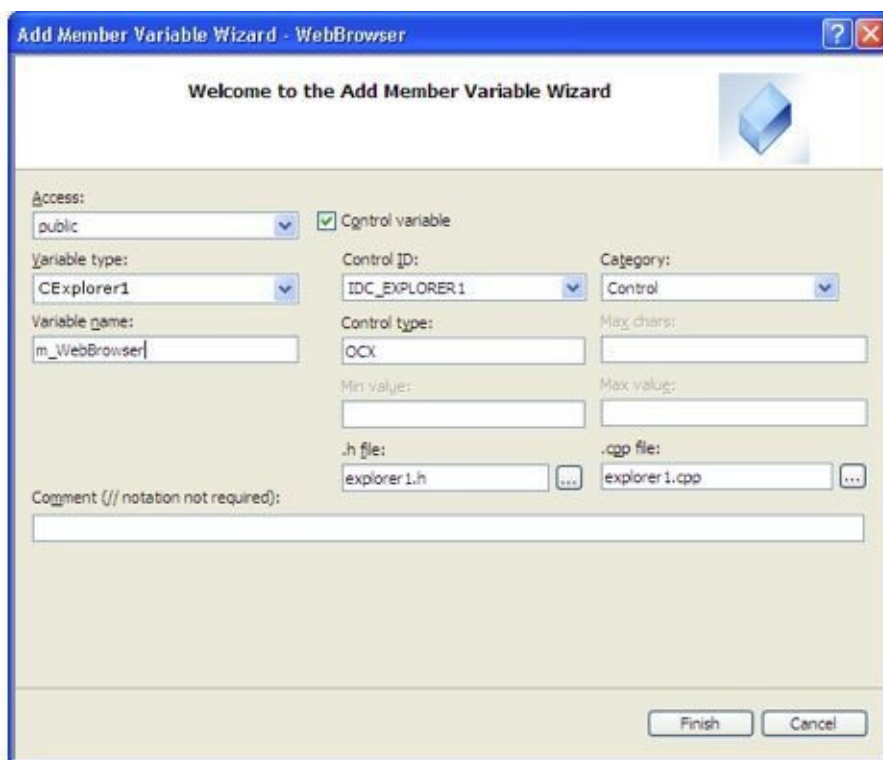
The browser control will then appear in the Toolbox window:



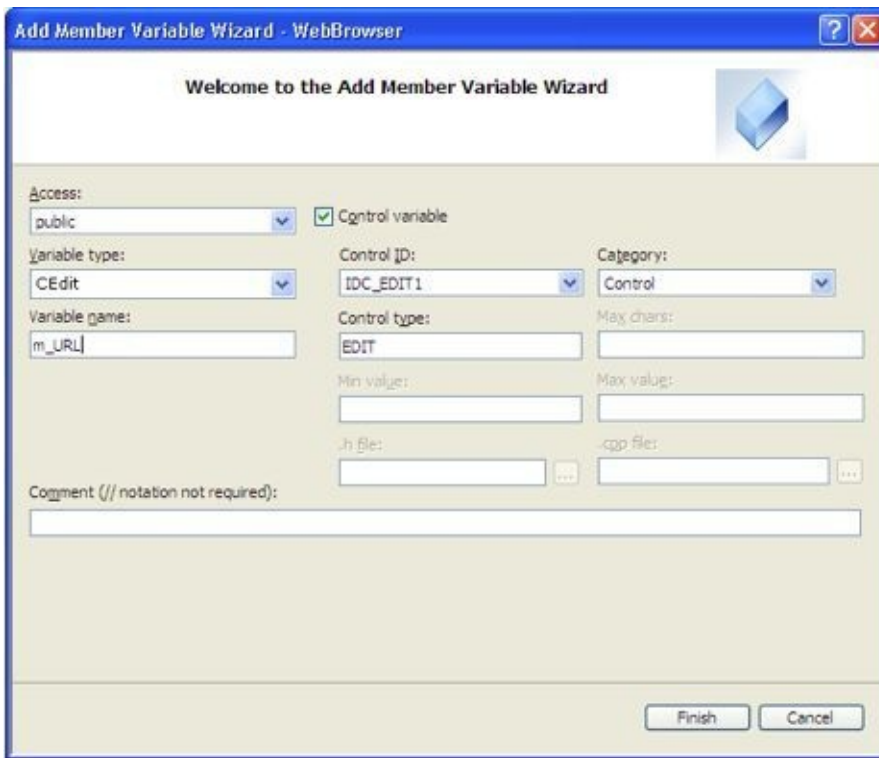
In your Resource View, drag this newly added control onto your dialog window while resizing it according to your preference. Then add an edit control with which to enter a URL string variable, as well as a button to your dialog with the caption Browse (you can modify the OK button that is automatically generated by setting its properties). Right-click the OK button, select Properties, and rename the caption from OK to Browse. Rename its ID from **IDOK** to **ID_BROWSE**:



Now create the member variables for each of the controls that have been added. Create a member variable for the Web Browser control by right-clicking on the Web Browser control and selecting Add Variable. In the Add Member Variable Wizard that appears, give it a name such as **m_WebBrowser** and click OK:



In the same manner, create a member variable for edit control as well: right-click the edit control and select Add Variable, and give this control a name such as **m_URL**:



Now create an event-handler for the Browse button, so that when clicked the application navigates the Web Browser to the location entered in the edit control. Double-click on the Browse button in the Resource View and this will automatically generate the event-handling method for you. In this method we simply add the few lines of code that will navigate our Web browser to the desired internet location:

```
void CWebBrowserDlg::OnBnClickedButton()  
{  
    CString l_strURL;  
    m_URL.GetWindowTextA( l_strURL );  
    m_WebBrowser.Navigate( l_strURL, 0, 0, 0, 0 );  
}
```

Run the program as shown: enter the desired internet location and click the Browse button:



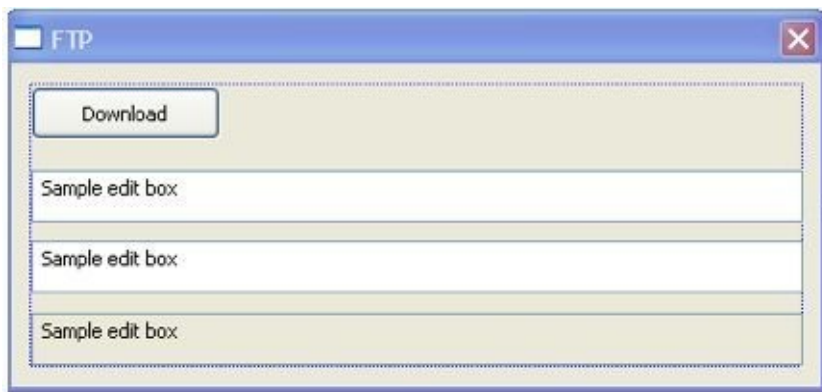
2.2 Downloading files via File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is a protocol for transferring data files over the Internet. This next example will now demonstrate how to use FTP to download a file from an FTP site. On many FTP sites, files seem to come and go but there is one file that has remained for years and looks set to stay that way: the text file of the FTP standards itself, rfc959.txt:

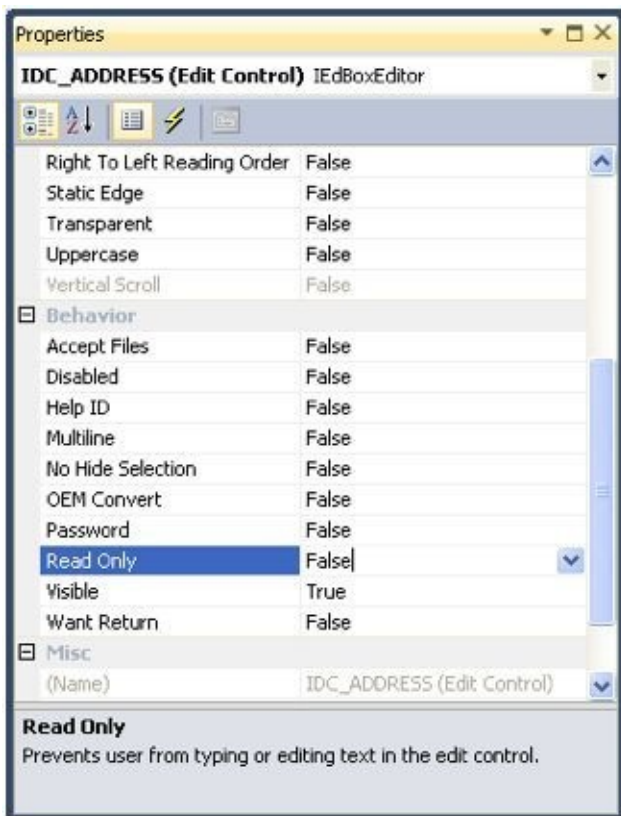
`ftp://ftp.funet.fi/pub/standards/RFC/rfc959.txt`

We will download rfc959.txt in our example program. This dialog will display a button named “Download” and a read-only text box to display the status of the download.

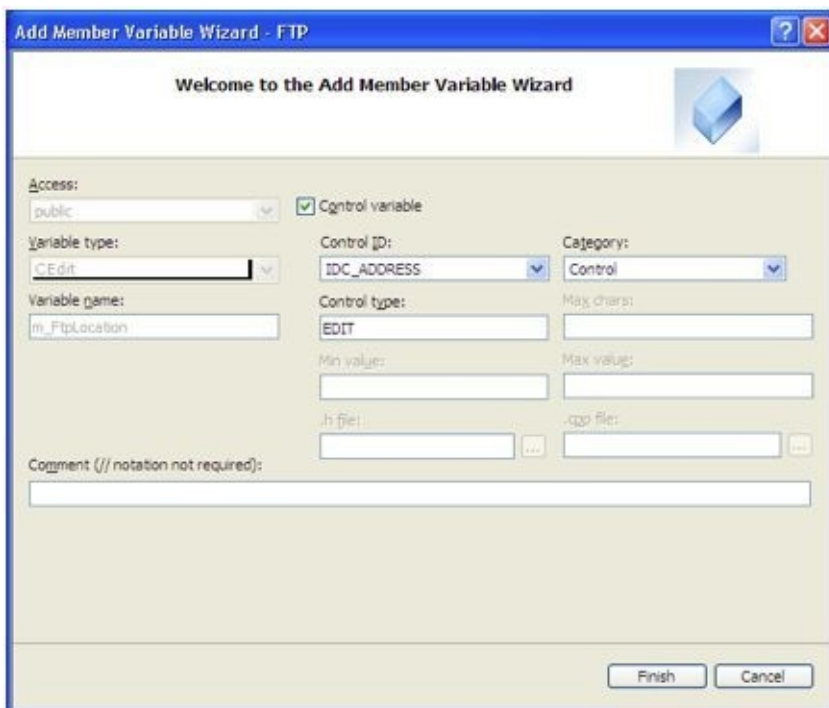
Use the AppWizard to create a dialog-based application named FTP. In this dialog we will need just one button (labeled Download) and three edit boxes. Two of the edit boxes will be editable: one to enter the location of the FTP site and another to enter the file path. Another read-only edit box will be added to display the status/success of the download:



To set the read-only status of an edit control, right-click the control, select Properties, and set the Read Only property. This shows the Read-only property of the FTP address edit control:



Create a member variable for the first edit control. We will use this to enter the FTP file location. And in the Resource View, right-click the control, select Add Variable and give it a name such as **m_FileLocation**:



Repeat for the next edit control, the one we will use to enter the path location. **Name it m_FilePath:**

Add Member Variable Wizard - FTP

Welcome to the Add Member Variable Wizard

Access: public ☒ Control variable

Variable type: CEdit Control ID: IDC_FILEPATH Category: Control

Variable name: m_FilePath Control type: EDIT Max chars:

Min value: Max value:

.h file: .cpp file:

Comment (// notation not required):

Finish Cancel

For the third read-only edit control, we will also add a variable, this time calling it **m_StatusText**:

Add Member Variable Wizard - FTP

Welcome to the Add Member Variable Wizard

Access: public ☒ Control variable

Variable type: CEdit Control ID: IDC_STATUS Category: Control

Variable name: m_StatusText Control type: EDIT Max chars:

Min value: Max value:

.h file: .cpp file:

Comment (// notation not required):

Finish Cancel

In the OnInitDialog method, after the comment telling you to add your own initialization code, we will set some defaults for the default FTP address of the file we wish to download:

```
CString l_strFtpLocation = "ftp.funet.fi";  
m_FtpLocation.SetWindowTextA( l_strFtpLocation );  
CString l_strFilePath = "pub/standards/RFC/rfc959.txt";  
m_FilePath.SetWindowTextA( l_strFilePath );
```

Double click the Download button to automatically generate the event handling code when this button is clicked. On clicking the Download button, this routine will first obtain the FTP location and file path from the edit controls. Then it creates a new Internet session object pointer, and if successful it creates an FTP object pointer using the FTP location passed to it, which in this example is "ftp.funet.fi".

Finally it will attempt to download the file by passing the FTP file path location and writing the file to the chosen destination, which in this program has been hard coded to C:\Temp.

```
void CFTPDlg::OnBnClickedDownload()  
{  
    CString strFtpLocation, strFilePath;  
    m_FtpLocation.GetWindowTextA( strFtpLocation );  
    m_FilePath.GetWindowTextA( strFilePath );  
    int nPosition = strFilePath.ReverseFind( '/' );  
  
    CString strFileName = nPosition == -1 ?  
    strFilePath : strFilePath.Mid( nPosition + 1 );  
  
    CString strFtpFilePath = strFtpLocation + "/" + strFilePath;  
    CString strDestinationPath = "C:\\temp";  
  
    CInternetSession* pInternetSession = new CInternetSession();  
    CFTPConnection* pFTPConnection = NULL;  
  
    if ( NULL == pInternetSession )  
    {
```

```
MessageBox( NULL, "Could not establish Internet session", MB_OK );
}

pFTPConnection = pInternetSession->GetFtpConnection( strFtpLocation );

if ( NULL == pFTPConnection )
{
    MessageBox( NULL, "Could not establish FTP connection", MB_OK );
}

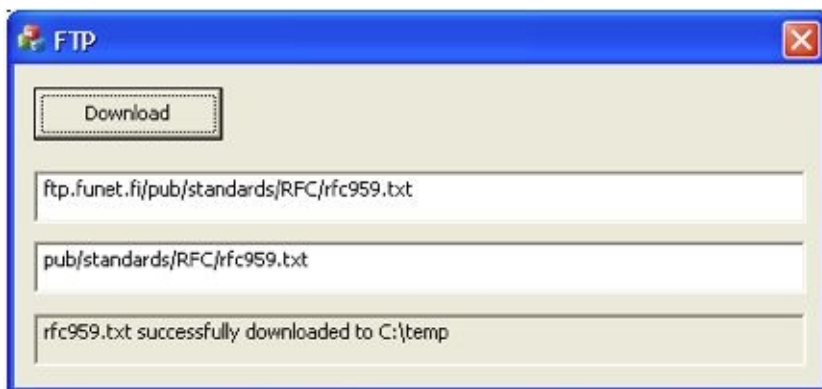
Sleep( 5000 );

BOOL success = pFTPConnection->GetFile(
    strFilePath,
    strDestinationPath + "\\" + strFileName,
    FALSE,
    FILE_ATTRIBUTE_NORMAL,
    FTP_TRANSFER_TYPE_ASCII,
    INTERNET_FLAG_DONT_CACHE );

if ( TRUE == success )
{
    m_FtpLocation.SetWindowTextA( strFtpFilePath );
    CString strDownloadMessage = strFileName +
        " successfully downloaded to " +
        strDestinationPath;
    m_StatusText.SetWindowTextA( strDownloadMessage );
}
else
{
    m_StatusText.SetWindowTextA( "Download failed." );
}
```

```
UpdateData( FALSE );  
pFTPConnection->Close();  
pInternetSession->Close();  
}
```

Running the application accesses the file from the specified FTP site and writes it to the chosen destination:



2.3 Playing Media Files using DirectShow

DirectShow is a framework consisting of a suite of APIs that can be used by software developers for media streaming on Windows platforms. Using DirectShow enables you to play or capture audio and video streams or edit media. This section is a high-level introduction to using DirectShow that shows you how to play an audio or a video file.

There are some pre-requisites for using DirectShow, however. All applications using DirectShow must include the **Dshow.h** header file and link to the static library file **strmiids.lib**. These are required for all DirectShow applications:

```
#include <dshow.h>

#pragma comment(lib, "Strmiids.lib")
```

Given that DirectShow is based on the Component Object Model (COM), some knowledge of applying COM client programming is necessary to write a DirectShow application. (Should you at some point wish to extend DirectShow to support new formats or develop your own custom effects, you would need to write your own components and implement them as COM objects.)

It is necessary to initialize the COM library beforehand, using **CoInitialize()** and then checking the result:

```
HRESULT hr = CoInitialize(NULL);
if (FAILED(hr))
{
    std::cout << "Could not initialize COM library" << std::endl;
    return;
}
```

When you are finished using COM you must also uninitialize it using **CoUninitialize()**, meaning that all DirectShow function calls take place between **COM CoInitialize()** and **CoUninitialize()**.

Define the DirectShow interfaces you will need for this application by building the filter

graph and then controlling media streaming and media event handling:

```
IGraphBuilder *pGraph = NULL;  
IMediaControl *pControl = NULL;  
IMediaEvent *pEvent = NULL;
```

Call **CoCreateInstance()** to create the Filter Graph Manager. A filter graph in processing media is a directed graph whose nodes represent data processing stages and whose edges represent uni-directional data flow. The Filter Graph Manager also handles synchronization, event notification, and other aspects of the controlling the filter graph.

```
hr = CoCreateInstance(  
CLSID_FilterGraph,  
NULL,  
CLSCTX_INPROC_SERVER,  
IID_IGraphBuilder,  
(void **)&pGraph );
```

Once the instance of the Filter Graph Manager is created, the real work starts. Use the returned **IGraphBuilder** pointer to query for the **IMediaControl** and **IMediaEvent** interfaces:

```
hr = pGraph->QueryInterface(IID_IMediaControl, (void **)&pControl);  
hr = pGraph->QueryInterface(IID_IMediaEvent, (void **)&pEvent);
```

Then use **IGraphBuilder::RenderFile** to build the filter graph automatically for the media supplied to it – the Filter Graph manager will connect the appropriate filters for the specified media file. This is otherwise known as “Intelligent connect” in DirectShow:

```
hr = pGraph->RenderFile( L”C:\temp\Song.mp4”, NULL );
```

Use **IMediaControl::Run** to start the data flow in the filter graph. **IMediaControl** provides the methods to run, pause or stop the media.

Use **IMediaEvent::WaitForCompletion** to wait for it to complete (caution: using **INFINITE** in a real application could cause it to block indefinitely.)

```
pControl->Run();  
pEvent->WaitForCompletion( INFINITE, &evCode );
```

Full code listing:

```
#include <dshow.h>  
#include <iostream>  
#pragma comment(lib, "Strmiids.lib")  
  
void main()  
{  
    IGraphBuilder *pGraph = NULL;  
    IMediaControl *pControl = NULL;  
    IMediaEvent *pEvent = NULL;  
  
    // Initialize the COM library.  
    HRESULT hr = CoInitialize( NULL );  
    if ( FAILED( hr ) )  
    {  
        std::cout << "Could not initialize COM library" << std::endl;  
        return;  
    }  
  
    // Create the filter graph manager and query for interfaces.  
    hr = CoCreateInstance(  
        CLSID_FilterGraph,  
        NULL,  
        CLSCTX_INPROC_SERVER,  
        IID_IGraphBuilder,  
        (void **)&pGraph );
```

```

if ( FAILED( hr ) )
{
std::cout << "Could not create the Filter Graph Manager."
<< std::endl;
return;
}

hr = pGraph->QueryInterface( IID_IMediaControl, (void **)&pControl );
hr = pGraph->QueryInterface( IID_IMediaEvent, (void **)&pEvent);

// Build and run graph. NOTE: change path to an actual file on your system
hr = pGraph->RenderFile( L"C:\temp\Song.mp4", NULL );

if ( SUCCEEDED( hr ) )
{
hr = pControl->Run();
if ( SUCCEEDED( hr ) )
{
long evCode;
pEvent->WaitForCompletion( INFINITE, &evCode );
}
}

pControl->Release();
pEvent->Release();
pGraph->Release();
CoUninitialize();
}

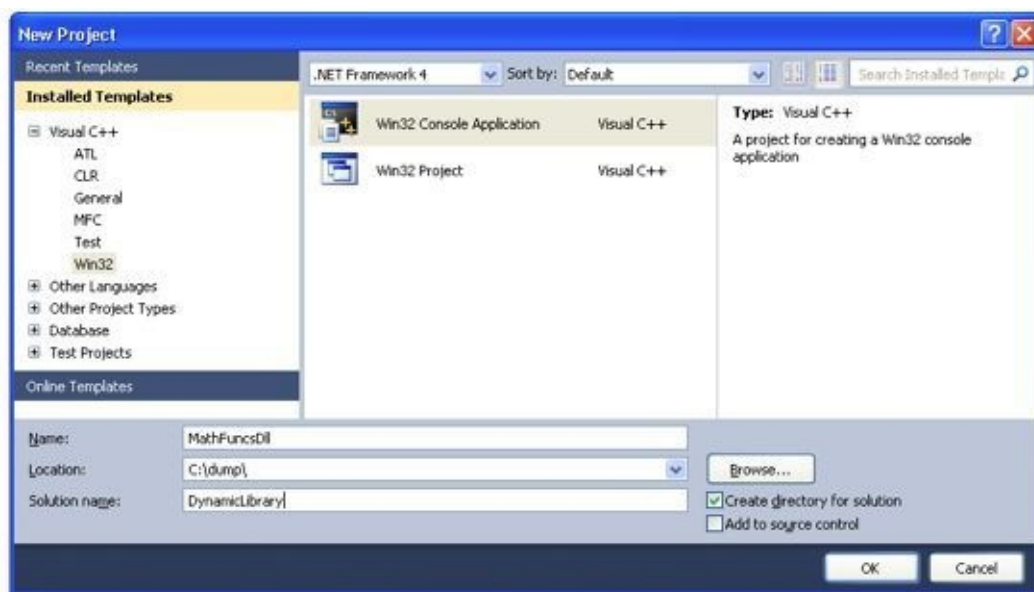
```

2.4 Creating a dynamic link library (DLL)

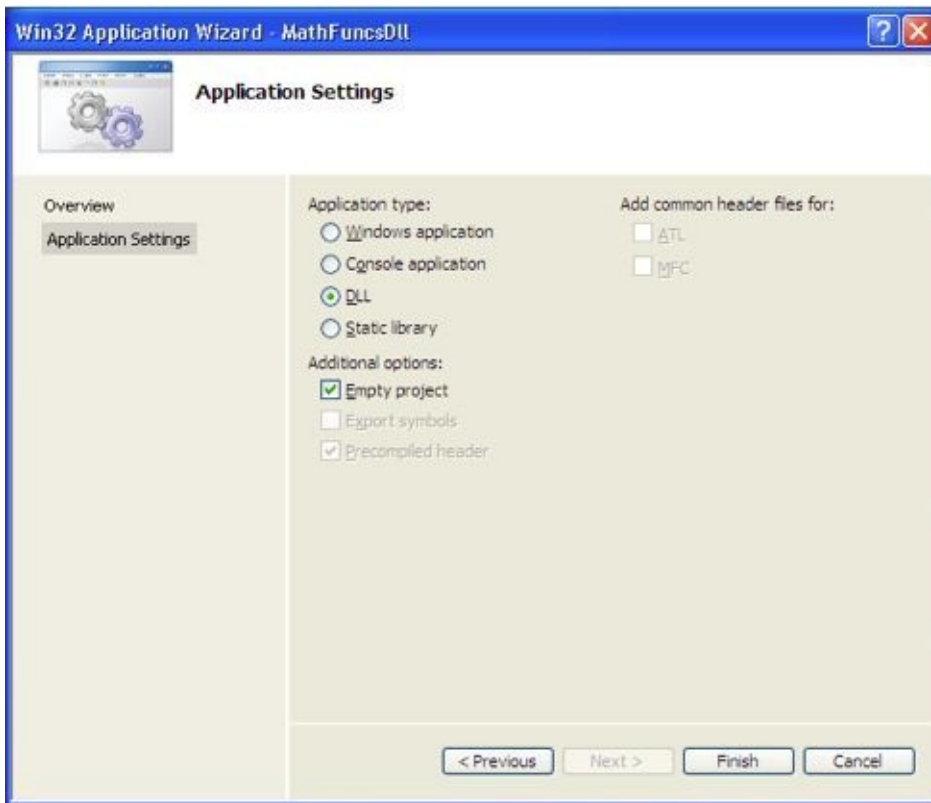
A DLL is a code or data library that can be used by other applications or libraries. For example, common methods for logging, processing or working with user interface controls can be made available in libraries so that several applications can use the same functionality. This not only reduces the need to create the same code multiple times, but also ensures commonality (the same dialog is used across different applications). This section describes how to create your own DLL and utilize its routines from an application created elsewhere.

Create a new dll project

From the File menu, select New and then Project. In the Project types pane, under Visual C++, select Win32 > Win32 Console Application. In the Name field choose a name for the project, such as MathFuncsDll. And in the Solution name field, choose a name for the solution, such as DynamicLibrary. This initial creation is summarized by the following screenshot:



Now press OK to start the Win32 application wizard. Press Next and in the Application Settings page set DLL as the Application type if available or Console application if DLL is not available. Some versions of Visual Studio do not support creating a DLL project using wizards. (If so, it is possible to change this later to make your project compile into a DLL.) In Additional options select Empty project. This is summarized as follows:



Press Finish to create the project.

Add class(es) to the dynamic link library

Select the Project menu > Add New Item and in the Categories pane that appears, under the Visual C++ section of Installed Templates, select Code. Select Header File (.h) and choose a name for the header file, such as MathFuncsDll.h:

Then press Add. A blank file will be displayed. Paste in the example code:

```
// MathFuncsDll.h  
namespace MathFuncs  
{  
class MyMathFuncs  
{  
public:  
// Returns a + b  
static __declspec(dllexport) double Add(double a, double b);  
// Returns a - b
```

```

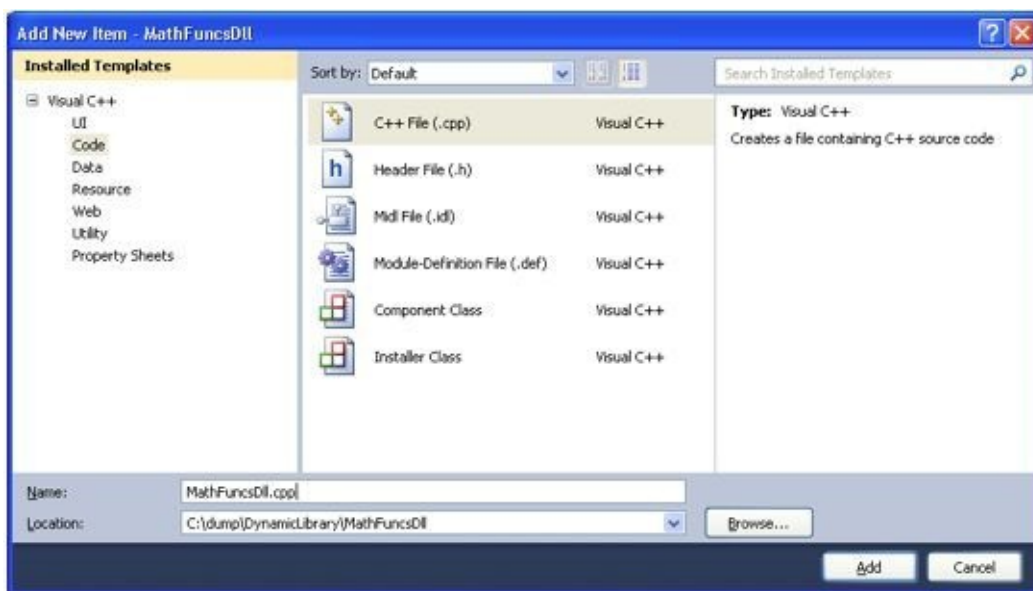
static __declspec(dllexport) double Subtract(double a, double b);
// Returns a * b
static __declspec(dllexport) double Multiply(double a, double b);
// Returns a / b
// Throws DivideByZeroException if b is 0
static __declspec(dllexport) double Divide(double a, double b);
};
}

```

Note the **__declspec(dllexport)** modifiers in the method declarations above. These modifiers enable the method to be exported by the DLL so they can be used by other applications.

Create the source code for the class(es)

From the Project menu, select Add New Item. The Add New Item dialog will be displayed. From the Categories pane, under Visual C++, select Code. Select C++ File (.cpp). Choose a name for the source file, such as MathFuncsDll.cpp.



Press Add. A blank .cpp file will be displayed. Paste in this example code:

```

// MathFuncsDll.cpp
// compile with: /EHsc /LD

```

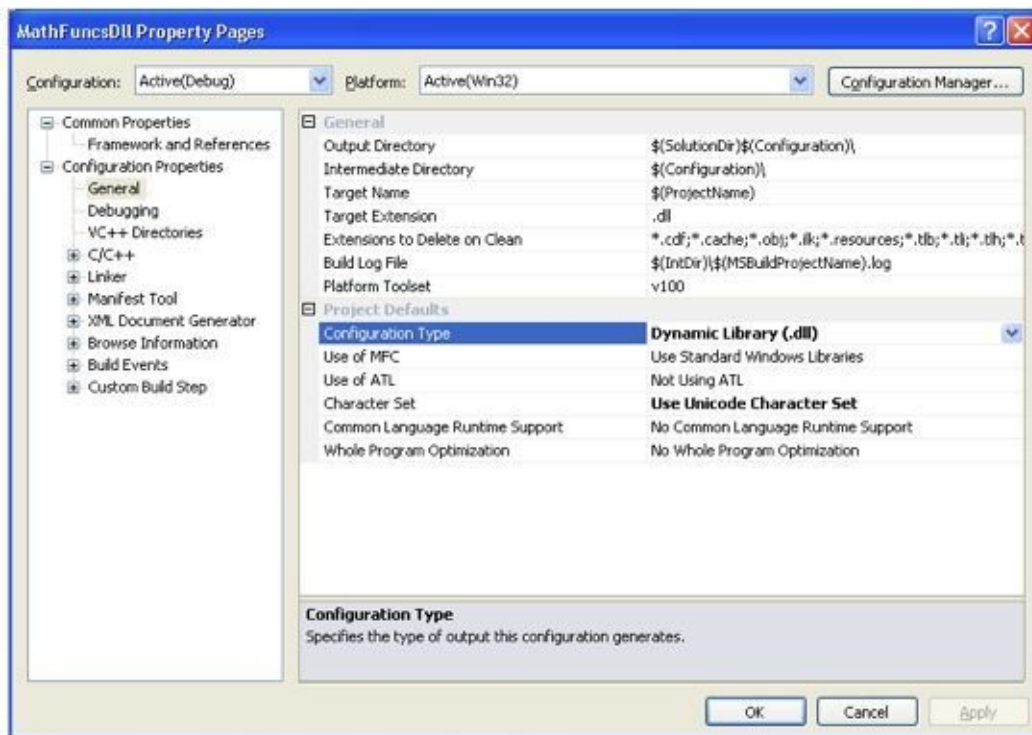
```

#include "MathFuncsDll.h"
#include <stdexcept>
using namespace std;
namespace MathFuncs
{
double MyMathFuncs::Add(double a, double b)
{
return a + b;
}
double MyMathFuncs::Subtract(double a, double b)
{
return a - b;
}
double MyMathFuncs::Multiply(double a, double b)
{
return a * b;
}
double MyMathFuncs::Divide(double a, double b)
{
if (b == 0)
{
throw new invalid_argument("b cannot be zero!");
}
return a / b;
}
}

```

Build the project into a DLL

From the Project menu, select MathFuncsDll Properties, and under Configuration Properties, select General. In the right pane, under Project Defaults, change the Configuration Type to Dynamic Library (.dll).



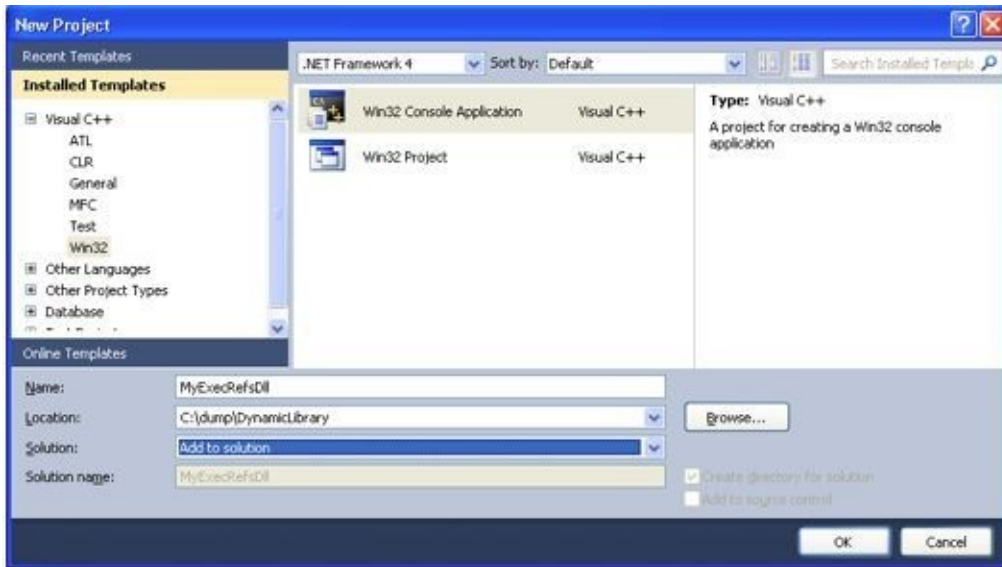
Press OK to save the changes.

Compile the DLL

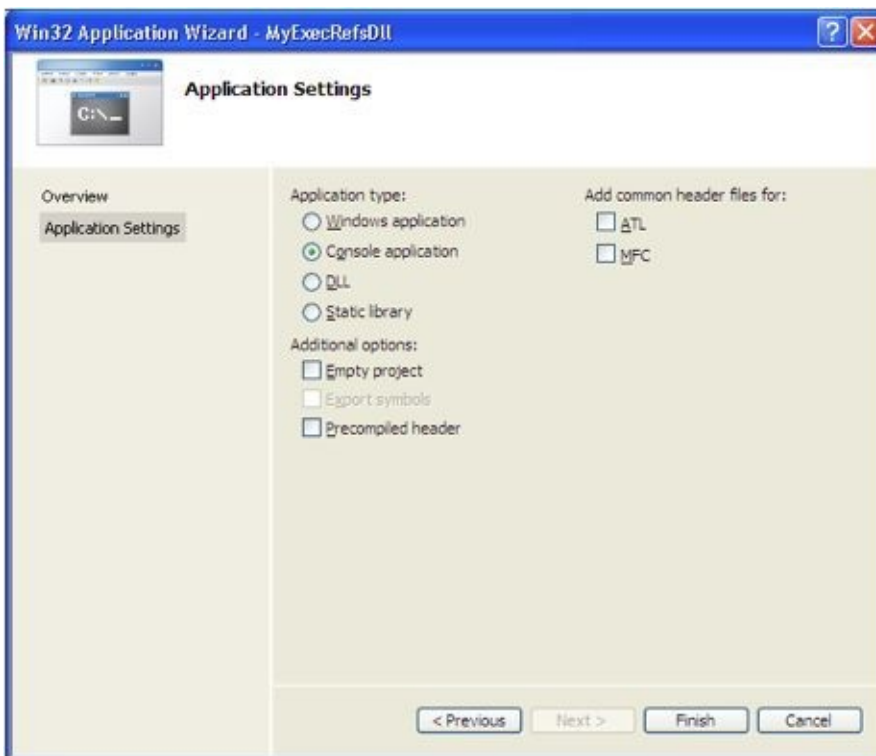
Select Build Solution from the Build menu. This creates a DLL that can be used by other programs.

Create a separate application that references the DLL

Now we will demonstrate how to reference the DLL we created from other applications. From the File menu, select New and then Project. From the Project types pane, under Visual C++, select Win32. In the Templates pane, select Win32 Console Application. Choose a name for the project, such as MyExecRefsDll, and enter it in the Name field. Next to Solution, select Add to Solution from the drop down list. This will add the new project to the same solution as the dynamic link library. The summary is shown in the following screenshot:



Press OK to start the Win32 Application Wizard. From the Overview page of the Win32 Application Wizard, press Next. From the Application Settings page of the Win32 Application Wizard, under Application type, select Console application. From the Application Settings page of the Win32 Application Wizard, under Additional options, deselect Precompiled header:



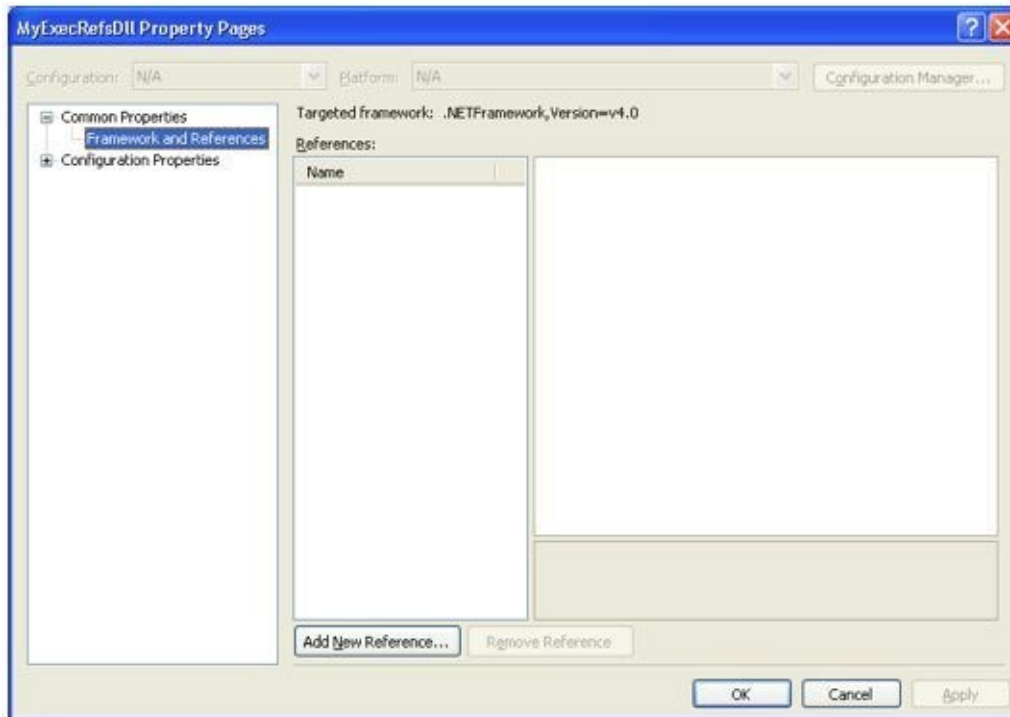
Press Finish to create the project.

Start using the DLL functionality

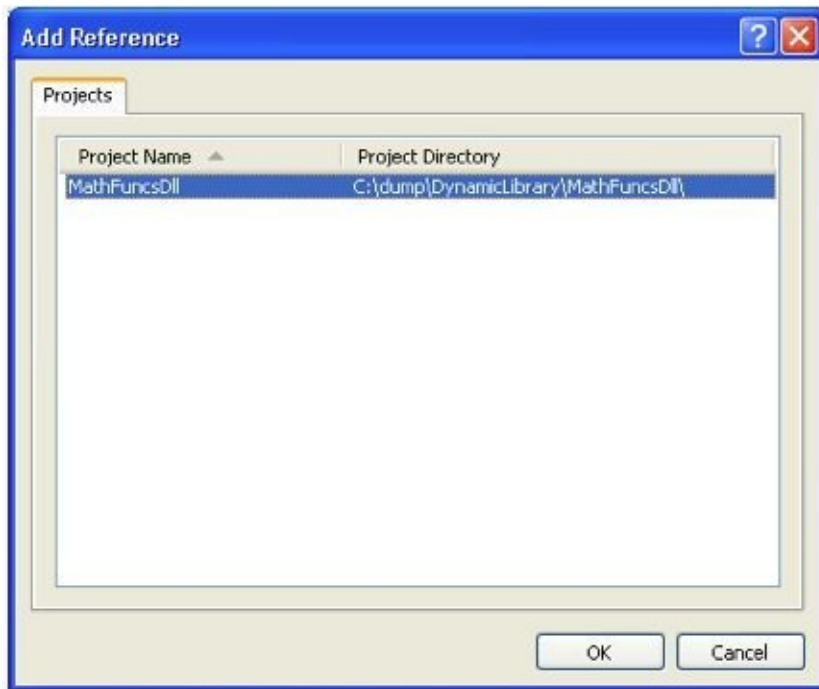
After creating the new Console Application, an empty program is created for you which is

named the same as the name you chose for the project above. In this example, MyExecRefsDll.cpp.

To use the routines you created in the DLL, you reference it as follows: from the Project menu, select References. And from the Property Pages dialog, expand the Common Properties node and select Framework and References.

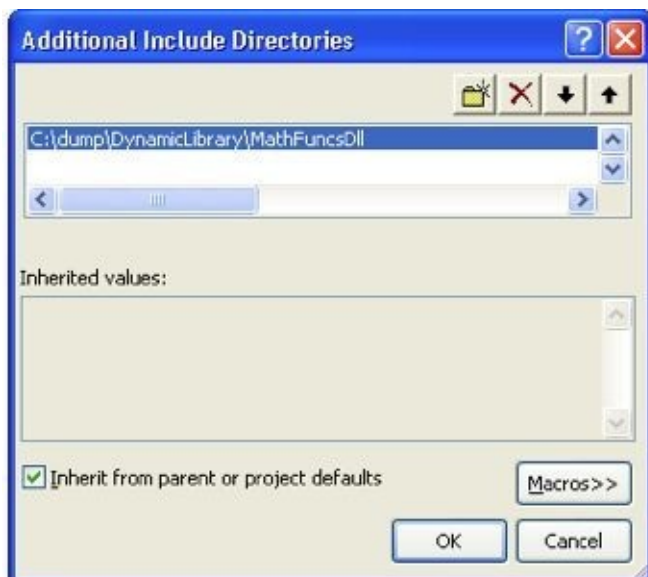


Select the Add New Reference button. This will display the Add Reference dialog, listing all the libraries that you can reference. From the Projects tab, select MathFuncsDll, then select OK:



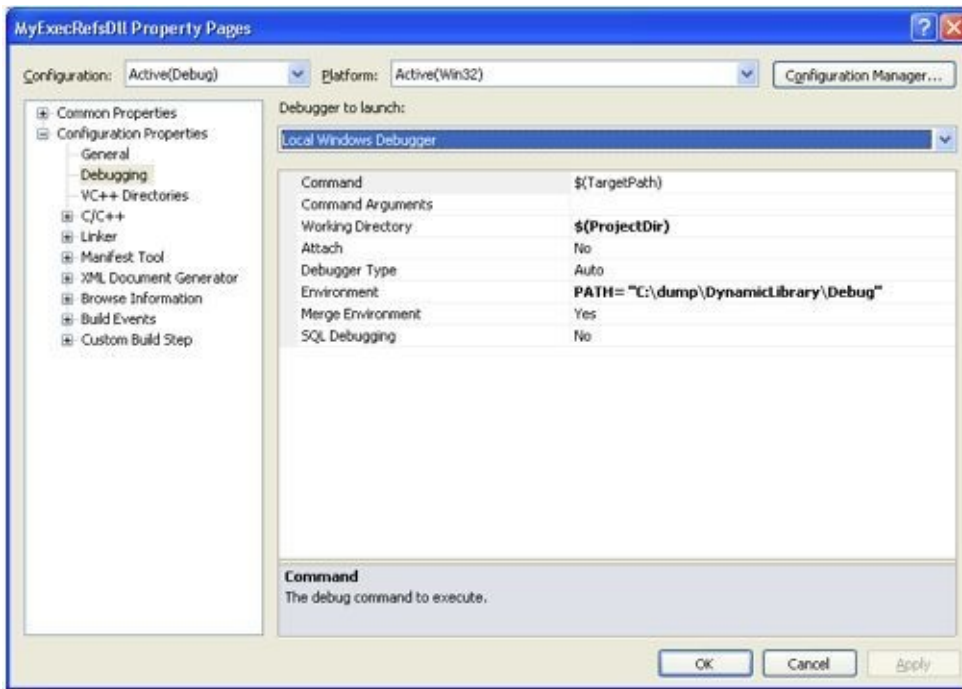
Reference the DLL header files

From the Property Pages dialog, expand the Configuration Properties node, then the C/C++ node, and select General. Next to Additional Include Directories, type in the path to the location of the MathFuncsDll.h header file:



Tell the system where to locate the DLLs at runtime

From the Property Pages dialog, expand the Configuration Properties node and select Debugging. Next to Environment, type in the following: `PATH=` , where the blank is replaced with the actual location of your MathFuncsDll.dll.



Press OK to save all the changes made.

Start using the actual DLL components

Now replace the contents of MyExecRefsDll.cpp with the following code:

```
// MyExecRefsDll.cpp
// compile with: /EHsc /link MathFuncsDll.lib
#include <iostream>
#include "MathFuncsDll.h"
using namespace std;
int main()
{
    double a = 7.4;
    int b = 99;
    cout << "a + b = " << MathFuncs::MyMathFuncs::Add(a, b) << endl;
    cout << "a - b = " << MathFuncs::MyMathFuncs::Subtract(a, b) << endl;
    cout << "a * b = " << MathFuncs::MyMathFuncs::Multiply(a, b) << endl;
    cout << "a / b = " << MathFuncs::MyMathFuncs::Divide(a, b) << endl;
    return 0;
}
```

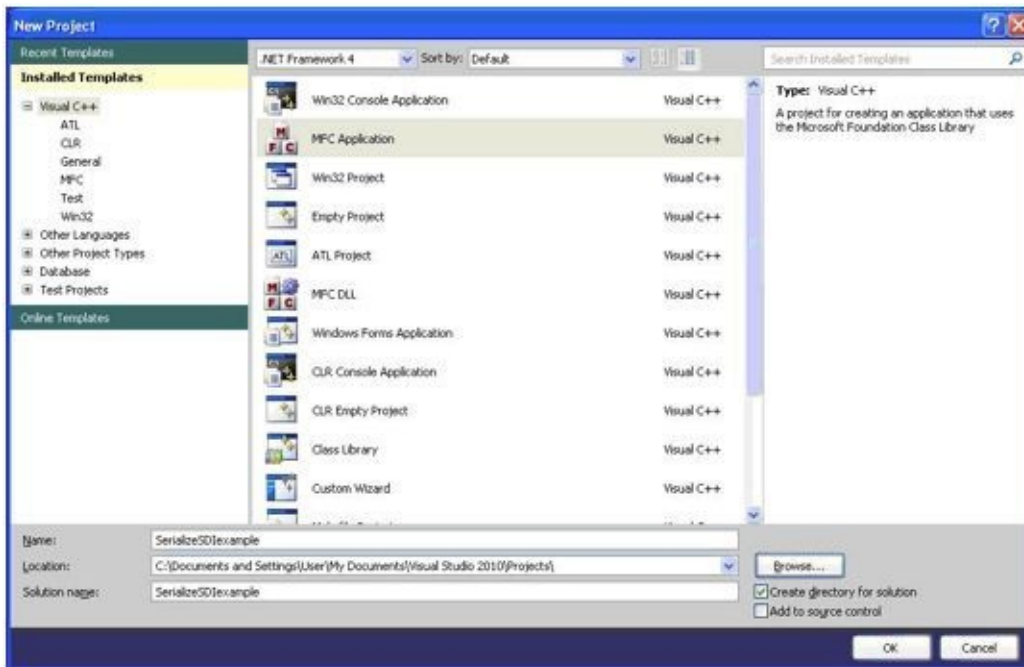
Right-click the MyExecRefsDll project and 'Select as startup project'. Right click on your solution, do a complete clean and rebuild and then run.

2.5 Serializing Your Data

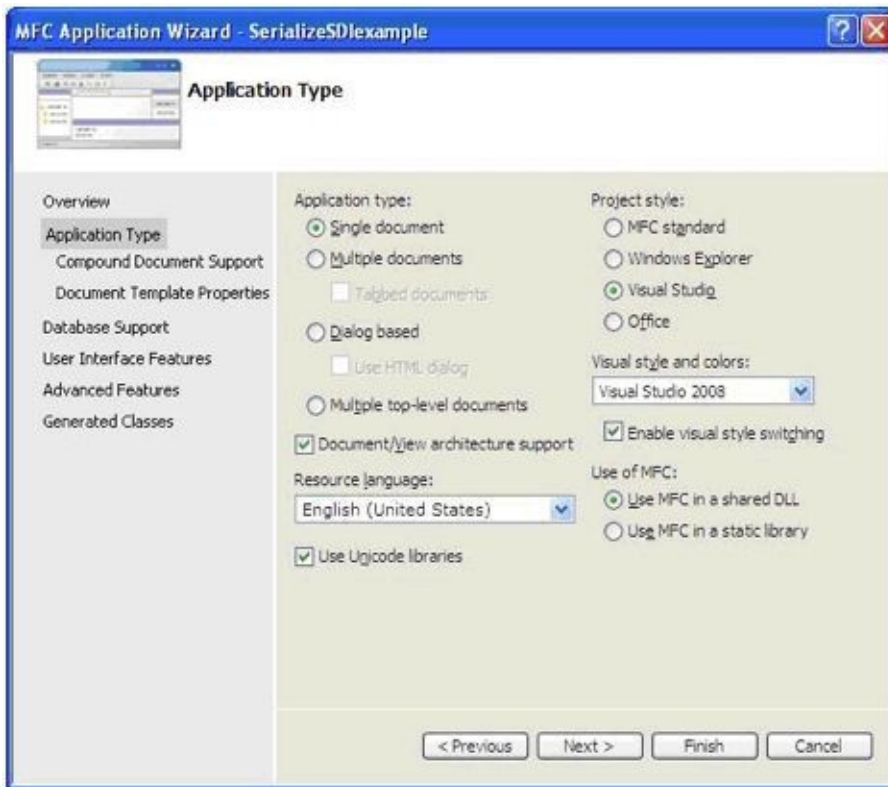
This section shows you how to utilize the built-in file handling capabilities that MFC has to serialize your data. That means reading from the data or writing it to a disk.

Using the MFC document object

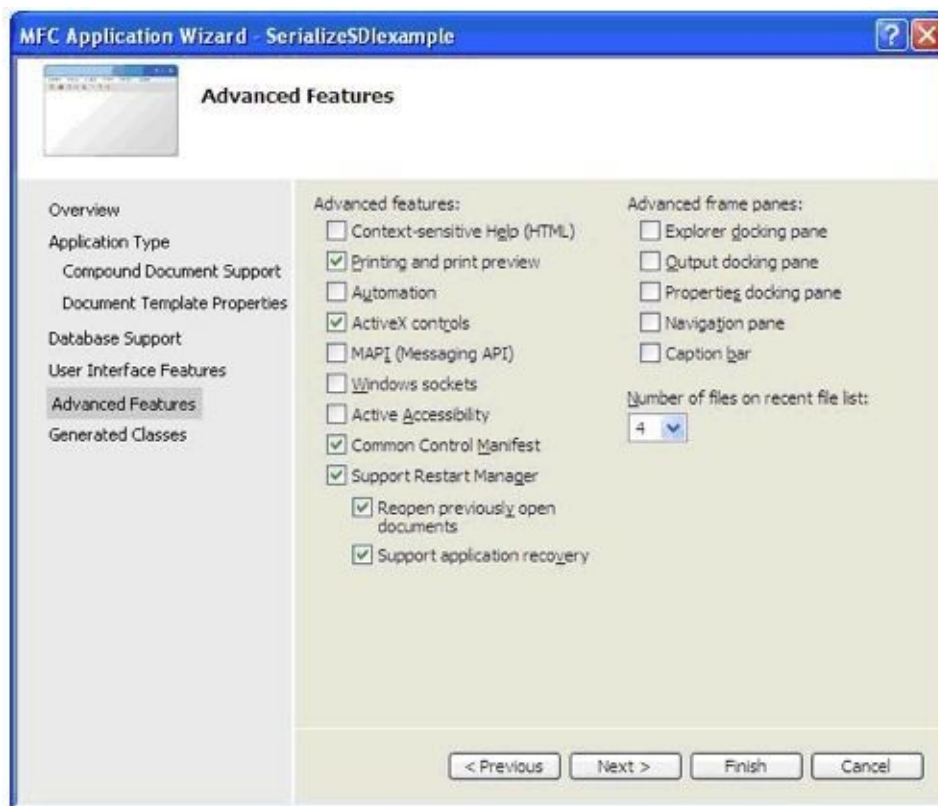
One way is to use the MFC document object to do the serializing for you. This will be demonstrated by creating a simple SDI (single document interface) using the AppWizard. Select File > New > Project and select the Visual C++ > MFC Application as the choice of installed template:



In the Application Type, select Single Document:



In the Advanced Features section, uncheck the Advanced frame panes:



For this example, we will let the program accept and display input from the keyboard.

The first bit of coding is to store the characters the user types in using a **CString** object, **m_strData** in the document class that is created in `SerializeSDIexampleDoc.h`:

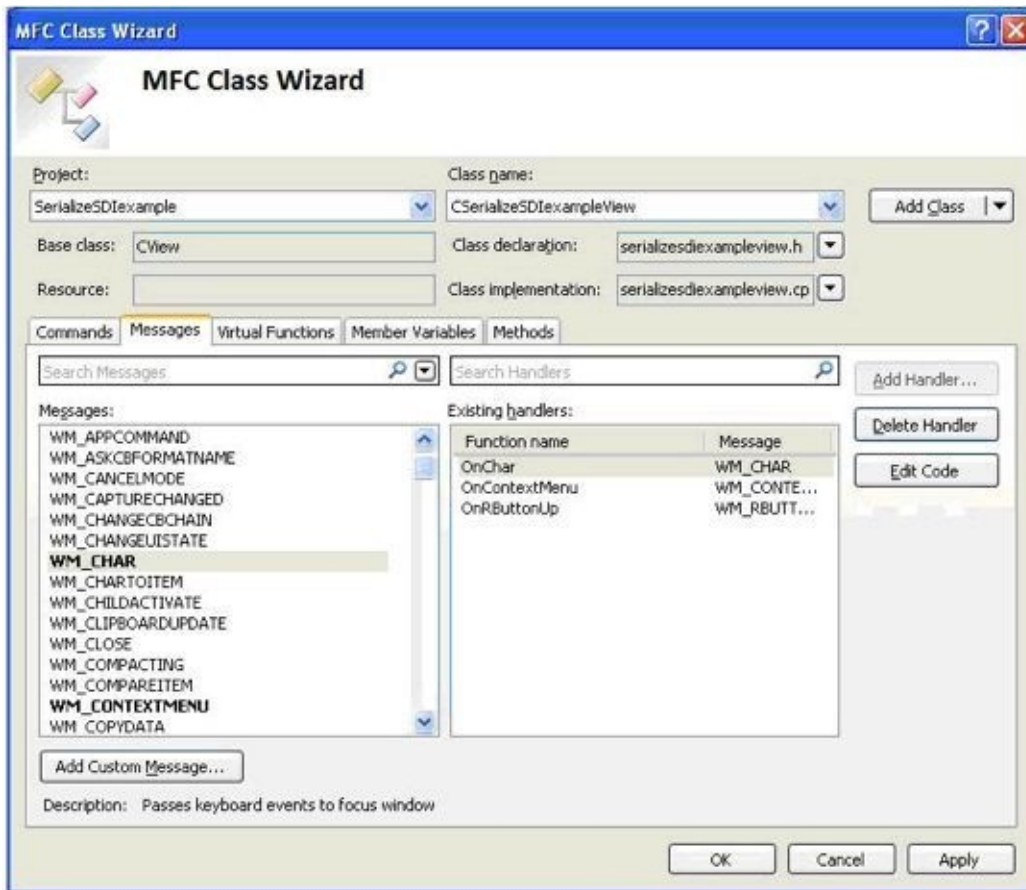
```
class CSerializeSDIexampleDoc : public CDocument
{
protected: // create from serialization only
CSerializeSDIexampleDoc();
DECLARE_DYNCREATE(CSerializeSDIexampleDoc)
// Attributes
public:
CString m_strData;
...
```

And initialise the value of **m_strData** in the document's constructor:

```
CSerializeSDIexampleDoc::CSerializeSDIexampleDoc()
{
    m_strData = "";
}
```

We then need to write some code to handle the **WM_CHAR** Windows messages that are received when the user inputs characters from the keyboard. You can either just write this code yourself, or use the Class Wizard. To use the Class Wizard, right-click the project folder and select Class Wizard...

Ensure the **CSerializeSDIexampleView** class name is selected. Select the Message tab and then select the 'Add Handler...' button.



Then press Apply. Click the Edit Code button so we can write the code to store the characters the user types in.

In addition, this handler checks to see if the user has entered backspace, in which case it deletes the last character by shortening the string data member by 1:

```
void CSerializeSDIexampleView::OnChar(UINT nChar, UINT nRepCnt, UINT  
nFlags)
```

```
{  
// TODO: Add your message handler code here and/or call default  
CSerializeSDIexampleDoc* pDoc = GetDocument();  
ASSERT_VALID(pDoc);  
if ( nChar == 8 )  
{  
int length = pDoc->m_strData.GetLength();  
CString str = pDoc->m_strData.Left( length - 1 );  
pDoc->m_strData = str;  
}
```

```

else
{
CString str;
str.Format( TEXT( "%c" ), nChar );
pDoc->m_strData += str;
}
Invalidate();
CView::OnChar(nChar, nRepCnt, nFlags);
}

```

Invalidate() is used to invalidate the view when the user types a new character. We then add an message handler to display the new text string in **OnDraw()** method inside **CSerializeSDIexampleView**, which is automatically created. Uncomment **CDC*** **/*pDC*/** and add the code to display the text:

```

void CSerializeSDIexampleView::OnDraw(CDC* pDC)
{
CSerializeSDIexampleDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
if (!pDoc) return;
pDC->TextOut( 0, 0, pDoc->m_strData );
}

```

Notice that the **CSerializeSDIexampleDoc** class already has a **Serialize()** method. This is where we serialize the **m_strData** string. Use the **CArchive** reference passed to it in the same as you would use **std::cout** or **std::cin** to read and write to the disk respectively:

```

void CSerializeSDIexampleDoc::Serialize(CArchive& ar)
{
if (ar.IsStoring())
{
ar << m_strData;
}
else

```

```

{
ar >> m_strData;
}
}

```

One last modification we make to the project is to ensure the application remembers when the user has added more data to the **m_strData** object. This means the application will prompt the user with a “Save changes to document” message when the user attempts to exit the application.

In CSerializeSDIexampleView::OnChar indicate that the document has changed by setting the modified flag:

```

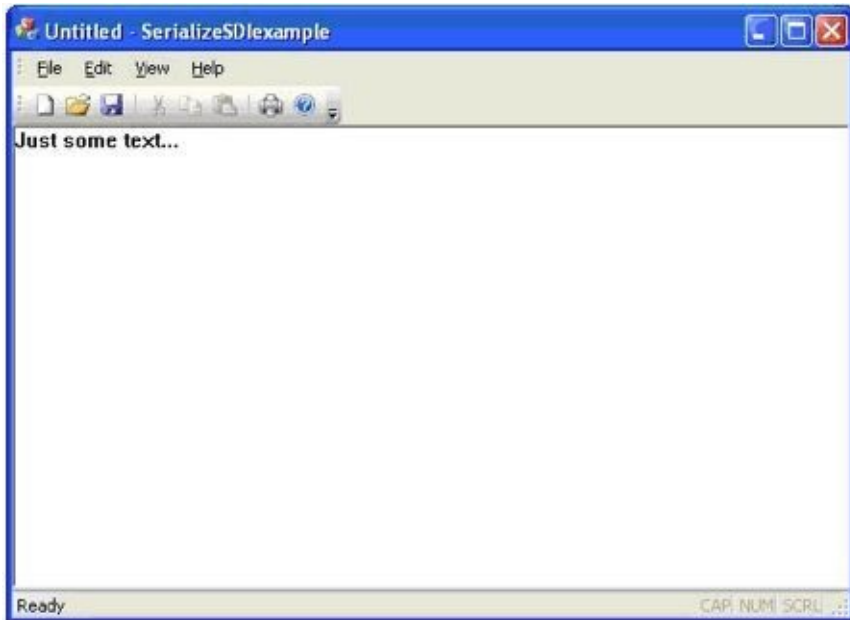
void CSerializeSDIexampleView::OnChar(UINT nChar, UINT nRepCnt, UINT
nFlags)
{
// TODO: Add your message handler code here and/or call default
CSerializeSDIexampleDoc* pDoc = GetDocument();
ASSERT_VALID(pDoc);
if ( nChar == 8 )
{
int length = pDoc->m_strData.GetLength();
CString str = pDoc->m_strData.Left( length - 1 );
pDoc->m_strData = str;
}
else
{
CString str;
str.Format( TEXT( “%c” ), nChar );
pDoc->m_strData += str;
}

Invalidate();
pDoc->SetModifiedFlag();

```

```
CView::OnChar(nChar, nRepCnt, nFlags);  
}
```

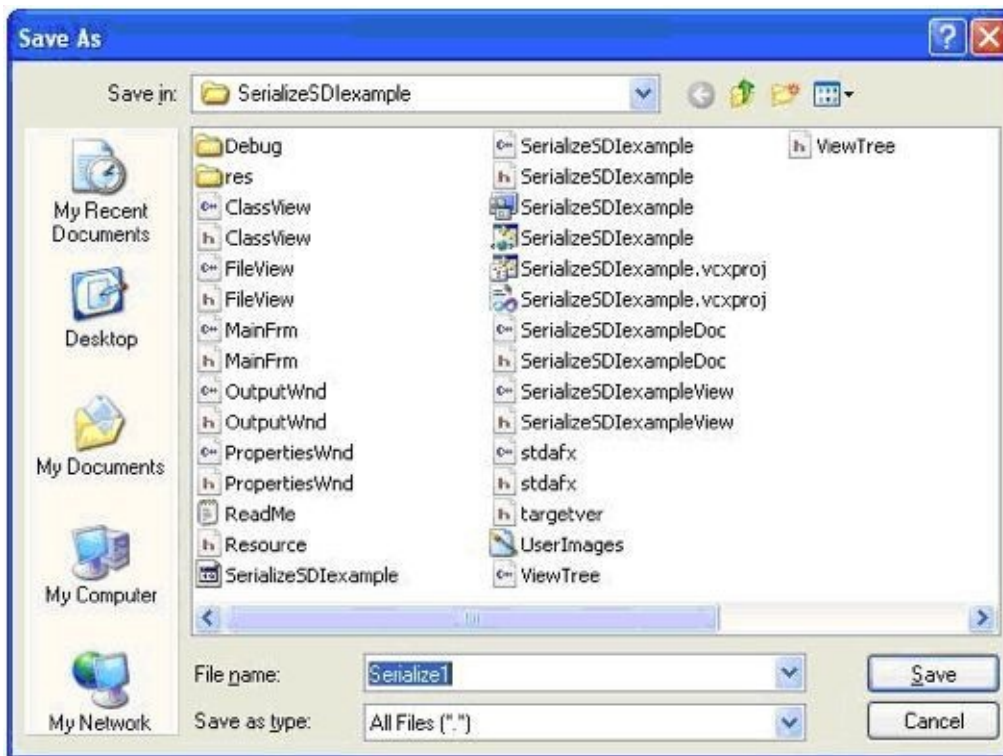
When running the program we can enter characters or delete them using the backspace:



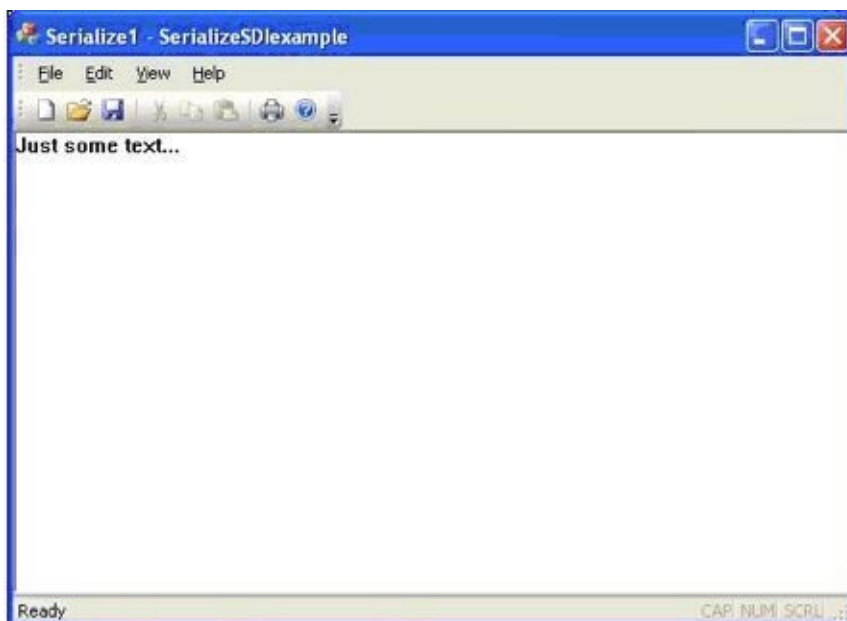
Now try closing the application without saving the text, so that you are prompted if you wish to save:



Give the file a name and then save:



Now restart the application, open the file Serialize1.txt in order to re-open:



Chapter Summary

To summarize this master lesson, you can now:

- Create your own web browser.
- Download files via the File Transfer Protocol.
- Play or capture media files using DirectShow.
- Create a dynamic link library (DLL) and reference it from another application.
- Apply the serialization mechanism provided by MFC to store and retrieve the member variables of your application on a storage medium such as a hard disk.

Chapter 3: Getting Started with the Boost Libraries

3.1 Introduction

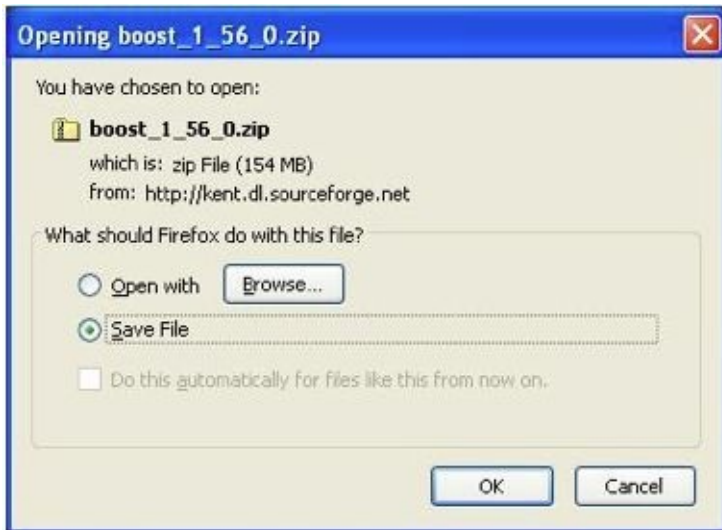
Boost is a powerful set of libraries that can be applied to a wide range of tasks and application domains. If you are seeking to accomplish a C++ programming task in a robust and generic way, there is a strong chance that a Boost library already exists for this purpose. They range from threads and file processing to numerical analysis and graph algorithms. A comprehensive description of any more than a small number of these libraries is beyond the scope of this ebook. The Boost homepage contains extensive documentation on all of the individual components and there are some good books available on the subject.

What we will focus on here is how to really get started. Knowing how to get Boost installed and ready on your chosen development environment and getting a feel for how these libraries are used is often tricky for those who are unfamiliar. It does not help that some of the available documentation can be sketchy.

So this chapter is an in-depth guide to setting up Boost in a number of common Integrated Development Environments (IDEs) including some usage examples.

3.2 Pre-requisites

Before getting to grips with using Boost, first make sure you have the necessary pre-requisites described in this chapter. If you do not already have Boost, download and extract the Boost zip file to a destination of your choice. Official releases of Boost may be obtained from <http://www.boost.org/users/download/>



Generating compiled libraries.

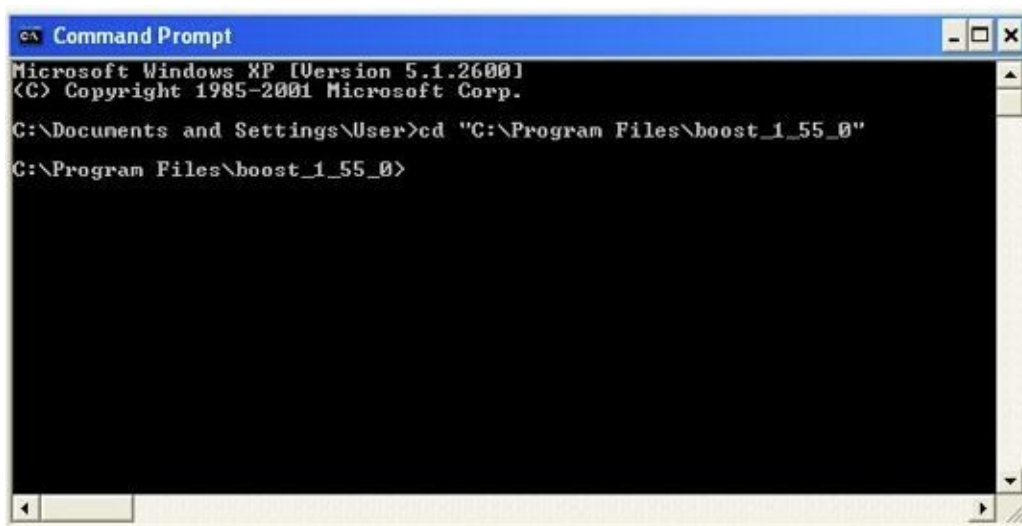
As stated on the ‘Getting Started on Windows’ Boost web page, most Boost libraries are of the header-based type. In other words, once they are installed to your computer they require no further compilation or intervention to make them work. To use such libraries, you need only add the correct includes like this:

```
#include <boost/lambda/lambda.hpp>
```

There exist other Boost libraries that require additional compilation to use them, such as threads, file systems, regular expressions and the unit test framework. The following sections describe how to generate compiled libraries for the following Windows-based C++ compiler environments: Visual C++, MinGW and CygWin. Follow the instructions that are appropriate to your environment.

Generating compiled libraries for Visual Studio

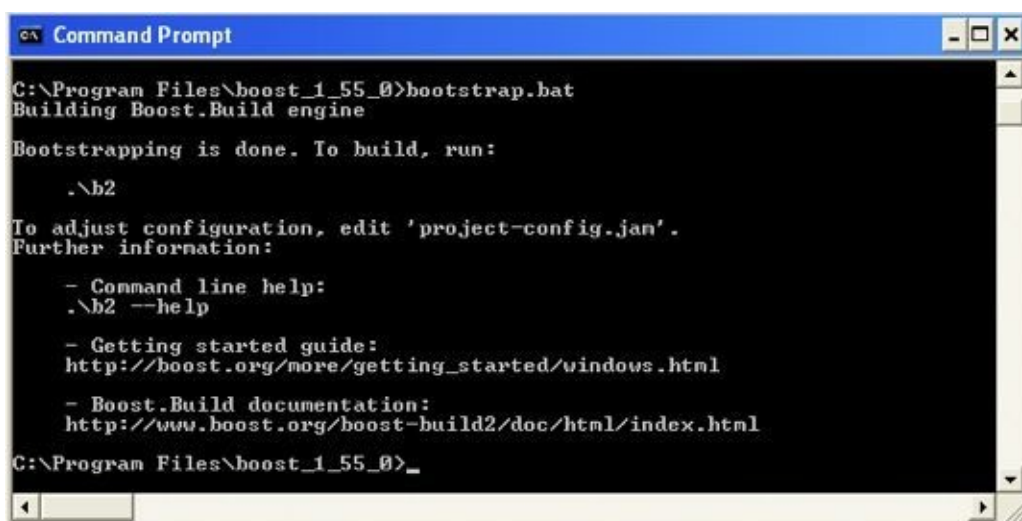
Open a command prompt and navigate to the location of your boost root folder. Enclose the path in quotation marks if it contains white spaces:



```
Command Prompt
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\User>cd "C:\Program Files\boost_1_55_0"
C:\Program Files\boost_1_55_0>
```

From the command prompt run the bootstrap command **bootstrap.bat**:



```
Command Prompt

C:\Program Files\boost_1_55_0>bootstrap.bat
Building Boost.Build engine

Bootstrapping is done. To build, run:

    .\b2

To adjust configuration, edit 'project-config.jan'.
Further information:

    - Command line help:
      .\b2 --help

    - Getting started guide:
      http://boost.org/more/getting_started/windows.html

    - Boost.Build documentation:
      http://www.boost.org/boost-build2/doc/html/index.html

C:\Program Files\boost_1_55_0>_
```

Run `.\b2` from the command prompt. This can take quite some time to finish:

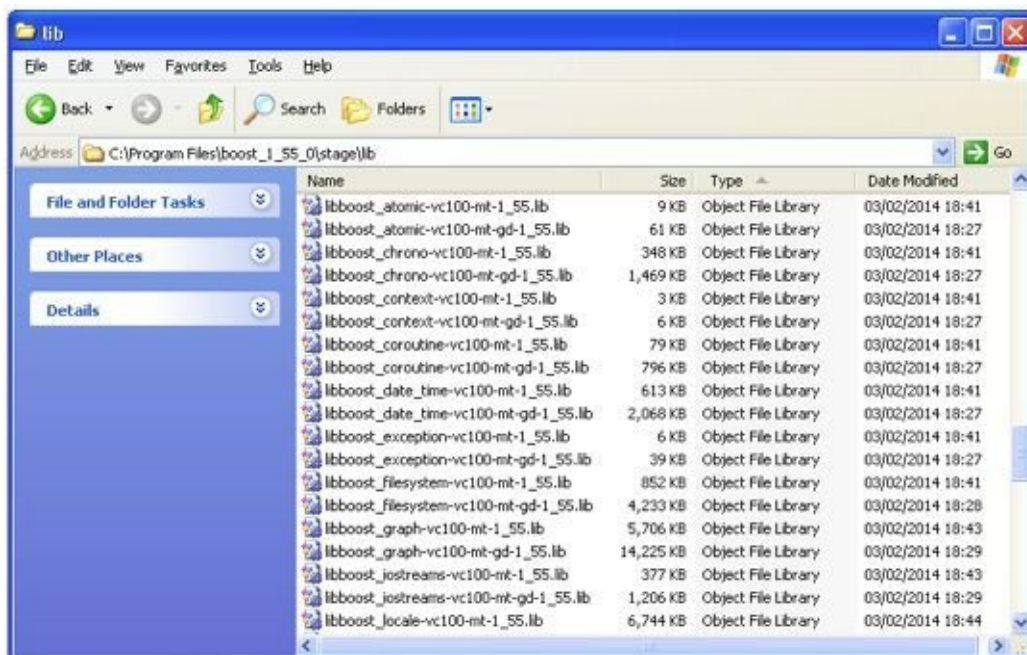
```
ca Command Prompt
C:\Program Files\boost_1_55_0>.b2
link.jan: No such file or directory

Building the Boost C++ Libraries.

Performing configuration checks

- 32-bit : yes (cached)
- arm : no (cached)
- mips1 : no (cached)
- power : no (cached)
- sparc : no (cached)
- x86 : yes (cached)
- has_icu builds : no (cached)
warning: Graph library does not contain MPI-based parallel components.
note: to enable them, add "using mpi;" to your user-config.jam
- zlib : no (cached)
- iconv (libc) : no (cached)
- iconv (separate) : no (cached)
- icu : no (cached)
```

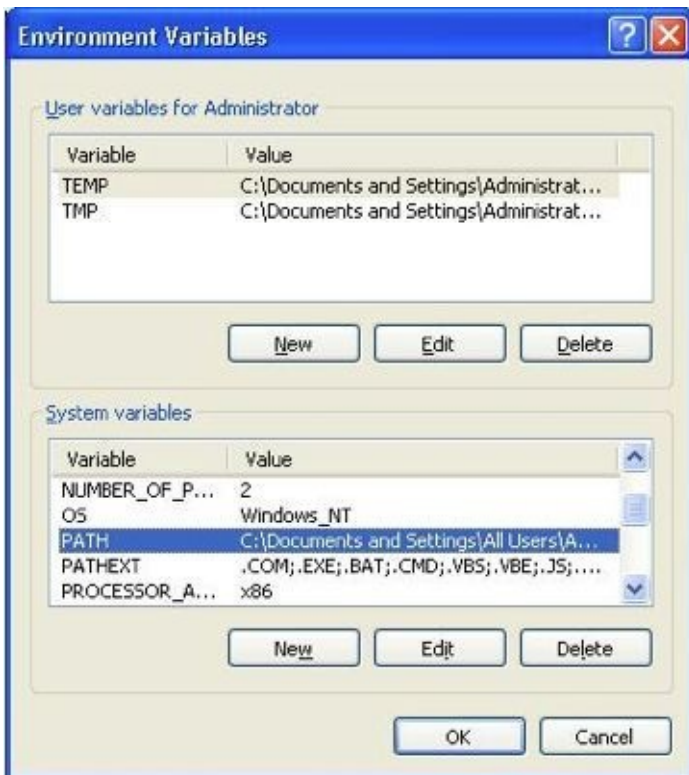
And that's all there is to it. The compiled libraries that Visual Studio uses get installed to the <root>/stage/lib directory:



Generating compiled libraries for MinGW

MinGW (Minimalist GNU for Windows) is a free and open source alternative for developing Microsoft Windows applications. It contains the GNU Compiler Collection (GCC) and various Windows header files and static libraries to enable the use of the Windows API.

Get started by setting the PATH environment variable by right-clicking My Computer and selecting Properties, followed by the Advanced tab, followed by the Environment Variables button. In the System Variables section select the PATH environment variable and then select the Edit button:



In the Edit control that appears, include the setting for including MinGW by appending “C:\mingw\bin” followed by a semi-colon as shown:



To generate the Boost compiled libraries using MinGW, open a command prompt, navigate to the Boost root directory and run the command **bootstrap.bat mingw:**

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>cd C:\Program Files\boost_1_56_0
C:\Program Files\boost_1_56_0>bootstrap.bat mingw
Building Boost.Build engine
Bootstrapping is done. To build, run:

    .\b2

To adjust configuration, edit 'project-config.jan'.
Further information:

- Command line help:
  .\b2 --help

- Getting started guide:
  http://boost.org/more/getting_started/windows.html

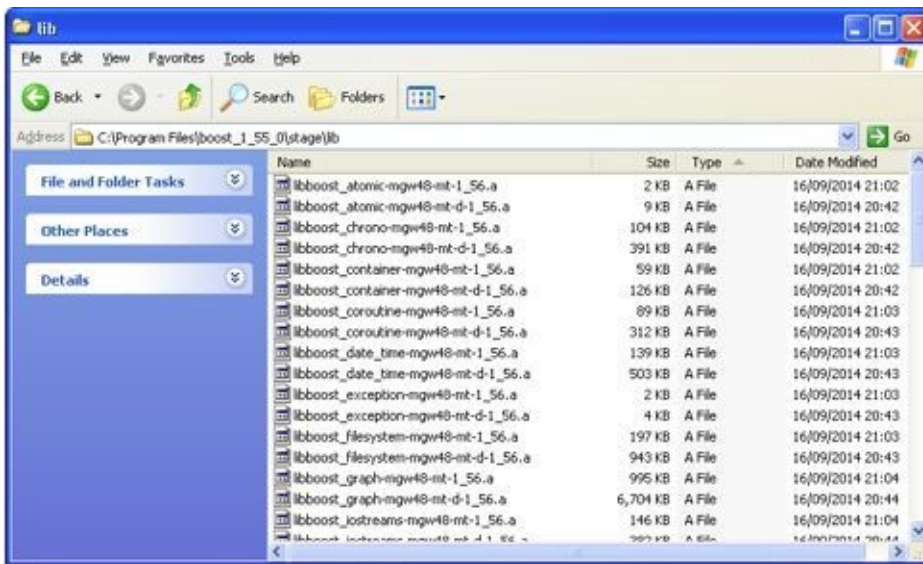
- Boost.Build documentation:
  http://www.boost.org/boost-build2/doc/html/index.html
```

This will generate the two executables needed to compile the Boost libraries, **b2.exe** and **bjam.exe**. Once done execute the command **b2.exe toolset=gcc**

```
C:\WINDOWS\system32\cmd.exe
C:\Program Files\boost_1_56_0>b2 toolset=gcc
Building the Boost C++ Libraries.

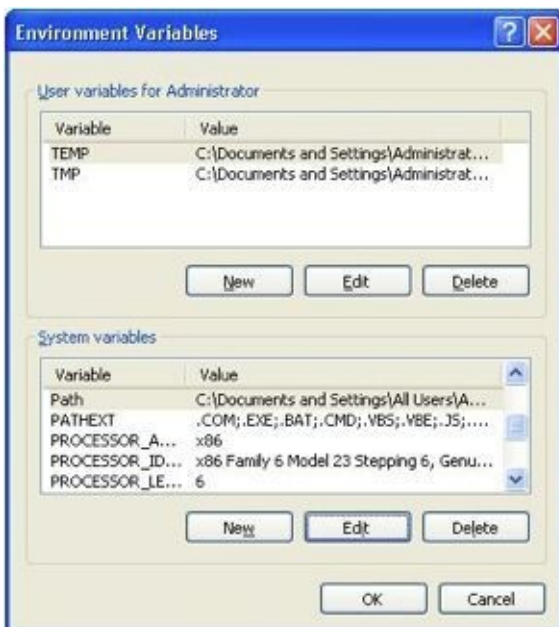
Performing configuration checks
- 32-bit : yes <cached>
- arm : no <cached>
- mips1 : no <cached>
- power : no <cached>
- sparc : no <cached>
- x86 : yes <cached>
- has_icu builds : no <cached>
warning: Graph library does not contain MPI-based parallel components.
note: to enable them, add "using mpi;" to your user-config.jam
- zlib : no <cached>
- iconv <libc> : no <cached>
- iconv <separate> : yes <cached>
- icu : no <cached>
- icu <lib64> : no <cached>
- message-compiler : no <cached>
- compiler-supports-ssse3 : yes <cached>
```

Then the compiled libraries to be used by MinGW are generated as shown:



Generating compiled libraries for CygWin

The advantage of using Cygwin is that makes available many UNIX tools lacking in the Windows operating system. If Cygwin is your choice of compiler, set the Environment Variables so that your IDE can access the Cygwin tool collection. From the Start button, right-click 'My Computer' and select 'Properties'. Select the 'Advanced' tab and then 'Environment Variables'. Select the 'Path' in the 'System Variables' section and then select the 'Edit' button:



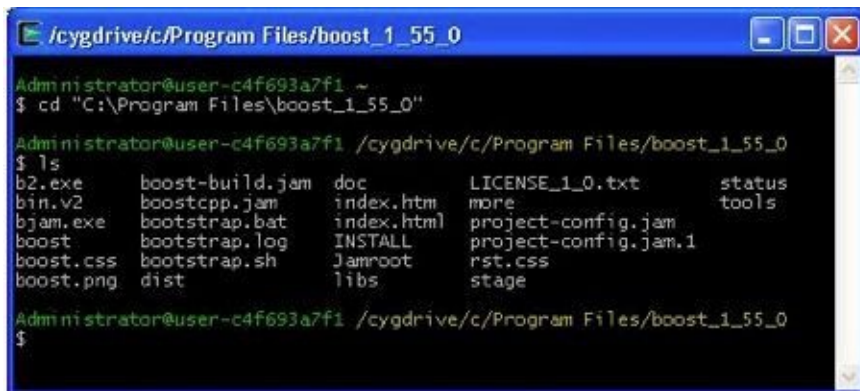
And then set the location of the Cygwin tool set:



To configure Boost to work with Cygwin, open the Cygwin terminal and navigate to the location where your Boost libraries have been installed. For Windows directories with white spaces enclose it in quotes like this:

\$ cd "C:\Program Files\boost_1_55_0"

Check to see that the boost/ folder exists at this location using the ls command:

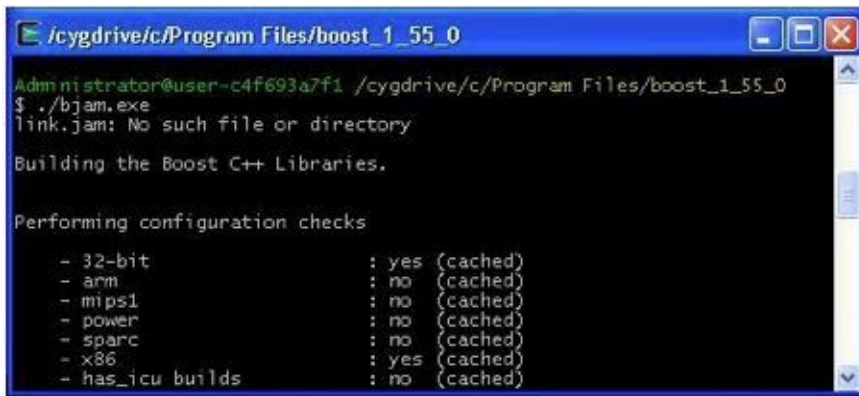


If the boost directory is present, run the bootstrap command:

\$./bootstrap.sh --prefix=boost/

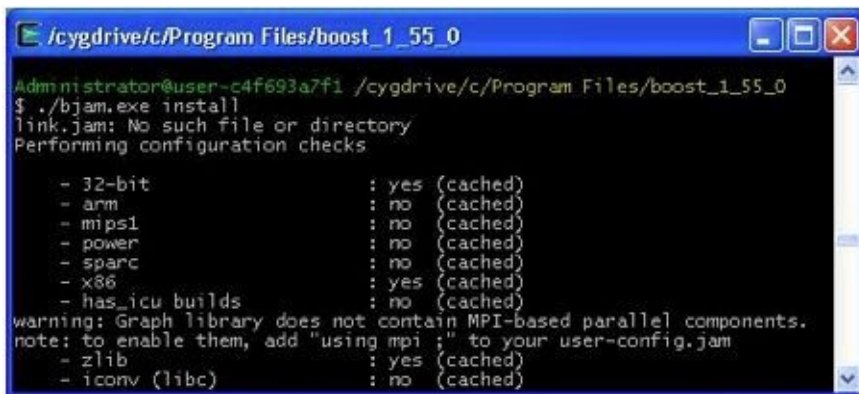


Then run the **\$./bjam.exe** command (this can take a while):



```
/cygdrive/c/Program Files/boost_1_55_0
Administrator@user-c4f693a7f1 /cygdrive/c/Program Files/boost_1_55_0
$ ./bjam.exe
link.jam: No such file or directory
Building the Boost C++ Libraries.
Performing configuration checks
- 32-bit      : yes (cached)
- arm        : no (cached)
- mips1      : no (cached)
- power      : no (cached)
- sparc      : no (cached)
- x86        : yes (cached)
- has_icu builds : no (cached)
```

Now run the `$./bjam install` command (this can take a very long time):



```
/cygdrive/c/Program Files/boost_1_55_0
Administrator@user-c4f693a7f1 /cygdrive/c/Program Files/boost_1_55_0
$ ./bjam.exe install
link.jam: No such file or directory
Performing configuration checks
- 32-bit      : yes (cached)
- arm        : no (cached)
- mips1      : no (cached)
- power      : no (cached)
- sparc      : no (cached)
- x86        : yes (cached)
- has_icu builds : no (cached)
warning: Graph library does not contain MPI-based parallel components.
note: to enable them, add "using mpi;" to your user-config.jam
- zlib       : yes (cached)
- iconv (libc) : no (cached)
```

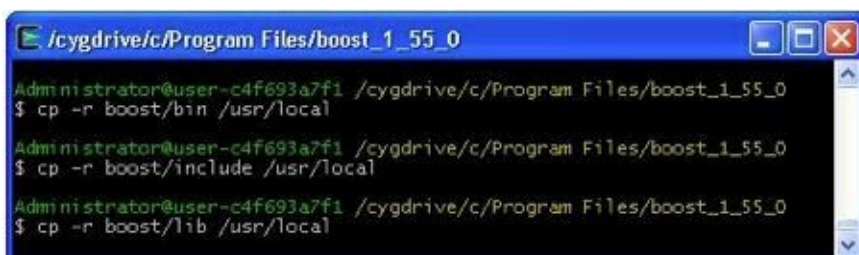
Once this stage has been completed you will find that three additional folders have been created within the boost/ folder, namely **bin**, **include** and **lib**.

Lastly, recursively copy the folders to the `/usr/local/` destination:

`$ cp -r boost/bin /usr/local/`

`$ cp -r boost/include /usr/local/`

`$ cp -r boost/lib /usr/local/`



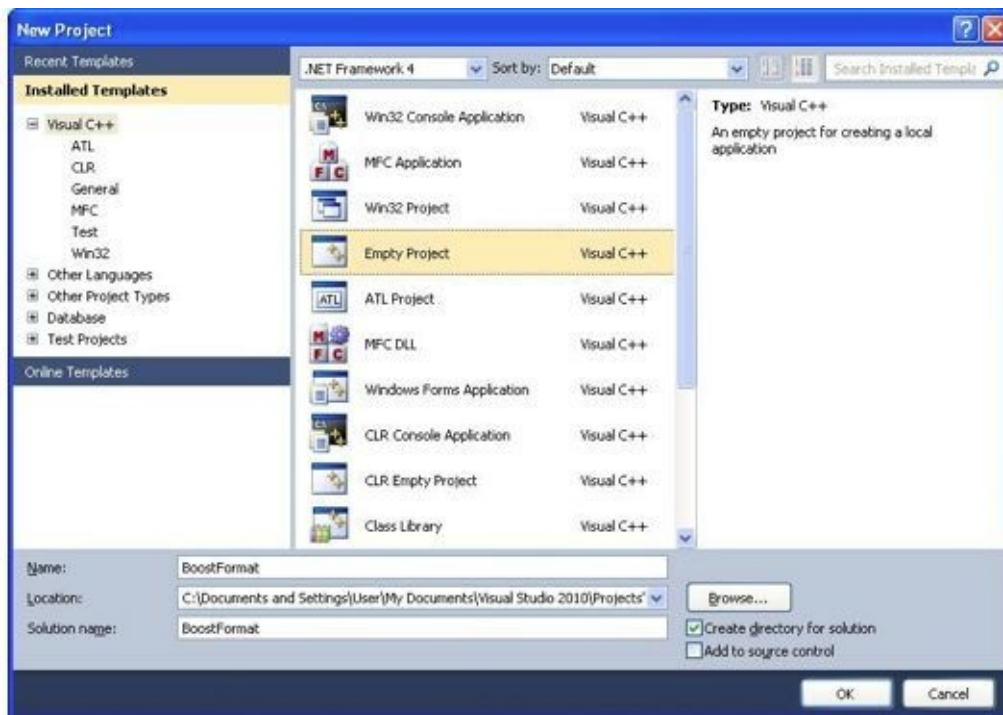
```
/cygdrive/c/Program Files/boost_1_55_0
Administrator@user-c4f693a7f1 /cygdrive/c/Program Files/boost_1_55_0
$ cp -r boost/bin /usr/local
Administrator@user-c4f693a7f1 /cygdrive/c/Program Files/boost_1_55_0
$ cp -r boost/include /usr/local
Administrator@user-c4f693a7f1 /cygdrive/c/Program Files/boost_1_55_0
$ cp -r boost/lib /usr/local
```

3.3 Configuring Boost in Microsoft Visual Studio

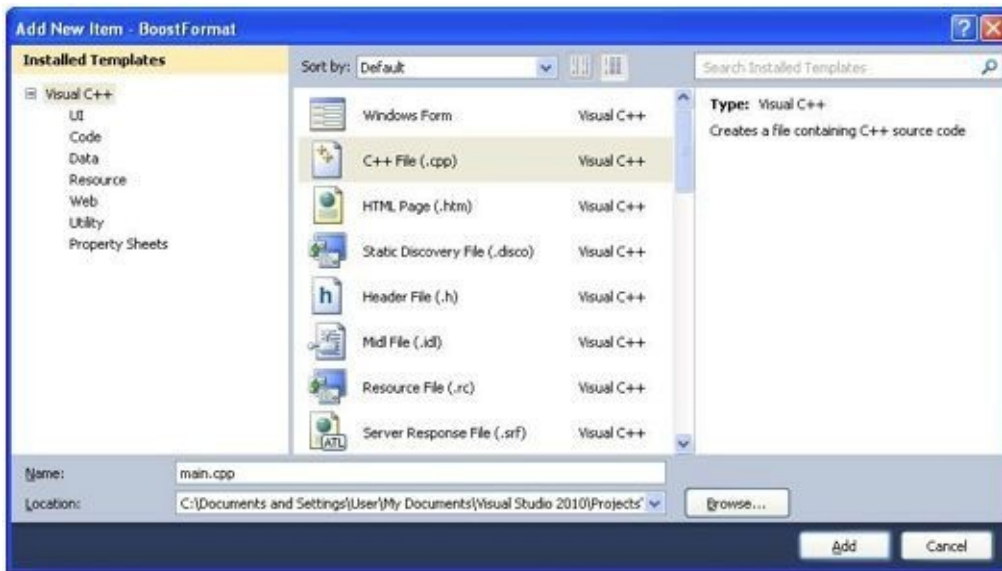
Header-only library example: Boost Format

To get a feel for using Boost in Visual Studio let's start with a library requiring no separate compilation: Boost Format, a type-safe utility for formatting arguments according to a format string. Create an empty project by selecting File > New > Project.

Set the 'Create directory for solution' checkbox and name your project:



Add the code by right-clicking the project folder and selecting Add > New Item. Select C++ File and name it as 'main.cpp':



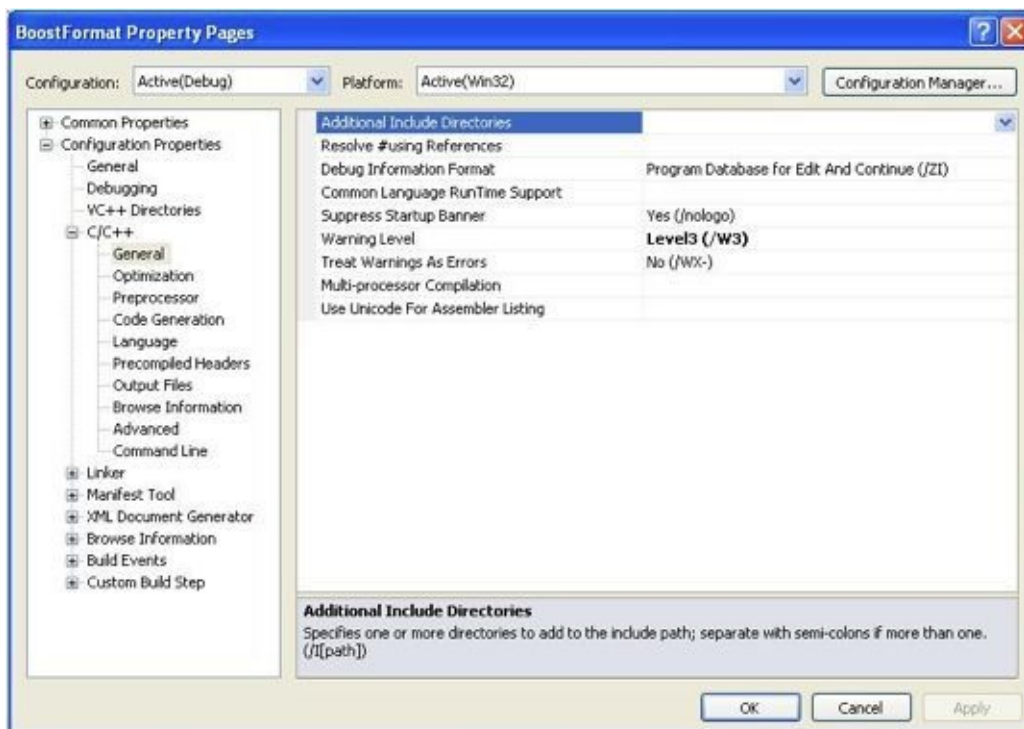
Insert this code snippet into main.cpp as an example of how to format and output an array of hexadecimal values:

```
#include <iostream>
#include <boost/format.hpp>
int main()
{
unsigned int arr[ 5 ] = { 0x05, 0x04, 0xAA, 0x0F, 0x0D };
std::cout << boost::format(“%02X-%02X-%02X-%02X-%02X”)
% arr[0]
% arr[1]
% arr[2]
% arr[3]
% arr[4]
<< endl;
}
```

Notice that build errors will occur if we try to compile the project without having first specified the additional includes required to use the Boost libraries:

fatal error C1083: Cannot open include file: ‘boost/format.hpp’: No such file or directory

Right-click your project folder and select 'Properties'. In the C++ > General tab edit the Additional Include Directories section:



Tell Visual Studio where your Boost include files reside:



And that's all you need to do to use non-compiled Boost libraries. Your program should now successfully compile.

Running the program generates the hexadecimal output:

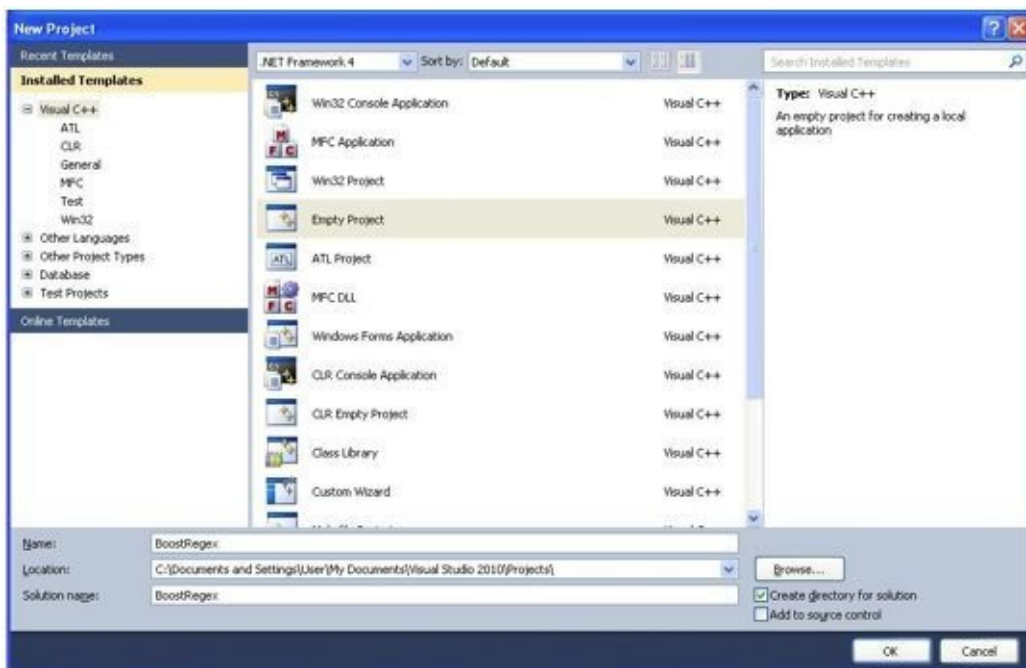


Compiled library example: Boost Regex

A Boost library requiring separate compilation is Boost Regex, a C++ implementation of regular expressions, which are a form of pattern matching often used in text processing.

Select File > New > Project > Visual C++ > Empty Project.

Call it BoostRegex:



Select New > File > C++ File and name it main.cpp.

Into main.cpp copy the following code:

```
#include <boost/regex.hpp>
#include <iostream>
```

```

#include <string>
#include <sstream>

int main()
{
// Regex to match YYYY-MM-DD dates
static const boost::regex date_regex("[0-9][0-9][0-9][0-9]-([1-9]|0[1-9]|1[012])-([1-9]|0-2|[1-9]|3[01])");

std::string input = "2014-08-28";
boost::smatch char_matches;

if ( boost::regex_match( input, char_matches, date_regex ) )
{
std::stringstream ss_match;

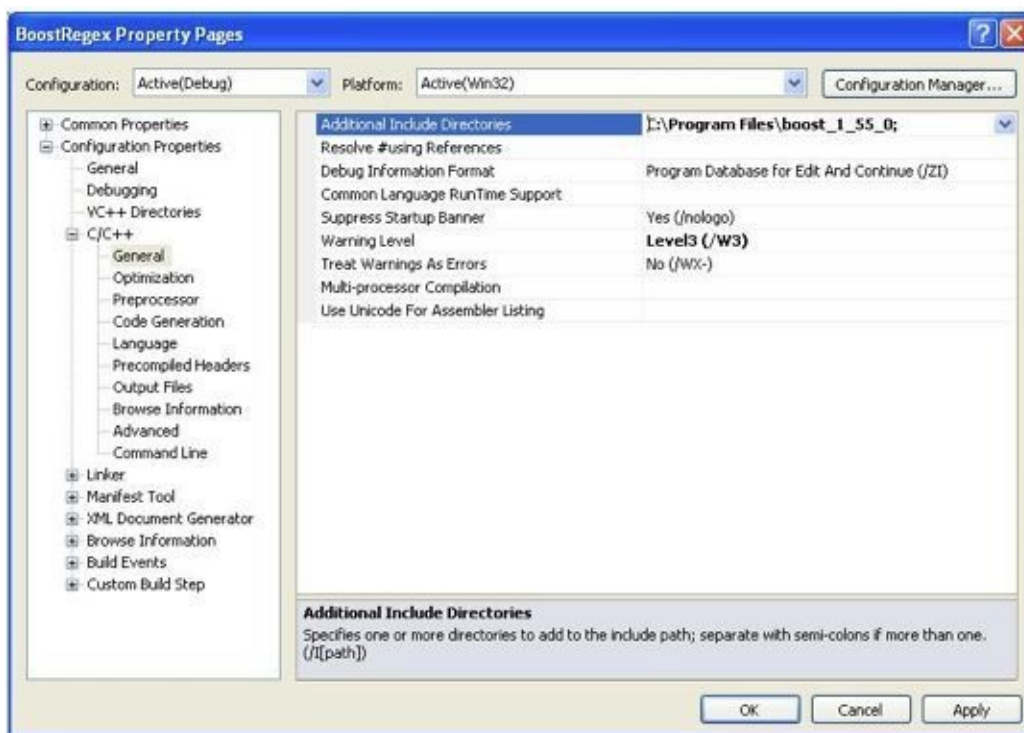
ss_match << char_matches[1]
<< "-"
<< char_matches[2]
<< "-"
<< char_matches[3];
std::cout << ss_match.str();
}
else
{
std::cout << "First example should have matched the regex.\n";
}
}

```

Set the include directory for Boost Regex. Failing to do this will generate the following compilation error:

fatal error C1083: Cannot open include file: 'boost/regex.hpp': No such file or directory

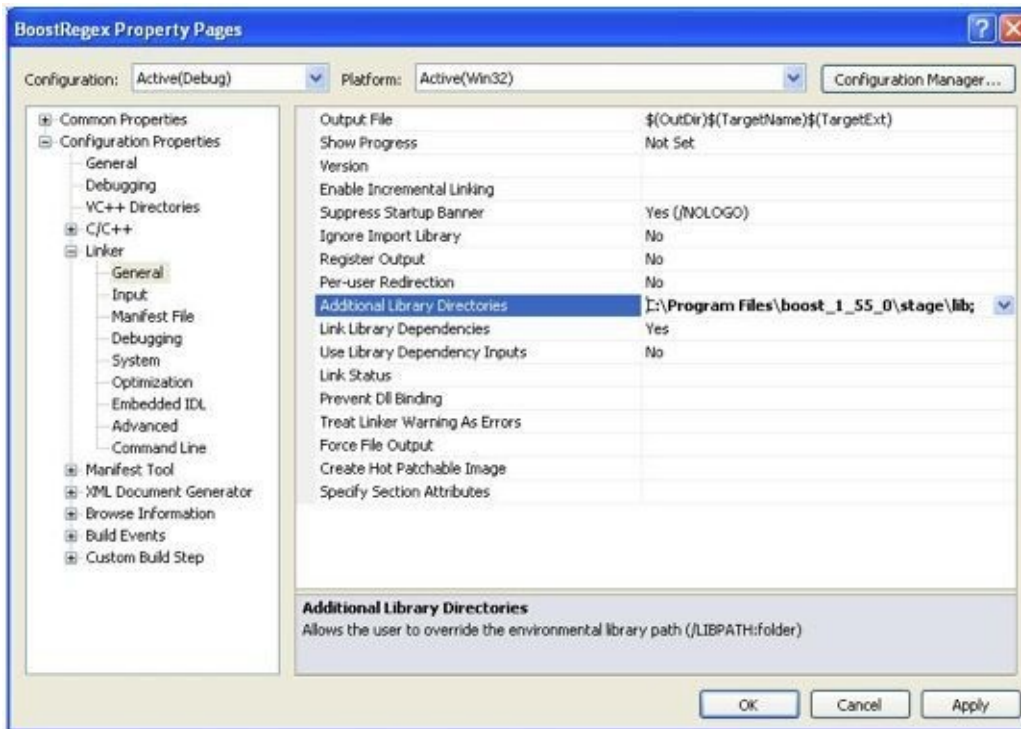
Right-click your project folder and select Properties. In the C/C++ > General tab set the Additional Include Directories field:



Failing to set the location of the additional library directories will generate the following link error when you try to compile your project:

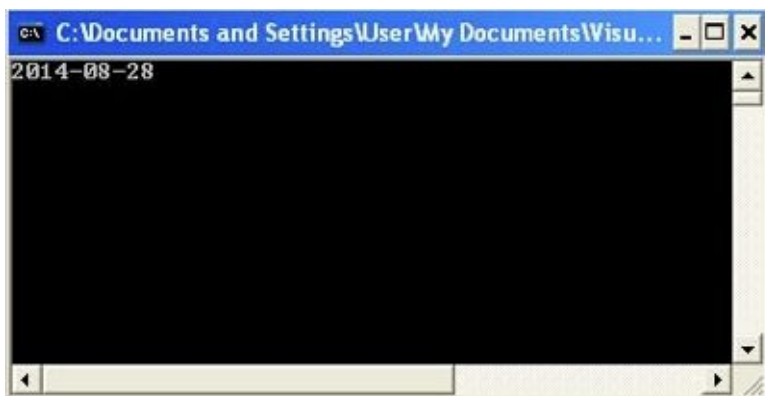
LINK : fatal error LNK1104: cannot open file 'libboost_regex-vc100-mt-gd-1_55.lib'

Right-click your project folder and select Properties.



In the Linker > General tab, set the Additional Library Directories field to tell Visual Studio where to obtain the required library files:

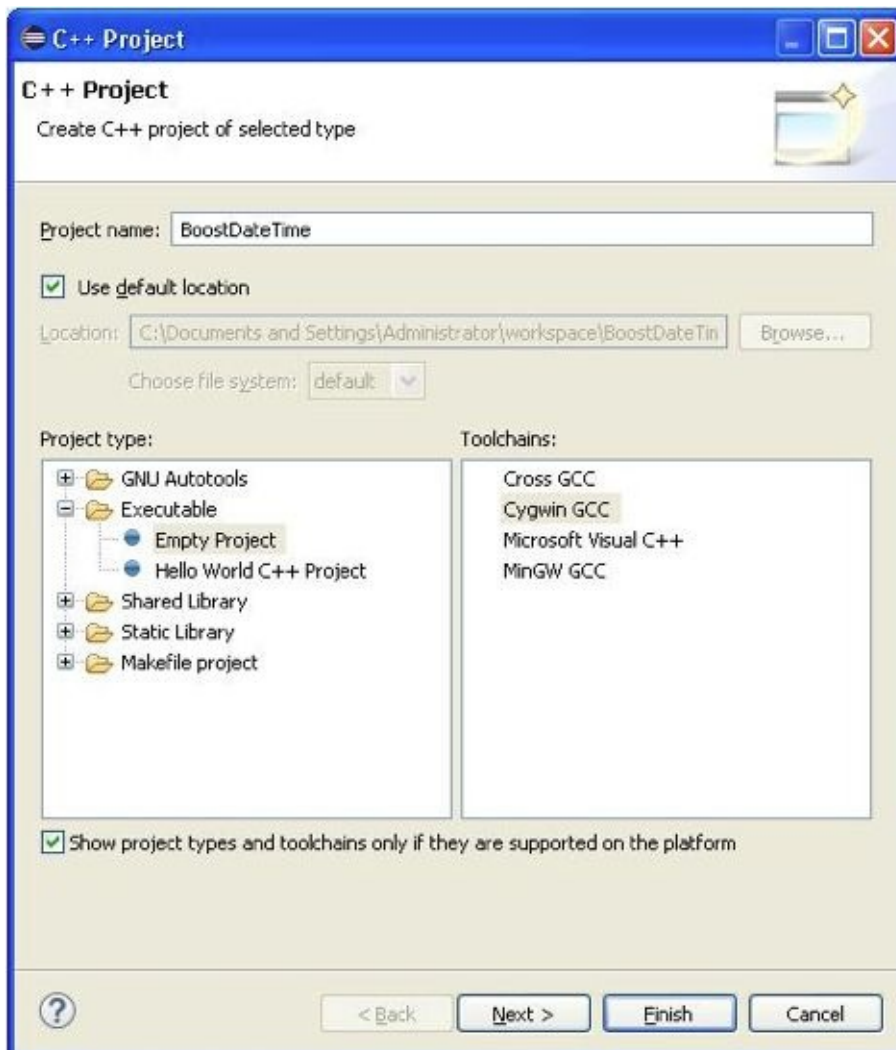
And that's it. Your Boost Regex Visual Studio project should now compile and allow you to run the above example:



3.4 Configuring Boost in Eclipse

Header-only library example: Boost Date Time

Boost Date Time is a highly customizable library for programming with dates and times. Open Eclipse and select New > C++ Project. In the 'C++ Project' dialog that appears, select Empty Project, give the project a name such as 'BoostDateTime', and select the desired toolchain (such as Cygwin):



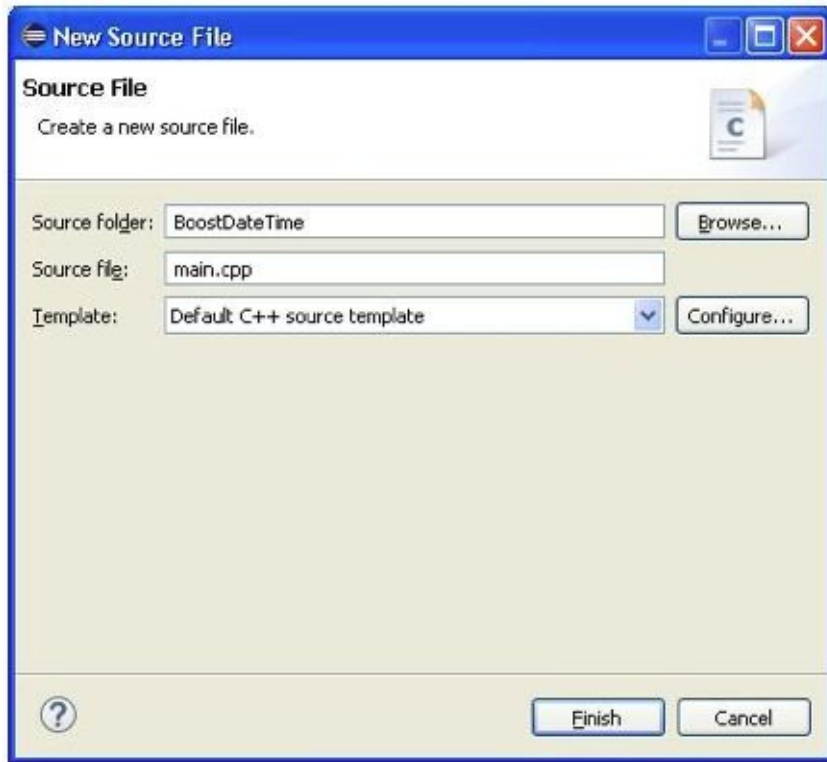
Click Next and make sure the Release and Debug checkboxes are set:



Click Finish.

Now that we have created the project, right click on the 'BoostDateTime' project folder and select New > Source File.

Fill in the fields according to your preferences, and then click Finish:



Overwrite the main.cpp file that gets created with the following code so that we may use this example of a header-only example of Boost Date Time:

```
#include <boost/format.hpp>
```

```
#include <boost/date_time.hpp>
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
const boost::posix_time::ptime now =
```

```
boost::posix_time::second_clock::local_time();
```

```
const boost::wformat f = boost::wformat( L"%02d.%02d.%s %02d:%02d" )
```

```
% now.date().year_month_day().day.as_number()
```

```
% now.date().year_month_day().month.as_number()
```

```
% now.date().year_month_day().year
```

```
% now.time_of_day().hours()
```

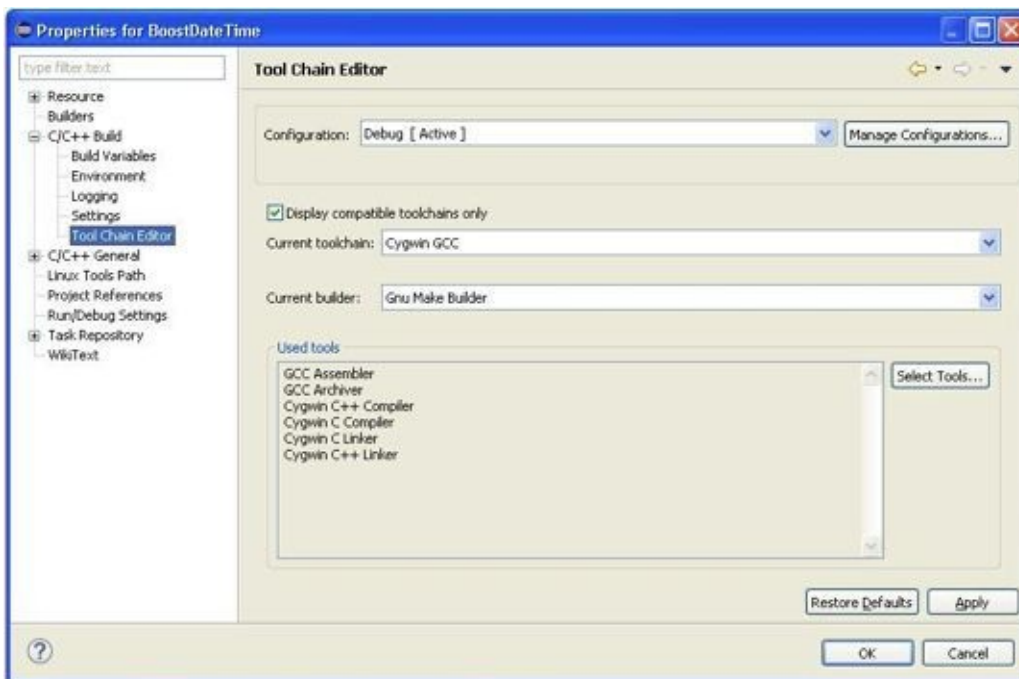
```
% now.time_of_day().minutes();
```

```
std::wcout << f.str() << std::endl;
}
```

Note that when inserting this code, the Problems tab is populated with error messages signifying a number of unresolved dependencies. This is because we have yet to configure our project to work with Boost.



Right-click the project folder and select Properties. In the C/C++ Build section, verify that you have a toolchain selected, such as Cygwin GCC:



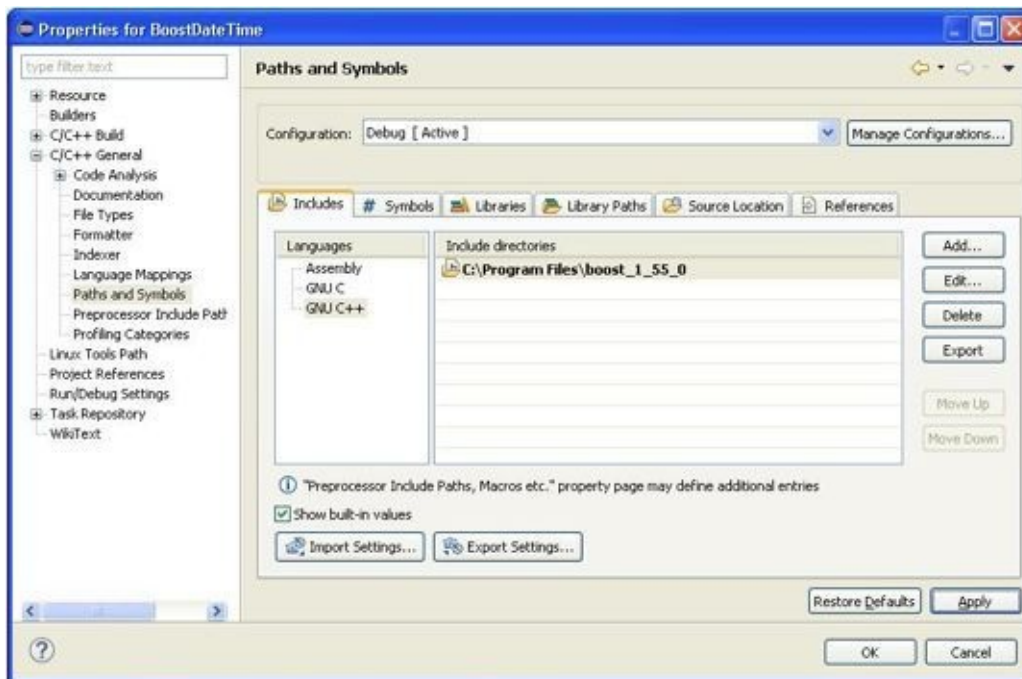
Select the C/C++ General section. In the Paths and Symbols section select the Includes tab and the 'GNU C++' language.

Now we need to tell Eclipse where to find the Boost library folder we extracted. Select the Add button and when the Add directory path' dialog is launched, select the 'File

system...' button to set the Boost folder directory:



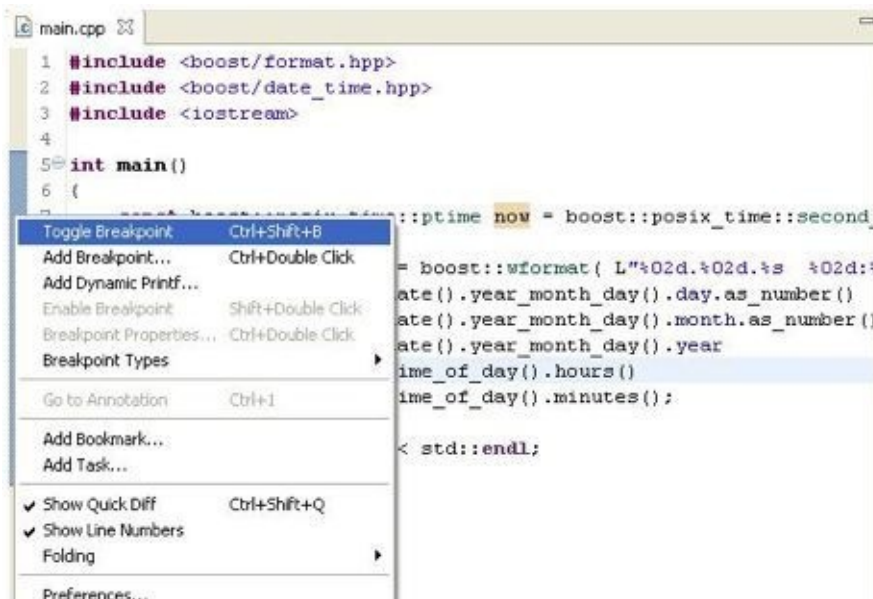
Click OK and then click Apply when you are returned to the project Properties dialog:



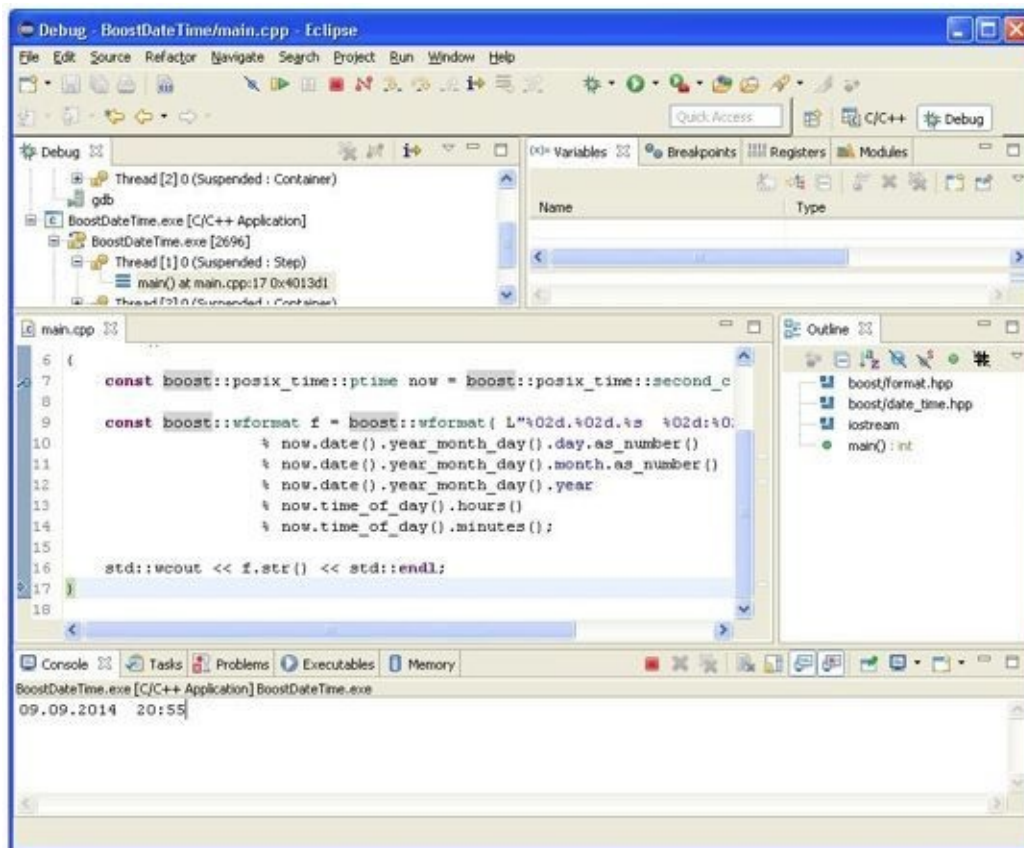
Repeat by adding the same path for the 'GNU C' language. From the menu, select **Select Project > Clean**, making sure to check the BoostDateTime checkbox:



Press OK to clean and build the project. To start debugging your Eclipse project, right-click the left-side of the code window and select 'Toggle breakpoint':



A small button should appear at the point at where you chose the breakpoint. To start debugging, select Run > Debug from the main menu. Once the debugging process starts you will get directed to the Debug perspective. Press F6 to step through each line of code, and view the results in the Console tab:



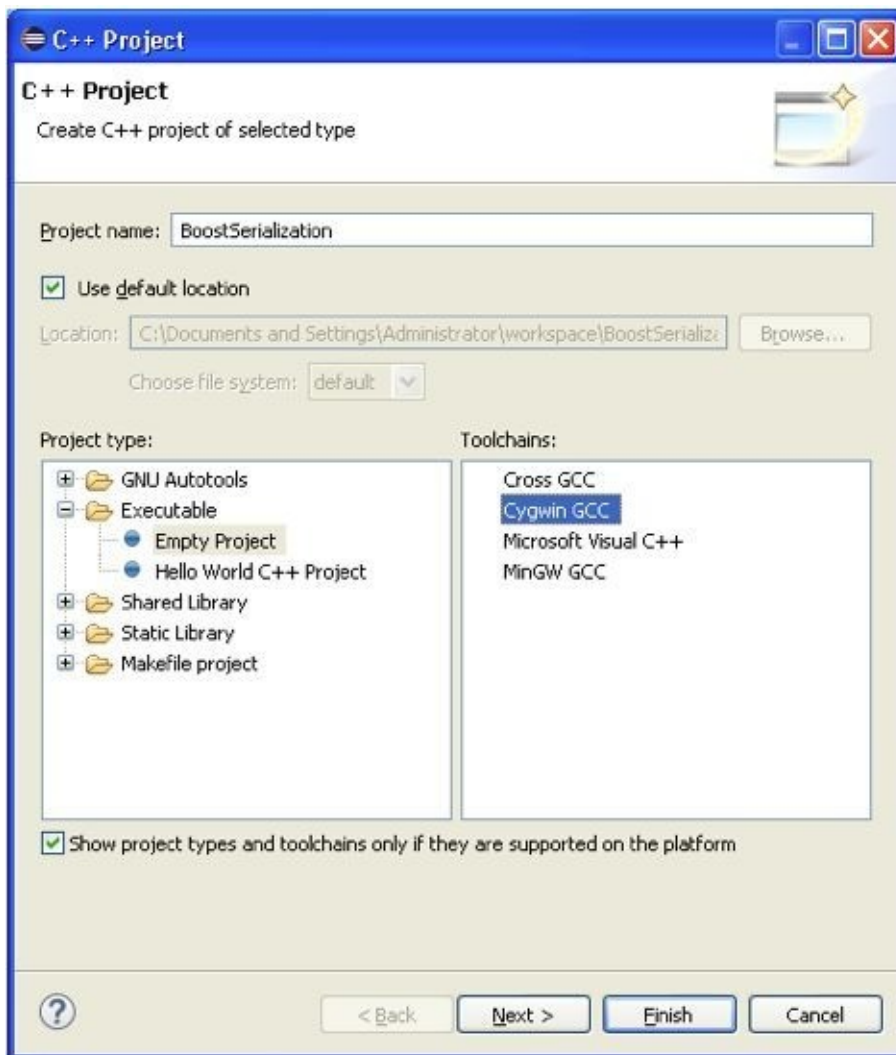
This gives us the desired output via Boost Format and that's all you need to do to use Boost non-Compiled libraries in Eclipse.

Compiled library example: Boost Serialization

Using the compiled Boost libraries in Eclipse involves a few additional steps to the ones described for header-only libraries.

For this example, we will demonstrate the use of Boost Serialization, which is a means of converting C++ objects into a format that can be saved and restored at a later date.

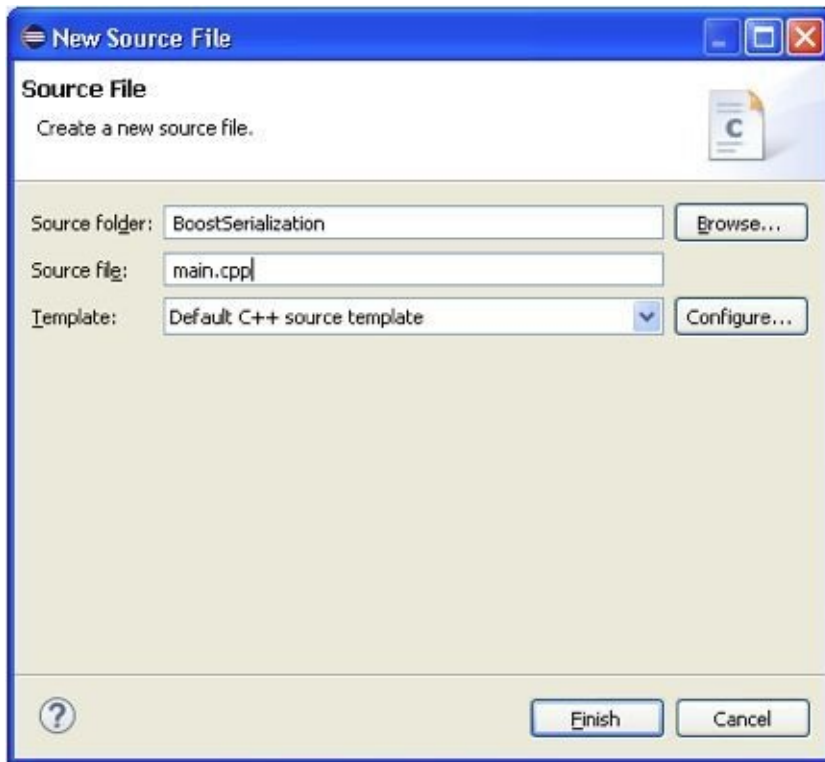
As before, select New > C/C++ Project.



Click next and ensure the Debug and Release checkboxes are set as before.

Click Finish.

As with the non-compiled example, right-click the project and select New > Source File.



Then right click the src folder just created and select New > Source File. We will call it main.cpp. Click Finish and copy in the following code, an example usage of Boost Serialization:

```
#include <vector>  
#include <fstream>  
#include "boost/serialization/vector.hpp"  
#include "boost/archive/text_oarchive.hpp"  
#include "boost/archive/text_iarchive.hpp"  
  
class Info  
{  
private:  
// Allow serialization to access non-public data members.  
friend class boost::serialization::access;  
  
// Serialize the std::vector member of Info  
template<class Archive>  
void serialize(Archive & ar, const unsigned int version)
```

```
{  
    ar & filenames;  
}
```

```
std::vector<std::string> filenames;
```

```
public:
```

```
void AddFilename( const std::string& filename );
```

```
void Print() const;
```

```
};
```

```
void Info::Print() const
```

```
{
```

```
    std::copy(filenames.begin(),
```

```
    filenames.end(),
```

```
    std::ostream_iterator<std::string>(std::cout, "\n"));
```

```
}
```

```
void Info::AddFilename( const std::string& filename )
```

```
{
```

```
    filenames.push_back( filename );
```

```
}
```

```
int main(int argc, char** argv)
```

```
{
```

```
    Info info;
```

```
    info.AddFilename( "ThisFile.txt" );
```

```
    info.AddFilename( "ThatFile.txt" );
```

```
    info.AddFilename( "OtherFile.txt" );
```

```
// Save filename data contained in Info object
```

```
{
```



```
// Create an output archive
std::ofstream ofs( "store.dat" );
boost::archive::text_oarchive ar(ofs);

// Save the data
ar & info;
}

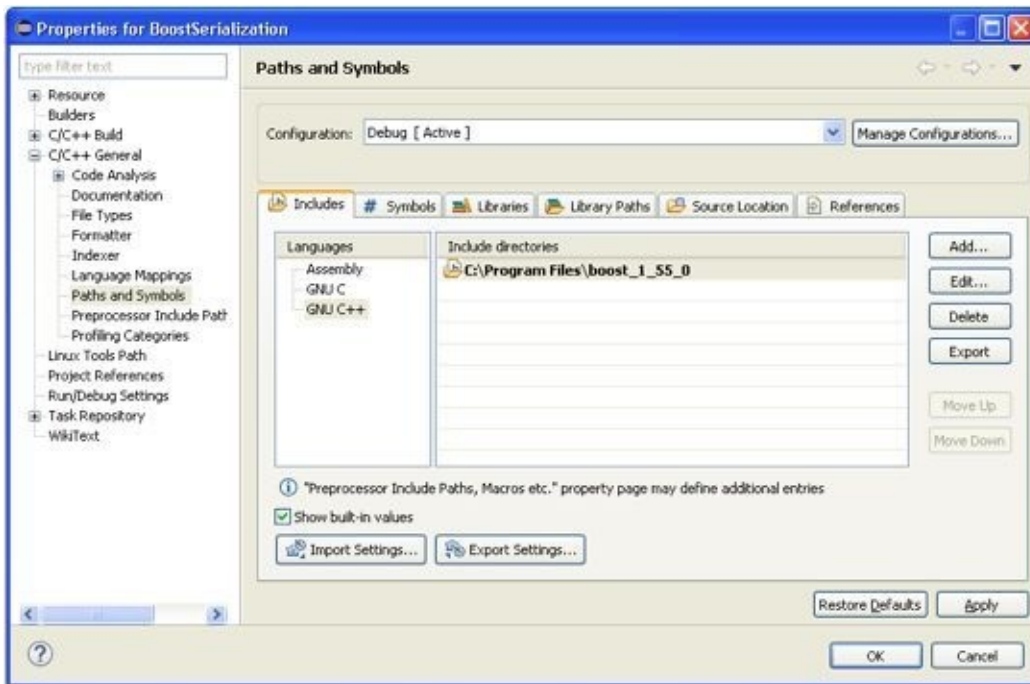
// Restore from saved data and print to verify contents
Info restored_info;
{
// Create and input archive
std::ifstream ifs( "store.dat" );
boost::archive::text_iarchive ar(ifs);

// Load the data
ar & restored_info;
}

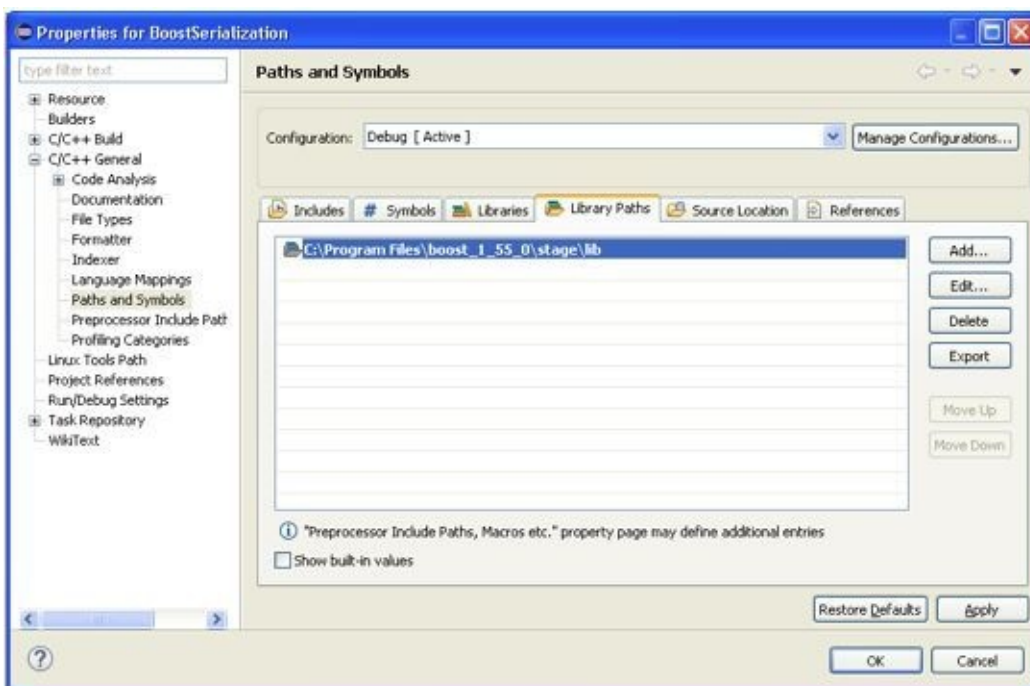
restored_info.Print();

return 0;
}
```

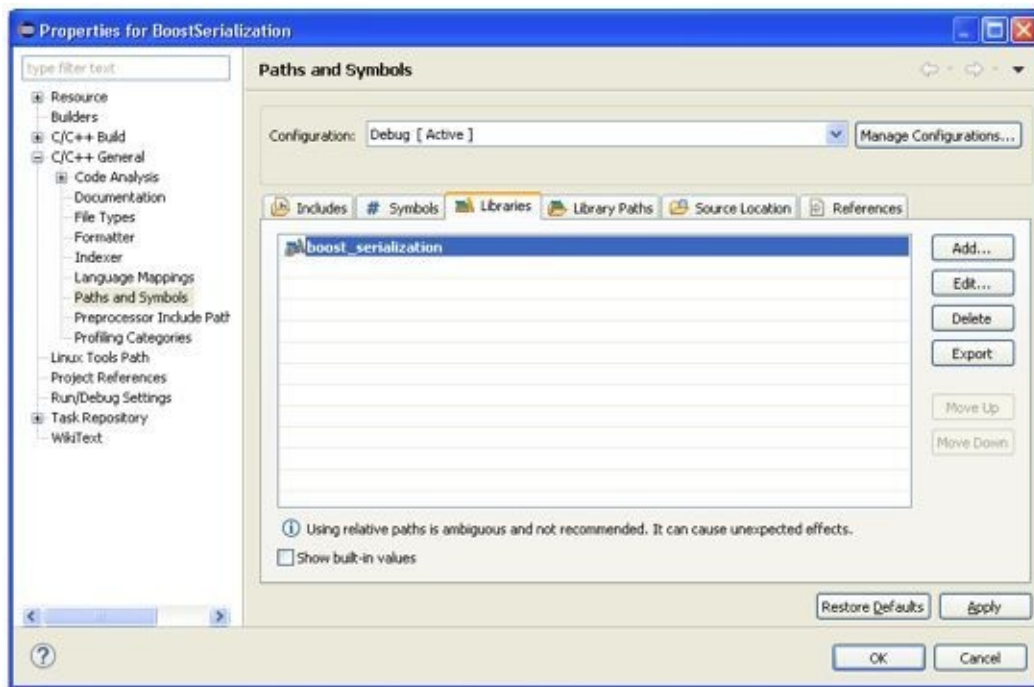
As with the header-only example, we need to set the include directory for this project:



For our compiled library example we also need to set the location of the library path. In the Properties dialog, select the 'Library Paths' tab, click the 'Add' button. In the 'Add' dialog that appears, set the library path location:



In the Libraries tab, add the 'boost_serialization' library:

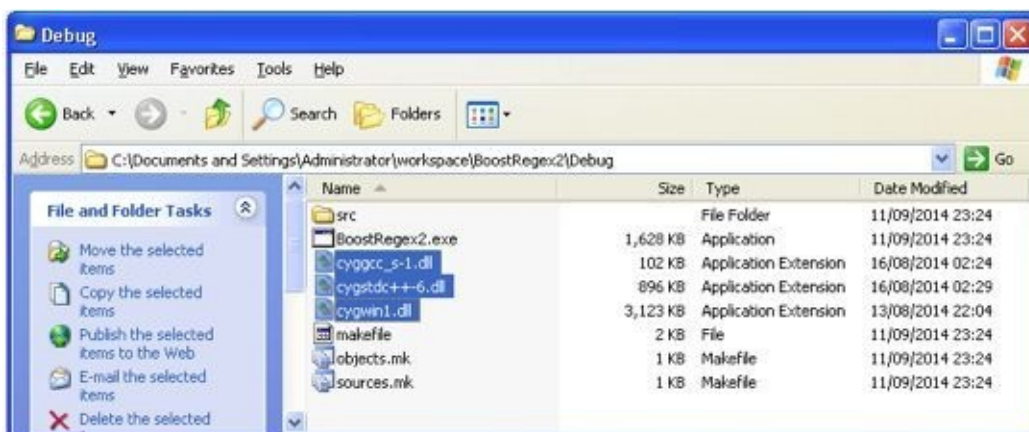


There remains one final task for our example to work: copying the following dlls from the C:\Cygwin/bin folder into the Debug/Release folder where our executable is located:

cyggcc_s-1.dll

cygstdc++-6.dll

cygwin1.dll

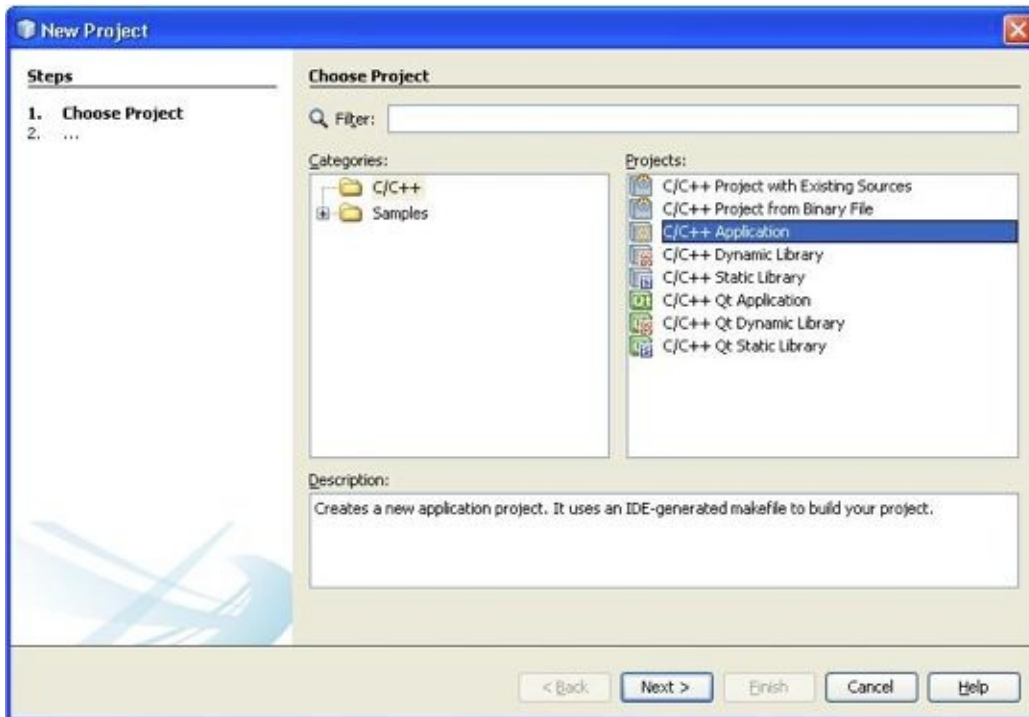


That's because directories on the Environment Variables PATH are not always searched (in my experience), so we put the DLLs in our application directory to be safe and enable us to build and run each application.

3.5 Configuring Boost in NetBeans

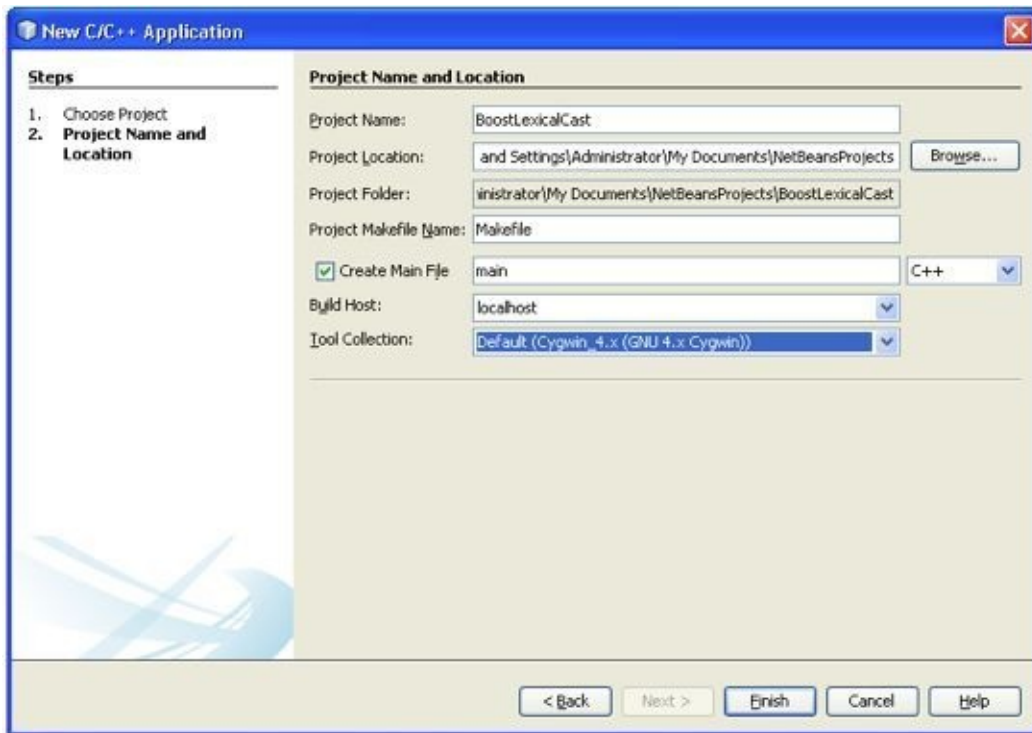
Header-only library example: Boost Lexical Cast

For our NetBeans header-only example we will demonstrate the use of Boost Lexical Cast, a safe means of converting arbitrary data types such as integers into literal text forms, and vice versa. In NetBeans select File > New Project > C/C++ Application:



Click Next and give your project a name and click Finish.

Notice how NetBeans recognizes the Tools collection installed as being Cygwin if installed.



Locate the main.cpp source file in the Projects tab and replace it with the following code example that implements Boost Lexical Cast .

```
#include <boost/lexical_cast.hpp>
```

```
int main()
```

```
{
```

```
try
```

```
{
```

```
std::cout << boost::lexical_cast<double>("3.141592") << "\n";
```

```
}
```

```
catch( boost::bad_lexical_cast const& e )
```

```
{
```

```
std::cout << "Error: " << e.what() << "\n";
```

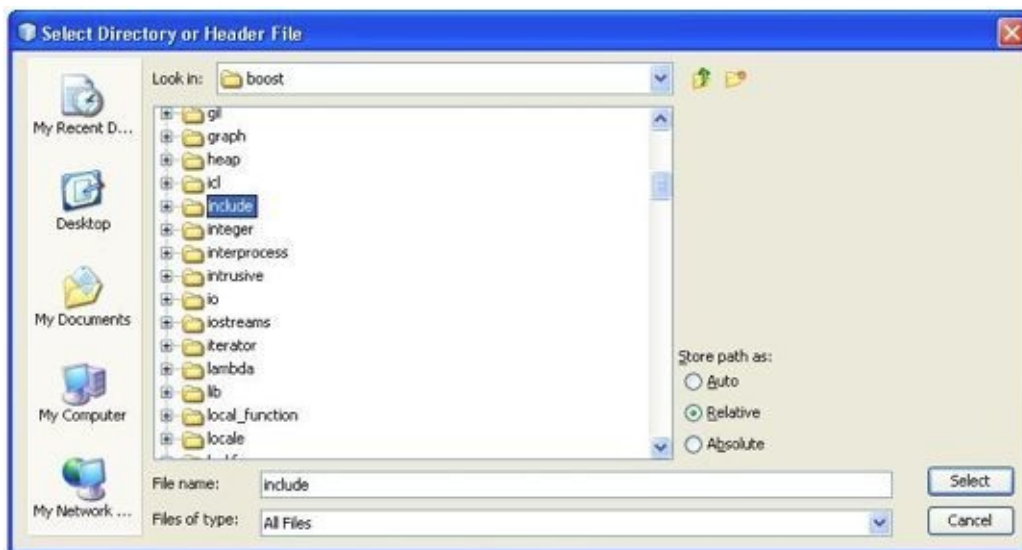
```
}
```

```
return 0;
```

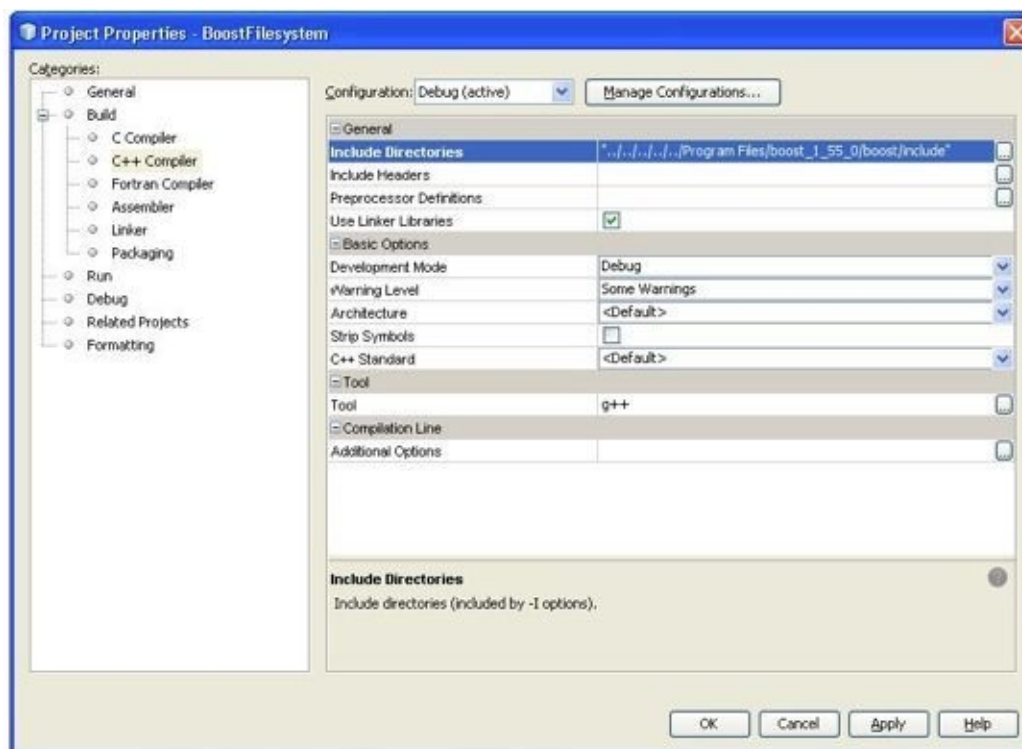
```
}
```

Given that Boost Lexical Cast is a header-only library, we need only set the additional include directories in our NetBeans project. Right-click your project folder and select

Properties. In the Build > C++ Compiler category, edit the 'Include Directories' section to set where the boost include files are located. You can use absolute paths or relative paths:



The Include directories will look like this:

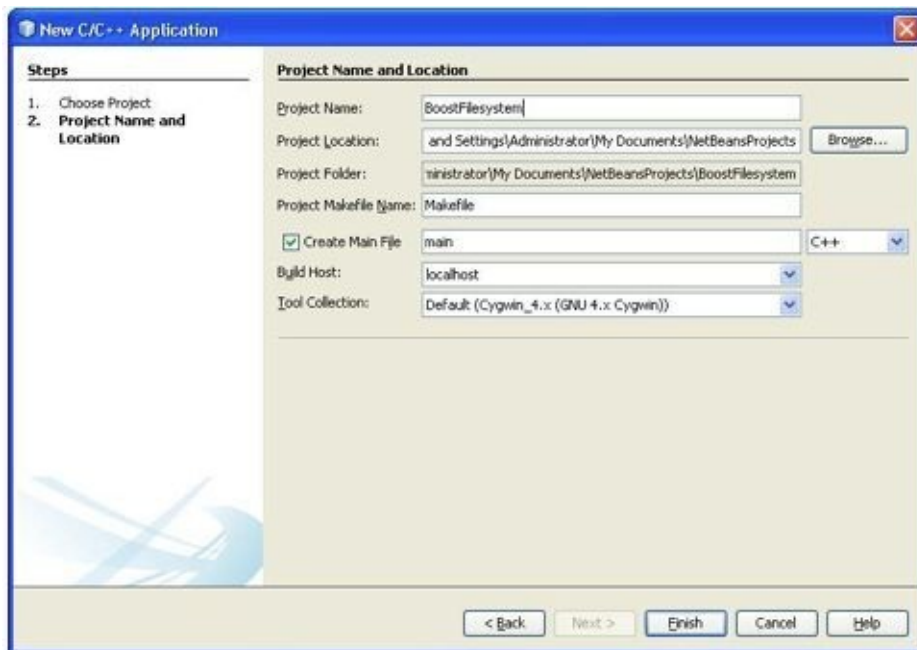


We are now done with configuring Netbeans to work with a Boost header-only library. All that remains is to build the program which gives us the following output:

RUN SUCCESSFUL (total time: 219ms)

Compiled library example: Boost Filesystem

The Boost Filesystem library is an interface for identifying and manipulating files and directories, and is another example of a Boost library requiring separate compilation. Create a new C++ Application as per the previous Netbeans example:



In main.cpp copy this example of using Boost Filesystem to iterate over and print the names of files in a directory:

```
#include <boost/filesystem.hpp>
#include <boost/foreach.hpp>

int main()
{
    boost::filesystem::path targetDir( "C:\\MyStuff" );
    boost::filesystem::directory_iterator it( targetDir ), eod;
    BOOST_FOREACH(boost::filesystem::path const &p, std::make_pair( it, eod ) )
    {
        if( is_regular_file( p ) ) {
            std::string filename = p.filename().string();
            std::cout << filename << std::endl;
        }
    }
}
```

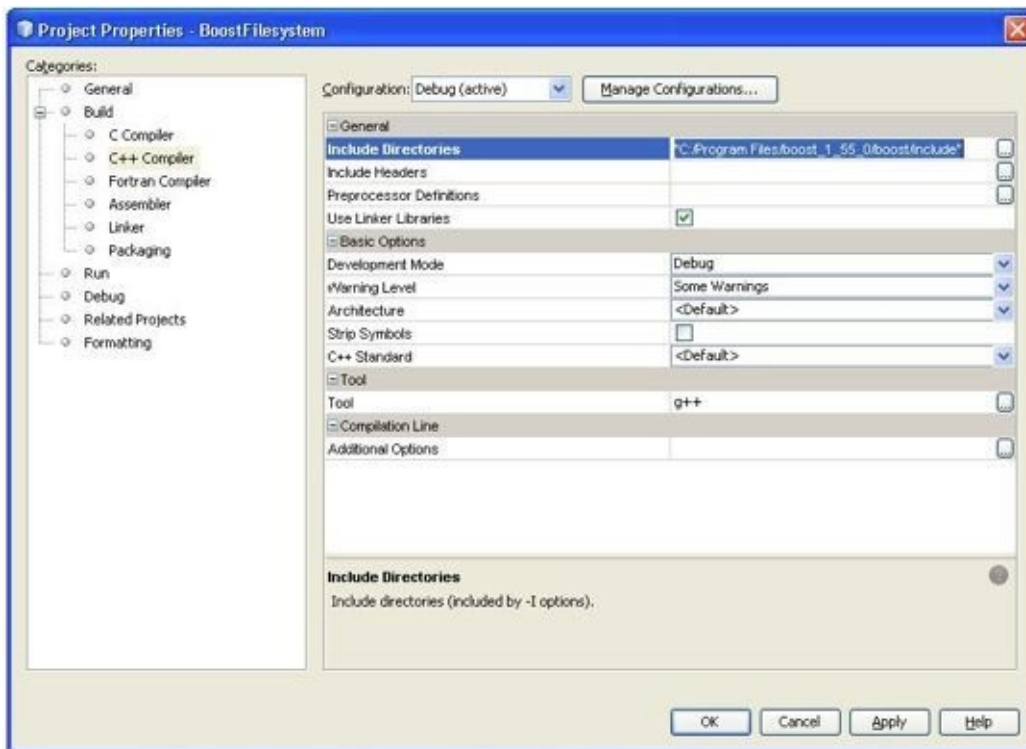


```

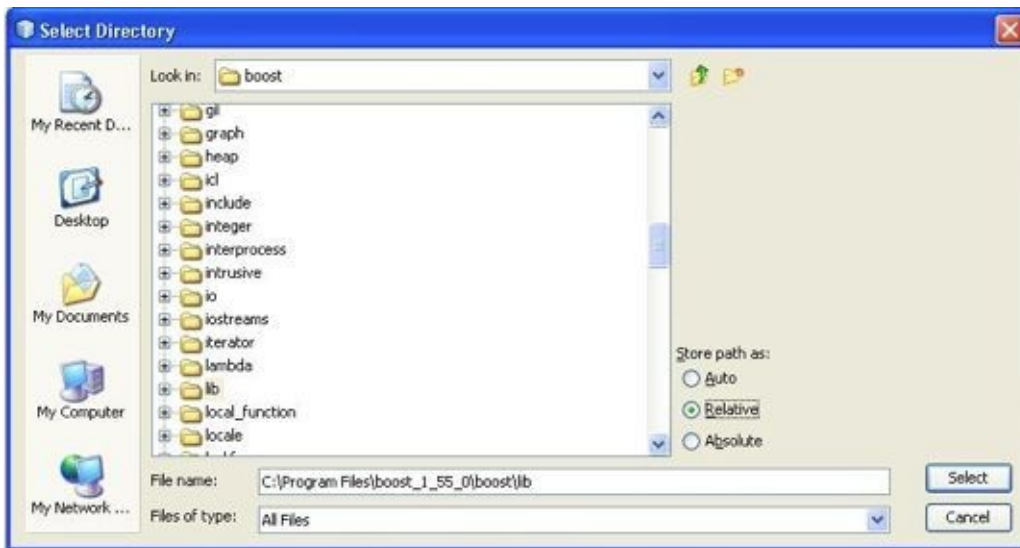
}
}
return 0;
}

```

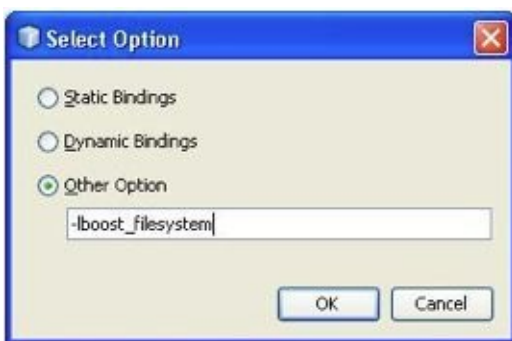
To get this project to properly build, first the set the Boost include directory as before:



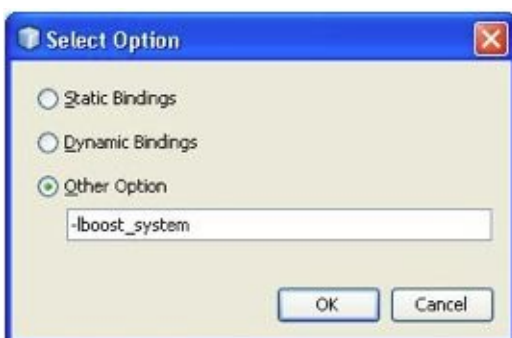
And because this is using a compiled library, we need to set the Linker options, where the library directories reside and what additional command line options are to be used. In the Linker section, select 'Additional Library Directories':



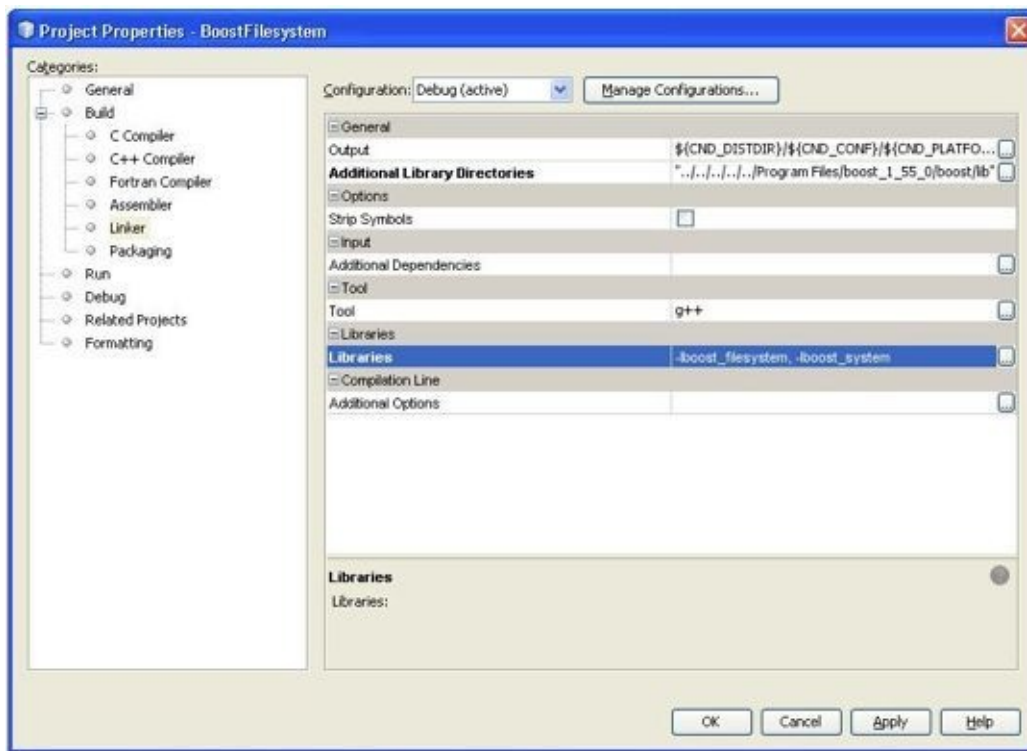
When setting additional include libraries, I prefer using relative paths as the full absolute file path is not needed. Then go to the Libraries section, and select the 'Add Option...' button. Check the 'Other Option' radio button and type `-lboost_filesystem` and click OK.



When using Boost Filesystem you will need to include `-lboost_system` as well, so repeat for this option:



When done the Linker property page will look like this:



This example usage of Boost Filesystem should now compile and run:

ABC.txt

RUN SUCCESSFUL (total time: 282ms)

Chapter Summary

To summarize this latest master lesson, you are now able to:

- Configure the Boost libraries, both header-only and compiled, for use in commonly-used Integrated Development environments including Microsoft Visual Studio, Eclipse and NetBeans.
- Create and use the compiled libraries required by commonly-used compiler environments such as Visual C++, MinGW and Cygwin.
- Apply the Boost libraries in your applications including those that implement file systems, dates and times, regular expressions, serialization and output formatting.

Chapter 4: Practical Applications of Graph Algorithms

A graph is a data structure that is used to represent networks and their interconnections. Network nodes are frequently known as vertices, while the links connecting them are known as edges or sometimes arcs. In practical scenarios, graphs can be used to model telecommunications networks, transportation systems, water distribution networks (pipes) and electrical wiring amongst other things. Graphs are usually visually represented as a series of dots or circles representing the vertices and lines drawn between them to represent the edges, sometimes using arrows to indicate direction.

4.1 Modeling Networks as Graphs

This example **Graph** class is based on Sedgewick's C++ implementation², specifically, the **DenseGRAPH** class. Sedgewick's implementation (see <https://www.cs.princeton.edu/~rs/Algs3.cxx5/code.txt>) specifically track the number of edges and vertices, have methods for inserting and removing edges, and use an adjacency matrix or adjacency list to track interconnections between vertices. I have modified the class slightly to allocate **Edge** objects as smart pointers to ensure memory is deallocated upon completion.

```
#include <vector>
```

```
// This code is from "Algorithms in C++, Third Edition,"
```

```
// by Robert Sedgewick, Addison-Wesley, 2002.
```

```
class Edge
```

```
{
```

```
public:
```

```
int v, w;
```

```
double wt;
```

```
Edge(int v = -1, int w = -1, double wt = 0.0) :
```

```
v(v), w(w), wt(wt) { }
```

```
};
```

```
typedef std::shared_ptr<Edge> EdgePtr;
```

```
template <class Edge>
```

```
class DenseGRAPH
```

```
{
```

```
private:
```

```
int Vcnt, Ecnt;
```

```
bool digraph;
```

```
std::vector <std::vector<EdgePtr>> adj;
```

```
public:
```

```
DenseGRAPH(int V, bool digraph = false) :
```

```
adj(V),  
Vcnt(V),  
Ecnt(0),  
digraph(digraph)  
{  
for ( int i = 0; i < V; i++ ) adj[i].resize( V );  
}
```

```
int V() const { return Vcnt; }  
int E() const { return Ecnt; }
```

```
bool directed() const { return digraph; }  
void insert( EdgePtr e )  
{  
int v = e->v;  
int w = e->w;  
if (adj[v][w] == 0) Ecnt++;  
adj[v][w] = e;  
if (!digraph) adj[w][v] = e;  
}
```

```
void remove(EdgePtr e)  
{  
int v = e->v;  
int w = e->w;  
  
if (adj[v][w] != 0)  
Ecnt--;
```

```
adj[v][w] = 0;
```

```
if (!digraph) adj[w][v] = 0;
```

```
}
```

```
EdgePtr edge(int v, int w) const  
{ return adj[v][w]; }
```

```
class adjIterator;  
friend class adjIterator;  
};
```

```
template <class Edge>  
class DenseGRAPH<Edge>::adjIterator  
{  
private:
```

```
const DenseGRAPH &G;  
int i, v;
```

```
public:  
adjIterator(const DenseGRAPH<Edge> &G, int v) :  
G(G),  
v(v),  
i(0) {}
```

```
EdgePtr beg() { i = -1; return nxt(); }
```

```
EdgePtr nxt()  
{  
for (i++; i < G.V(); i++)  
if (G.edge(v, i))  
return G.adj[v][i];  
return 0;  
}
```



```
bool end() const { return i >= G.V(); }  
};
```

Representations of network topologies can be created by creating an instance of **DenseGRAPH** (renamed as **Graph**) and adding the sets of edges:

```
#include "Graph.h"  
#include <iostream>  
int main()  
{  
Graph<Edge>* graph = new Graph<Edge>( 5, false );  
  
graph->insert( EdgePtr( new Edge( 0, 1 ) ) );  
graph->insert( EdgePtr( new Edge( 1, 2 ) ) );  
graph->insert( EdgePtr( new Edge( 2, 3 ) ) );  
graph->insert( EdgePtr( new Edge( 3, 4 ) ) );  
graph->insert( EdgePtr( new Edge( 0, 4 ) ) );  
  
std::cout << "No. vertices = " << graph->V() << std::endl;  
std::cout << "No. edges = " << graph->E() << std::endl;  
EdgePtr edgePtr = graph->edge( 2, 3 );  
return 0;
```

4.2 Implementing Dijkstra's Algorithm

Dijkstra's algorithm solves the shortest path problem for a graph with non-negative edge weights by producing a shortest path tree. This algorithm is often used in routing and as a subroutine in other graph algorithms such as the k-shortest paths algorithm. We build a shortest path tree (SPT) one edge at a time by always taking the next edge that gives a shortest path from the source to a vertex not on the SPT. In other words, we add vertices to the SPT in order of their distance (through the SPT) to the start vertex.

The original Dijkstra's algorithm did not use min priority queues and was of complexity $O(|V|^2)$, where V is the number of vertices. This performance was later improved by the use of linked-list representations and min priority queues. Edges that connect tree vertices to nontree vertices are stored using a priority queue implementation that allows for the updating of priorities.

This implementation initially finds all shortest paths within the constructor of the SPT template class and uses a priority queue of vertices held in order of their distance from the source node. This determines all the shortest paths from the source node.

The queue is initialized with priority 0 for the source node and priority infinity for the other nodes. The implementation then enters a loop whereby the lowest-priority node from the queue is moved to the SPT and relaxed along its incident edges. To relax an edge $v \rightarrow w$ means to test whether the best known way to get from s to w is to go from s to v , and then go from v to w . If so, we update our data structures.

```
#include "Graph.h"
```

```
template <class Graph, class Edge>
```

```
class SPT
```

```
{
```

```
private:
```

```
const Graph &G;
```

```
std::vector<double> wt;
```

```
std::vector<EdgePtr> spt;
```

```

public:
SPT(const Graph& G, int s) :
G(G),
spt(G.V()),
wt(G.V(), G.V())
{
PQi<double> pQ(G.V(), wt);

for (int v = 0; v < G.V(); v++)
{
pQ.insert(v);
}
wt[s] = 0.0;
pQ.lower(s);

while (!pQ.empty())
{
int v = pQ.getmin();
if ( v != s && spt[v] == 0 ) return;

typename Graph::adjIterator A(G, v);
for ( EdgePtr e = A.beg(); !A.end(); e = A.nxt() )
{
int w = e->w;

double P = wt[v] + e->wt;
if ( P < wt[w] )
{
wt[w] = P;
pQ.lower(w);
spt[w] = e;
}
}
}

```

```

}
}
}
EdgePtr pathR(int v) const
{
return spt[v];
}

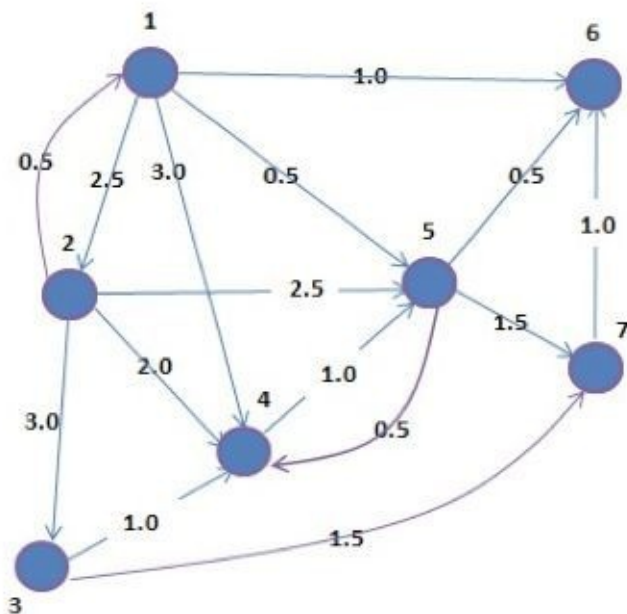
```

```

double dist(int v) const
{
return wt[v];
}
};

```

Suppose we wish to use the **Graph** data structure to represent the following network nodes and interconnections:



We can represent this by building a **Graph** structure with calls to the insert function:

```

std::auto_ptr< Graph<Edge> > graph( new Graph<Edge>( 8, true ) );

graph->insert( EdgePtr( new Edge( 1, 6, 1.0 ) ) );

```

```

graph->insert( EdgePtr( new Edge( 1, 2, 2.5 ) ) );
graph->insert( EdgePtr( new Edge( 1, 4, 3.0 ) ) );
graph->insert( EdgePtr( new Edge( 1, 5, 0.5 ) ) );
graph->insert( EdgePtr( new Edge( 2, 1, 0.5 ) ) );
graph->insert( EdgePtr( new Edge( 2, 3, 3.0 ) ) );
graph->insert( EdgePtr( new Edge( 2, 4, 2.0 ) ) );
graph->insert( EdgePtr( new Edge( 2, 5, 2.5 ) ) );
graph->insert( EdgePtr( new Edge( 3, 4, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 3, 7, 1.5 ) ) );
graph->insert( EdgePtr( new Edge( 4, 5, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 5, 4, 0.5 ) ) );
graph->insert( EdgePtr( new Edge( 5, 6, 0.5 ) ) );
graph->insert( EdgePtr( new Edge( 5, 7, 1.5 ) ) );
graph->insert( EdgePtr( new Edge( 7, 6, 1.0 ) ) );

```

To find all the shortest paths emanating from node 2 is a straightforward matter of creating the SPT template class instance.

This computes the complete shortest path tree in the constructor. The following code snippet shows how to generate the shortest paths and display the shortest path possible between nodes 2 to 7, which in this case would be 2->1->5->7:

```

// Find shortest path between nodes 2->7
int source = 2;
int target = 7;

std::stack<int> path;

SPT<Graph<Edge>, Edge> sp( *graph, source );
while ( true )
{
    EdgePtr e = sp.pathR( target );

```

```
path.push( target );
```

```
if ( target == source )
```

```
break;
```

```
target = e->v;
```

```
}
```

```
// Print the path nodes
```

```
while ( !path.empty() )
```

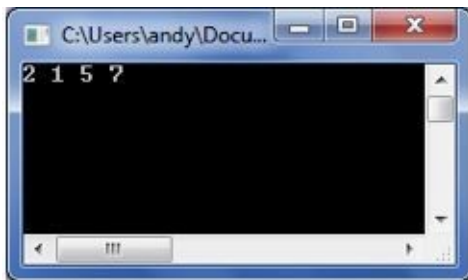
```
{
```

```
std::cout << path.top() << " ";
```

```
path.pop();
```

```
}
```

This gives the shortest possible path between nodes 2 and 7 as [2,1,5,7]:



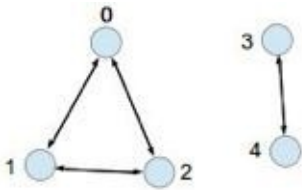
4.3 Testing for Graph Connectivity

A frequent requirement for graphs is as an efficient means of determining whether it is connected - or not.

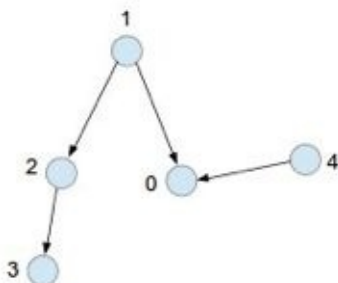
Is it possible a selected pair of nodes to communicate with each other? Or is the graph disconnected in some way?

Using the same **Graph** class discussed in the previous sections, this program uses a depth-first search algorithm to test whether all the graph nodes get visited during the recursive search. Here are some examples:

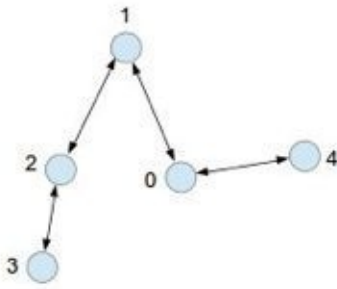
1. A disconnected un-directed graph, whereby nodes [3,4] are disconnected from nodes [0,1,2], thereby making it impossible for node [0] to communicate with node [3]:



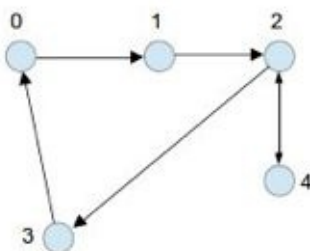
2. A disconnected directed graph. Node [1] can communicate with nodes [0,2,3] but not node [4], since a directed link exists between node [4] to node [0], but not node [0] to node [4]:



3. A connected un-directed graph. All nodes can communicate with each other:



4. A connected directed graph. All nodes can communicate with any other node. Though there is no link between nodes [0,3], a path between the two nodes exists via nodes [0,1,2,3].



Now let's add to our **Graph** class the following utility functions used to test if the graph is strongly connected:

// A recursive function to print DFS starting from v

void Graph::DFSUtil(int v, bool visited[])

{

// Mark the current node as visited and print it

visited[v] = true;

typename Graph::adjIterator A(*this, v);

for (EdgePtr e = A.beg(); !A.end(); e = A.nxt()) {

int w = e->w == v ? e->v : e->w;

if (!visited[w]) DFSUtil(w, visited);

}

}

// Function that returns reverse (or transpose) of this graph

Graph* Graph::getTranspose()

{

Graph<Edge>* g = new Graph<Edge>(Vcnt, digraph);

for (int v = 0; v < Vcnt; v++)

{

typename Graph::adjIterator A(*this, v);

for (EdgePtr e = A.beg(); !A.end(); e = A.nxt()) {

int w = e->w;

g->insert(EdgePtr(new Edge(w, v)));

}

}

return g;

}

// The main function that returns true if graph is strongly connected

bool Graph::isSC()

{

// Step 1: Mark all the vertices as not visited (For first DFS)

for (int i = 0; i < Vcnt; i++) visited[i] = false;

// Step 2: Do DFS traversal starting from first vertex.

DFSUtil(0, visited);

// If DFS traversal doesn't visit all vertices, then return false.

for (int i = 0; i < Vcnt; i++)

if (visited[i] == false) return false;

// Step 3: Create a reversed graph

Graph* gr = getTranspose();

// Step 4: Mark all the vertices as not visited (For second DFS)

for(int i = 0; i < Vcnt; i++) visited[i] = false;

// Step 5: Do DFS for reversed graph starting from first vertex.

// Starting Vertex must be same starting point of first DFS

gr->DFSUtil(0, visited);

// If all vertices are not visited in second DFS, return false

for (int i = 0; i < Vcnt; i++)

if (visited[i] == false) return false;

return true;

}

The following code listing shows how to implement each of the four examples listed above:

#include "Graph.h"

#include <iostream>

int main()

{

// Example 1: disconnected un-directed graph:

//

// 0 3

// /

// 1 2 4

// nodes [3,4] disconnected from nodes [0,1,2]

Graph<Edge> g1(5, false);

g1.insert(EdgePtr(new Edge(0, 1)));

g1.insert(EdgePtr(new Edge(0, 2)));

g1.insert(EdgePtr(new Edge(3, 4)));

g1.isSC() ? std::cout << "Yes\n" : std::cout << "No\n";

// Example 2: disconnected (directed) graph:

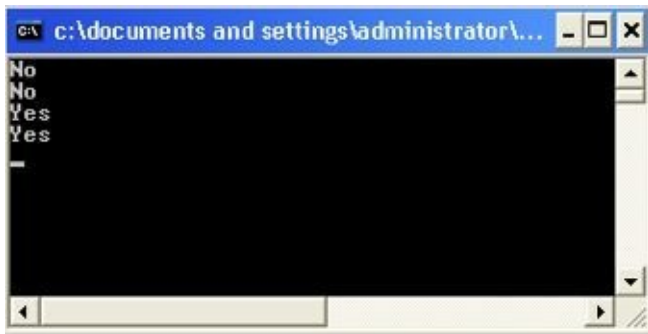
```

//
// 3 <- 2 <- 1 -> 0 <- 4
// eg 1 can communicate with [2,3,0] but not 4
Graph<Edge> g2( 5, true );
g2.insert( EdgePtr( new Edge(1, 0) ) );
g2.insert( EdgePtr( new Edge(4, 0) ) );
g2.insert( EdgePtr( new Edge(1, 2) ) );
g2.insert( EdgePtr( new Edge(2, 3) ) );
g2.isSC() ? std::cout << "Yes\n" : std::cout << "No\n";
// Example 3: connected (un-directed) graph:
//
// 3 <-> 2 <-> 1 <-> 0 <-> 4
// All nodes can communicate with any other node
Graph<Edge> g3( 5, false );
g3.insert( EdgePtr( new Edge(1, 0) ) );
g3.insert( EdgePtr( new Edge(4, 0) ) );
g3.insert( EdgePtr( new Edge(1, 2) ) );
g3.insert( EdgePtr( new Edge(2, 3) ) );
g3.isSC() ? std::cout << "Yes\n" :
std::cout << "No\n";
// Example 4: connected directed graph
//
// 0 -> 1 -> 2 -> 3 -> 0
//
// 4 -> 2 -> 4
// All nodes can communicate with any other node
Graph<Edge> g4( 5, true );
g4.insert( EdgePtr( new Edge(0, 1) ) );
g4.insert( EdgePtr( new Edge(1, 2) ) );
g4.insert( EdgePtr( new Edge(2, 3) ) );
g4.insert( EdgePtr( new Edge(3, 0) ) );
g4.insert( EdgePtr( new Edge(2, 4) ) );

```

```
g4.insert( EdgePtr( new Edge(4, 2) ) );  
g4.isSC() ? std::cout << "Yes\n" :  
std::cout << "No\n";  
return 0;  
}
```

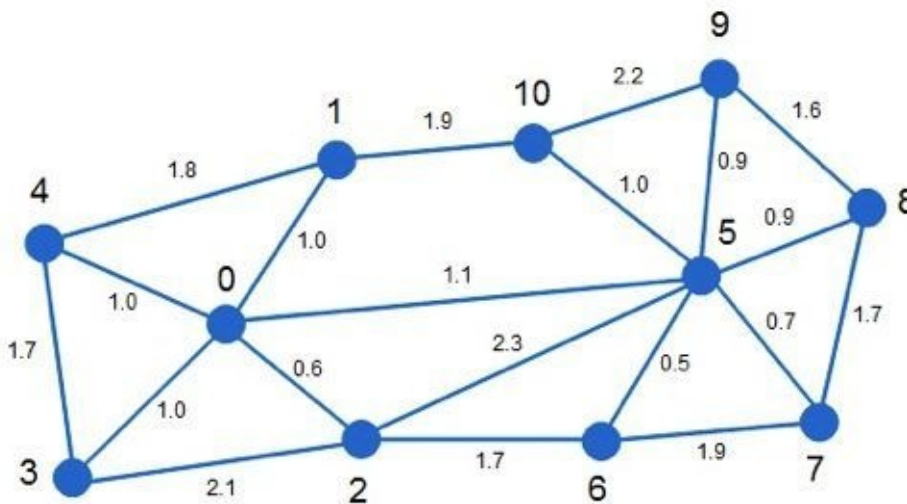
This gives the following output to determine the connected status of each graph:



4.4 Kruskal's Algorithm

Now let's look at a simple C++ implementation of Kruskal's algorithm which is used for finding minimal spanning trees in networks. This algorithm employs the same **Graph** structure to model a network, insert and delete edges, and so forth.

Consider the following 11-node example network showing the network nodes and the weight values of the links connecting them:



This is built using multiple calls to insert as seen here:

```
graph.insert( EdgePtr( new Edge( 0, 1, 1.0 ) ) );
```

Kruskal's algorithm works by obtaining a sorted list of the network links and then adding each least-cost link to another set while disallowing cycles. The complete code is as follows:

Graph.h

```
#include <vector>  
#include <algorithm>  
#include <memory>
```

```
// This code is from "Algorithms in C++, Third Edition,"  
// by Robert Sedgewick, Addison-Wesley, 2002.
```

```

class Edge
{
public:
    int v, w;
    double wt;
    Edge(int v = -1, int w = -1, double wt = 0.0) :
        v(v), w(w), wt(wt) { }
};

typedef std::shared_ptr<Edge> EdgePtr;

struct EdgeSort
{
    bool operator() ( const EdgePtr& ep1, const EdgePtr& ep2 )
    {
        return ep1->wt < ep2->wt;
    }
};

template <class Edge>
class DenseGRAPH
{
private:
    int Vcnt, Ecnt;
    bool digraph;
    std::vector <std::vector<EdgePtr>> adj;

public:
    DenseGRAPH(int V, bool digraph = false) :
        adj(V),
        Vcnt(V),
        Ecnt(0),

```

```
digraph(digraph)
{
for ( int i = 0; i < V; i++ ) adj[i].resize( V );
}
```

```
int V() const { return Vcnt; }
int E() const { return Ecnt; }
```

```
bool directed() const { return digraph; }
void insert( EdgePtr e )
{
int v = e->v;
int w = e->w;
if (adj[v][w] == 0) Ecnt++;
adj[v][w] = e;
if (!digraph) adj[w][v] = e;
}
```

```
void remove(EdgePtr e)
{
int v = e->v;
int w = e->w;

if (adj[v][w] != 0)
Ecnt--;

adj[v][w] = 0;

if (!digraph) adj[w][v] = 0;
}
```

```
EdgePtr edge(int v, int w) const
```

```
{ return adj[v][w]; }
```

```
// Get the weighted edges
```

```
void edges( std::vector<EdgePtr>& edges )
```

```
{
```

```
for( int u = 0; u < V(); ++u )
```

```
{
```

```
DenseGRAPH<Edge>::adjIterator A(*this, u);
```

```
for ( EdgePtr e = A.beg(); !A.end(); e = A.nxt() )
```

```
{
```

```
if ( NULL != e )
```

```
{
```

```
edges.push_back( e );
```

```
}
```

```
}
```

```
}
```

```
}
```

```
class adjIterator;
```

```
friend class adjIterator;
```

```
};
```

```
template <class Edge>
```

```
class DenseGRAPH<Edge>::adjIterator
```

```
{
```

```
private:
```

```
const DenseGRAPH &G;
```

```
int i, v;
```

```
public:
```



```
adjIterator(const DenseGRAPH<Edge> &G, int v) :  
G(G),  
v(v),  
i(0) {}
```

```
EdgePtr beg() { i = -1; return nxt(); }
```

```
EdgePtr nxt()  
{  
for (i++; i < G.V(); i++)  
if (G.edge(v, i))  
return G.adj[v][i];  
return 0;  
}
```

```
bool end() const { return i >= G.V(); }  
};
```

Main.cpp

```
#include "Graph.h"  
#include <iostream>
```

```
// A structure to represent a subset for union-find
```

```
struct subset  
{  
int parent;  
int rank;  
};  
typedef subset Subset;
```

```
// A utility function to find set of an element i  
// (uses path compression technique)  
int Find( Subset subsets[], int i )  
{  
// find root and make root as parent of i (path compression)  
if (subsets[i].parent != i)  
subsets[i].parent = Find(subsets, subsets[i].parent);  
return subsets[i].parent;  
}
```

```
// A function that does union of two sets of x and y by rank  
void Union( Subset subsets[], int x, int y )  
{  
int xroot = Find( subsets, x );  
int yroot = Find( subsets, y );  
// Attach smaller rank tree under root of high rank tree  
// (Union by Rank)  
if ( subsets[xroot].rank < subsets[yroot].rank )  
{  
subsets[xroot].parent = yroot;  
}  
else if (subsets[xroot].rank > subsets[yroot].rank)  
{  
subsets[yroot].parent = xroot;  
}  
else  
{  
// If ranks are same, then make one as root and increment  
// its rank by one  
subsets[yroot].parent = xroot;  
subsets[xroot].rank++;  
}
```

```
}
```

```
// The main function to construct MST using Kruskal's algorithm
```

```
void KruskalMST( DenseGRAPH<Edge>* graph, std::vector<EdgePtr>& mst )
```

```
{
```

```
const int V = graph->V();
```

```
// Sort edges in non-decreasing order of their weight
```

```
std::vector<EdgePtr> edges;
```

```
graph->edges( edges );
```

```
std::sort(edges.begin(), edges.end(), EdgeSort() );
```

```
// Allocate memory for creating V subsets
```

```
std::unique_ptr<subset[]> subsets( new subset[ V ]() );
```

```
// Create V subsets with single elements
```

```
for ( int v = 0; v < V; ++v )
```

```
{
```

```
subsets[v].parent = v;
```

```
subsets[v].rank = 0;
```

```
}
```

```
for ( std::vector<EdgePtr>::iterator it = edges.begin();
```

```
it != edges.end(); ++it )
```

```
{
```

```
EdgePtr edgePtr = *it;
```

```
int x = Find( subsets.get(), edgePtr->v );
```

```
int y = Find( subsets.get(), edgePtr->w );
```

```
// If including this edge doesn't cause cycle, include it
```

```
// in result and increment the index of result for next edge
```

```
if ( x != y )
```

```
{
```

```
mst.push_back( edgePtr );
```

```
Union( subsets.get(), x, y );
```

```

}
}
}

int main()
{
DenseGRAPH<Edge>* graph = new DenseGRAPH<Edge>( 11, true );

graph->insert( EdgePtr( new Edge( 0, 1, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 0, 2, 0.6 ) ) );
graph->insert( EdgePtr( new Edge( 0, 3, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 0, 4, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 0, 5, 1.1 ) ) );
graph->insert( EdgePtr( new Edge( 1, 4, 1.8 ) ) );
graph->insert( EdgePtr( new Edge( 1, 10, 1.9 ) ) );
graph->insert( EdgePtr( new Edge( 2, 3, 2.1 ) ) );
graph->insert( EdgePtr( new Edge( 2, 5, 2.3 ) ) );
graph->insert( EdgePtr( new Edge( 2, 6, 1.7 ) ) );
graph->insert( EdgePtr( new Edge( 3, 4, 1.7 ) ) );
graph->insert( EdgePtr( new Edge( 5, 6, 0.5 ) ) );
graph->insert( EdgePtr( new Edge( 5, 7, 0.7 ) ) );
graph->insert( EdgePtr( new Edge( 5, 8, 0.9 ) ) );
graph->insert( EdgePtr( new Edge( 5, 9, 0.9 ) ) );
graph->insert( EdgePtr( new Edge( 5, 10, 1.0 ) ) );
graph->insert( EdgePtr( new Edge( 6, 7, 1.9 ) ) );
graph->insert( EdgePtr( new Edge( 7, 8, 1.7 ) ) );
graph->insert( EdgePtr( new Edge( 8, 9, 1.6 ) ) );
graph->insert( EdgePtr( new Edge( 9, 10, 2.2 ) ) );

std::vector<EdgePtr> mst;
KruskalMST( graph, mst );

```

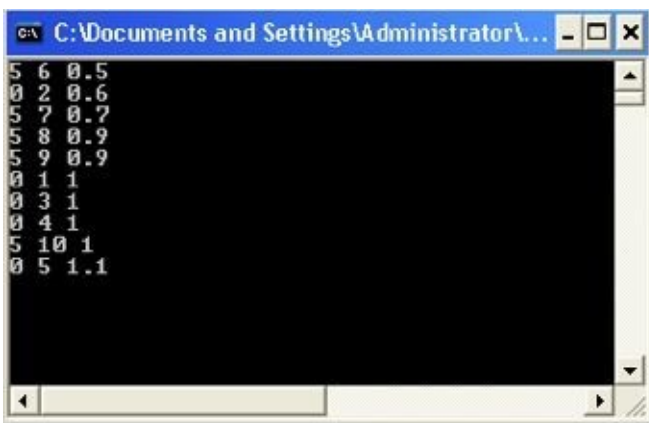
```

for ( std::vector<EdgePtr>::const_iterator it = mst.begin();
it != mst.end(); ++it )
{
std::cout << it->get()->v << " "
<< it->get()->w << " "
<< it->get()->wt << std::endl;
}

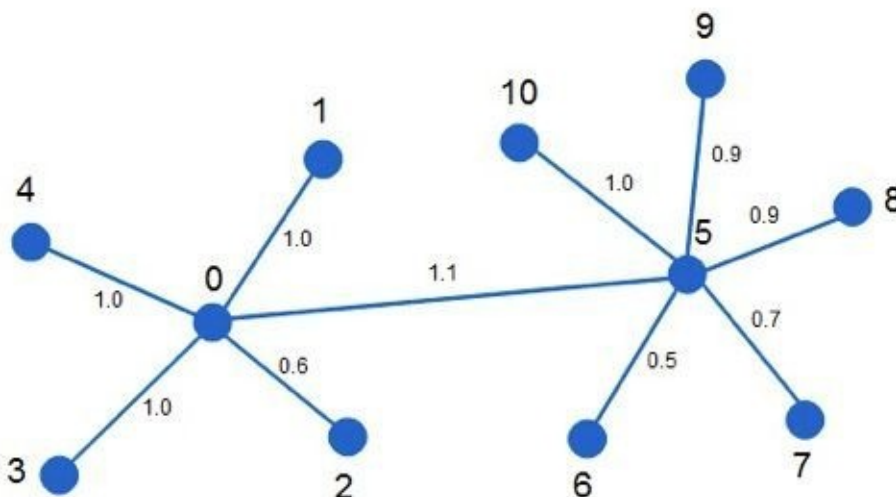
return 0;
}

```

This gives the following output:



Which in turn represents this minimal spanning tree:



Chapter Summary

To summarize this master lesson, you can now:

- Utilize a **Graph** class to model networks as a set of nodes and interconnecting edges
- Use **Graph** class methods to insert, remove and keep track of the edges and nodes in your network.
- Apply useful problem-solving algorithms for determining shortest paths, connectivity and minimal spanning trees.

Chapter 5: A Sudoku Puzzle Solver

Sudoku is a logic puzzle in which you are given a 9×9 grid of numbers in the range 1 to 9. That grid is then further sub-divided into 9 lots of 3×3 sectors.

The rules are simple: in each row, column, and 3x3 sector, each one of the numbers 1-9 must appear. At the start of the puzzle, just enough of the grid's elements are allocated to guarantee a single unique solution based on these rules. This chapter presents a C++ **Grid** class for both generating and solving a Sudoku puzzle grid.

At the heart of the class is a recursive backtracking algorithm used to actually solve the puzzle. Each recursive call to the Solve function solves the puzzle one step at a time by assigning a feasible number to each empty cell. Before the number is assigned it is first checked for feasibility: we check if the number is not already present in the current row, in the current column and in the current 3x3 grid. If it is safe to do so, the number is assigned and we recursively check to see if this latest assignment has solved the solution or not. If a solution is still not found, we try the next number for the current empty Sudoku cell. If none of the numbers 1-9 we have tried are successful then we return a false.

```
#include <vector>
```

```
#include <set>
```

```
#include <algorithm>
```

```
#include <ctime>
```

```
#include <iostream>
```

```
const int UNASSIGNED = 0;
```

```
const int N = 9;
```

```
class Grid
```

```
{
```

```
private:
```

```
std::vector<unsigned> grid;
```

```
std::set<unsigned> locations;
```

```
unsigned sizeSquared;
```

public:

Grid()

{

sizeSquared = N * N;

grid.resize(sizeSquared);

}

void Copy(unsigned matrix[N][N])

{

for (int row = 0; row < N; ++row) {

for (int col = 0; col < N; ++col) {

grid[row + col * N] = matrix[row][col];

}

}

}

void Print() const

{

for (int row = 0; row < N; row++) {

for (int col = 0; col < N; col++)

std::cout << grid[row + col * N] << " ";

std::cout << std::endl;

}

std::cout << std::endl;

}

bool Solve()

{

int row, col;

if (!HasUnassignedLocations(row, col)) return true;

// consider digits 1 to 9


```
for ( int num = 1; num <= 9; num++ ) {  
    if ( IsFeasible( row, col, num ) ) {  
        grid[ row + col * N ] = num;  
        if (Solve())  
        {  
            return true;  
        }  
        // Try again  
        grid[ row + col * N ] = UNASSIGNED;  
    }  
}  
return false;  
}
```

private:

```
bool HasUnassignedLocations( int &row, int &col )  
{  
    for (row = 0; row < N; row++)  
        for (col = 0; col < N; col++)  
            if (grid[row + col * N] == UNASSIGNED)  
                return true;  
  
    return false;  
}
```

```
bool IsFeasible(int row, int col, int num)  
{  
    // Check if number not already placed in current row, col, 3x3 box  
    return !InRow(row, num) &&  
        !InColumn(col, num) &&  
        !In3x3Box(row - row%3 , col - col%3, num);  
}
```

```
}
```

```
bool InRow(int row, int num)
```

```
{
```

```
for (int col = 0; col < N; col++)
```

```
if (grid[row + col * N] == num)
```

```
return true;
```

```
return false;
```

```
}
```

```
bool InColumn( int col, int num )
```

```
{
```

```
for (int row = 0; row < N; row++)
```

```
if (grid[row + col * N] == num)
```

```
return true;
```

```
return false;
```

```
}
```

```
bool In3x3Box( int boxStartRow, int boxStartCol, int num )
```

```
{
```

```
for (int row = 0; row < 3; row++)
```

```
for (int col = 0; col < 3; col++)
```

```
{
```

```
int rown = row+boxStartRow;
```

```
int coln = col+boxStartCol;
```

```
if (grid[rown + coln * N] == num)
```

```
return true;
```

```
}
```

```
return false;
```

```
}
```

```
};
```

```

int main()
{
    Grid gr;

    unsigned grid[N][N] = {{0, 7, 4, 3, 0, 2, 0, 0, 0},
        {0, 0, 0, 0, 0, 5, 0, 4, 0},
        {0, 0, 0, 6, 0, 7, 9, 0, 0},
        {0, 5, 6, 0, 0, 0, 7, 9, 0},
        {3, 0, 0, 0, 0, 0, 0, 0, 5},
        {0, 2, 7, 0, 0, 0, 6, 8, 0},
        {0, 0, 5, 7, 0, 1, 0, 0, 0},
        {0, 1, 0, 2, 0, 0, 0, 0, 0},
        {0, 0, 0, 4, 0, 8, 1, 6, 0}};

    gr.Copy( grid );
    std::cout << "Initial Puzzle:" << std::endl << std::endl;
    gr.Print();

    if ( true == gr.Solve() )
    {
        std::cout << "Solved Puzzle:" << std::endl << std::endl;
        gr.Print();
    }
    else
    {
        std::cout << "No solution exists";
    }
    return 0;
}

```

Now here is the console output for the C++ implementation of the Sudoku puzzle solver. It prints the initial grid and the completely filled grid as outputs:

```
C:\Documents and Settings\Administrat... - □ ×
Initial Puzzle:
0 7 4 3 0 2 0 0 0
0 0 0 0 0 5 0 4 0
0 0 0 6 0 7 9 0 0
0 5 6 0 0 0 7 9 0
3 0 0 0 0 0 0 0 5
0 2 7 0 0 0 6 8 0
0 0 5 7 0 1 0 0 0
0 1 0 2 0 0 0 0 0
0 0 0 4 0 8 1 6 0

Solved Puzzle:
9 7 4 3 1 2 8 5 6
2 6 1 9 8 5 3 4 7
5 8 3 6 4 7 9 1 2
1 5 6 8 2 4 7 9 3
3 9 8 1 7 6 4 2 5
4 2 7 5 3 9 6 8 1
6 4 5 7 9 1 2 3 8
8 1 9 2 6 3 5 7 4
7 3 2 4 5 8 1 6 9
```

In this short master lesson, you have learned the intricacies of Sudoku puzzle programming as a practical application of the **Grid** class in C++.

Chapter 6: A Mathematical Expression Parser

When implementing your own calculator it is necessary to be able to transform mathematical expressions into a format that is suitable for evaluation by computers. Expressions like $(1 + 8) - ((3 * 4) / 2)$ (known as “infix notation”) appear simple and intuitive to humans, but are less straightforward to implement in a programming language.

The shunting-yard algorithm is a method for parsing mathematical expressions written in infix notation into a format known as Reverse Polish Notation (RPN). RPN notation is different to infix notation in that every operator (+, -, * etc) comes after the operands (numbers) and there are no parentheses (brackets). This structure is more suited to evaluation by computers.

The infix expression $3 * 4$ for example becomes $3\ 4\ *$.

Pseudocode for the shunting-yard algorithm to convert an infix expression into Reverse Polish Notation:

While (tokens exist)

Read token.

If (token is a number }

add token to output queue

If (token is a function)

push token onto stack

If (token is a function argument separator eg a comma)

While (token at the top of the stack is NOT a left parenthesis)

pop operators off stack onto output queue.

If (no left parentheses were encountered)

Error - invalid input

If (token is an operator, o1)

While (operator token, o2, exists at the top of the stack **AND** (o1 is left-associative and its precedence \leq o2 **OR** o1 is right associative and its precedence $<$ o2)

pop o2 off the stack onto the output queue

push o1 onto the stack

If (token is a left parenthesis)

push token onto the stack.

If (token is right parenthesis)

While(token at the top of the stack is NOT a left parenthesis)

pop operators off the stack onto the output queue

Pop left parenthesis from the stack, but not onto output queue.

If (token at the top of the stack is a function token)

pop token onto the output queue.

If (stack has run out without finding a left parenthesis)

Error - mismatched parentheses.

While (operator tokens remain in the stack)

If (operator token on the top of the stack is a parenthesis)

Error - mismatched parentheses.

Pop the operator onto the output queue.

When the input expression becomes available in Reverse Polish Notation, it is possible to employ another stack-based algorithm to evaluate the RPN expression and hence the arithmetic result.

The pseudocode is as follows:

For each token

If (token is a number)

Push value onto stack

If (token is an operator)

Pop the 2 x top values from the stack

Evaluate the operator using popped values as arguments

Push result on to the stack

The single value remaining on the stack is the evaluation.

Full code implementation as follows:

ExpressionParser.h

#include <algorithm>

#include <string>

#include <vector>

#include <list>

#include <iostream>

#include <fstream>

#include <iterator>

#include <queue>

#include <stack>

#include <sstream>

#include <locale>

#include <stdlib.h>

#include <math.h>

class ExpressionParser

{

public:

ExpressionParser(const std::string& input);

bool MatchingParetheses();

bool Evaluate(const std::vector<std::string>& rpn, std::string& result);

bool InfixToRPN(std::vector<std::string>& inputs);

private:

void ReplaceAll(std::string& str,

const std::string& from,

const std::string& to);

```
void Tokenize( std::list<std::string>& tokens,  
const std::string& delimiter );  
private:
```

```
std::string m_strInput;  
};
```

ExpressionParser.cpp

```
#include "ExpressionParser.h"
```

```
const std::string charSet[] = { “(“, “)”, “%”, “+”, “-“, “*”, “/”, “^”, “,” };
```

```
const double pi = 3.1415927;
```

```
const double e = 2.71828182846;
```

```
template<class BidirIt>  
BidirIt Prev(BidirIt it,  
typename std::iterator_traits<BidirIt>::difference_type n = 1)  
{  
std::advance(it, -n);  
  
return it;  
}
```

```
int Modulo(int num, int div)
```

```
{  
int mod = num % div;
```

```
return ( num >= 0 || mod == 0 ) ? mod : div + mod;  
}
```



```

unsigned int OpArgCount( const std::string& s )
{

unsigned int val = 1;

if ( s == "*" || s == "/" || s == "%" ||
s == "+" || s == "-" || s == "=" ||
s == "^" || s == "POW" )
{
val = 2;
}
else if ( s == "!" )
{
val = 1;
}
return val;
}

```

// Return operator precedence

```

int OpPrecedence(const std::string& s)
{
int precedence = 1;

if ( s == "!" )
precedence = 4;
else if ( s == "*" || s == "/" || s == "%" )
precedence = 3;
else if ( s == "+" || s == "-" )
precedence = 2;
else if ( s == "=" )
precedence = 1;
return precedence;
}

```

```
}
```

// Return true if left associative; false otherwise

bool OpLeftAssoc(const std::string& s)

```
{
```

bool opLeftAssoc = false;

if (s == "*" || s == "/" ||

s == "%" || s == "+" || s == "-")

```
{
```

opLeftAssoc = true;

```
}
```

else if (s == "=" || s == "!")

```
{
```

opLeftAssoc = false;

```
}
```

return opLeftAssoc;

```
}
```

// Is token an operator

bool IsOperator(const std::string& s)

```
{
```

return s == "+" || s == "-" ||

s == "/" || s == "*" ||

s == "!" || s == "%" ||

s == "=";

```
}
```

// Is token a function argument separator eg comma

bool IsComma(const std::string& s)

```
{
```

```
return s == “,”;
```

```
}
```

```
// Convert string into all uppercase
```

```
std::string UpperCase( std::string input )
```

```
{
```

```
for ( std::string::iterator it = input.begin();
```

```
it != input.end();
```

```
++it )
```

```
*it = toupper(*it);
```

```
return input;
```

```
}
```

```
// Is token PI
```

```
bool IsPi( const std::string& s )
```

```
{
```

```
if ( UpperCase( s ) == “PI” )
```

```
return true;
```

```
return false;
```

```
}
```

```
// Is token Euler’s constant
```

```
bool IsE( const std::string& s )
```

```
{
```

```
if ( UpperCase( s ) == “E” )
```

```
{
```

```
return true;
```

```
}
```

```
return false;
```

```
}
```

```
// Is the token a function
```

```
bool IsFunction( const std::string& s )
```

```
{
```

```
std::string str = UpperCase( s );
```

```
bool isFunction = false;
```

```
if ( str.find( “^” ) != std::string::npos ||
```

```
str.find( “SIN” ) != std::string::npos ||
```

```
str.find( “COS” ) != std::string::npos ||
```

```
str.find( “TAN” ) != std::string::npos ||
```

```
str.find( “LN” ) != std::string::npos ||
```

```
str.find( “LOG” ) != std::string::npos ||
```

```
str.find( “EXP” ) != std::string::npos ||
```

```
str.find( “POW” ) != std::string::npos ||
```

```
str.find( “SQRT” ) != std::string::npos )
```

```
{
```

```
isFunction = true;
```

```
}
```

```
return isFunction;
```

```
}
```

```
// Is the number a float
```

```
bool IsFloat( const std::string& s )
```

```
{
```

```
std::istringstream iss( s );
```

```
float f;
```

```
iss >> std::noskipws >> f;
```

```
return iss.eof() && !iss.fail();
```

```
}
```

```
// Is the string a number
```

```
bool IsNumber( const std::string& s )
```

```
{
```

```
std::string::const_iterator it = s.begin();
```

```
while ( it != s.end() && std::isdigit(*it, std::locale() ) )
```

```
{
```

```
++it;
```

```
}
```

```
return !s.empty() && it == s.end();
```

```
}
```

```
ExpressionParser::ExpressionParser( const std::string& input )
```

```
{
```

```
m_strInput = input;
```

```
}
```

```
// Determine if matching number of left and right parentheses
```

```
bool ExpressionParser::MatchingParetheses()
```

```
{
```

```
const size_t nLeft = std::count( m_strInput.begin(),
```

```
m_strInput.end(), '(');
```

```
const size_t nRight = std::count( m_strInput.begin(),
```

```
m_strInput.end(), ')');
```

```
return nLeft == nRight && !m_strInput.empty();
```

```
}
```

```

// Split selected text into delimited vector array of strings
void ExpressionParser::Tokenize( std::list<std::string>& tokens,
const std::string& delimiter )
{
// Insert whitespaces before and after each special characters
size_t size = sizeof( charSet ) / sizeof( std::string );
for ( int i = 0; i < static_cast<int>( size ); i++ ) {
std::string s = charSet[ i ];
ReplaceAll( m_strInput, s, " " + s + " " );
}

size_t next_pos = 0;
size_t init_pos = m_strInput.find_first_not_of( delimiter, next_pos );
while ( next_pos != std::string::npos &&
init_pos != std::string::npos ) {
next_pos = m_strInput.find( delimiter, init_pos );
std::string token = m_strInput.substr( init_pos, next_pos - init_pos );
tokens.push_back( token );
init_pos = m_strInput.find_first_not_of( delimiter, next_pos );
}

std::string firstToken = tokens.front();
if ( firstToken == "-" ) {
std::list<std::string>::iterator it = tokens.begin();
it++;
if ( it == tokens.end() ) return;
std::string nextToken = *( it );
if ( IsNumber( nextToken ) || IsFloat( nextToken ) ) {
tokens.pop_front();
tokens.front() = firstToken + nextToken;
}
}

```

```

else if ( nextToken == "(" || IsFunction( nextToken ) ) {
tokens.front() = firstToken + "1";
tokens.insert( it, "*" );
}
// minus minus is a plus
else if ( nextToken == "-" && firstToken == "-" ) {
std::list<std::string>::iterator nit = it;
std::advance ( nit, -1 );
tokens.erase( it );
tokens.erase( nit );
}
}
// Deal with minus sign after opening parenthesis or operator
typedef std::list<std::string>::iterator t_iter;
std::string prevToken = "";
for ( t_iter it = tokens.begin(); it != Prev( tokens.end() ); it++ ) {
std::string token = *it;
std::list<std::string>::iterator nit = it;
std::advance ( nit, 1 );
if ( nit == tokens.end() ) break;
std::string ntoken = *nit;
if ( token == "-" && prevToken == "(" ) {
if ( IsNumber( ntoken ) || IsFloat( ntoken ) ) {
tokens.erase( nit );
*it = "-" + ntoken;
token = *it;
}
}
else if ( token == "-" &&
( IsOperator( prevToken ) || prevToken == "^" || prevToken == "%" ) ) {
// Minus minus becomes a plus
if ( token == "-" && prevToken == "-" ) {

```

```

std::list<std::string>::iterator nit = it;
std::list<std::string>::iterator nnit = nit;
nnit++;
std::advance ( nit, -1 );
tokens.erase( it );
*nit = "+";
std::list<std::string>::iterator pnit = nit;
std::advance ( pnit, -1 );

if ( IsOperator( *pnit ) || *pnit == "(" )
tokens.erase( nit );

token = *nnit; it = nnit;

if ( it == Prev( tokens.end() ) ) break;
}
else if ( IsNumber( ntoken ) ||
IsFloat( ntoken ) ||
IsFunction( ntoken ) ) {
bool exit = false;
if ( nit == Prev( tokens.end() ) ) exit = true;
tokens.erase( nit );
*it = "-" + ntoken;
token = *it;
if ( exit ) break;
}
else if ( ntoken == "(" ) {
*it = "-1";
token = *it;
tokens.insert( nit, "*" );
}
}

```



```

prevToken = token;
}
// Deal with minus sign before opening parenthesis
prevToken = ""; t_iter prevIt;
for ( t_iter it = tokens.begin(); it != tokens.end(); it++ ) {
    std::string token = *it;
    if ( token == "(" && prevToken == "-" ) {
        tokens.insert( it, "1" );
        tokens.insert( it, "*" );
    }
    prevToken = token;
    prevIt = it;
}
}

```

```

// Replace all instances of selected string with replacement string
void ExpressionParser::ReplaceAll( std::string& str, const std::string& from,
const std::string& to )
{
    size_t start_pos = 0;

    while( (start_pos = str.find(from, start_pos)) != std::string::npos)
    {
        str.replace(start_pos, from.length(), to);
        start_pos += to.length();
    }
}

```

```

// Deduce the numerical result from the RPN string passed to it
bool ExpressionParser::Evaluate(
const std::vector<std::string>& rpn,
std::string& result )

```

```

{
typedef std::vector<std::string>::const_iterator rpn_iter;

std::stack<std::string> stack;

for ( rpn_iter it = rpn.begin(); it != rpn.end(); it++ )
{
std::string token = *it;
if( IsNumber( token ) || IsFloat( token ) ||
IsPi( token ) || IsE( token ) )
{
if ( IsPi( token ) )
{
std::stringstream s;
s << pi;
token = s.str();
}
else if ( IsE( token ) )
{
std::stringstream s;
s << e;
token = s.str();
}
stack.push( token );
}
else if( IsOperator( token ) || IsFunction( token ) )
{
double result = 0.0;
unsigned int nargs = OpArgCount( UpperCase( token ) );
bool isUnary = false;
unsigned int stackArgs = stack.size();
std::vector<double> args;

```

```

if( stackArgs < nargs)
{
// For dealing with unary '-' or unary '+'
if ( stackArgs == 1 &&
nargs == 2 && ( token == "+" || token == "-" ) )
{
std::string value = stack.top();
result = strtod( value.c_str(), NULL );
stack.pop(); isUnary = true;
}
else { // (Error) Insufficient values in the expression.
return false;
}
}
else
{
// Else, Pop the top n values from the stack.
while ( nargs > 0 )
{
std::string value = stack.top();
double d = strtod( value.c_str(), NULL );
args.push_back( d );
stack.pop(); nargs--;
}
}
// Evaluate the operator, with the values as arguments.
if ( IsOperator( token ) && !isUnary )
{
double d2 = args[ 0 ];
double d1 = args[ 1 ];
if ( token == "+" ) {
result = d1 + d2;

```

```

}
else if ( token == "-" ) {
result = d1 - d2;
}
else if ( token == "*" ) {
result = d1 * d2;
}
else if ( token == "/" ) {
result = d1 / d2;
}
else if ( token == "%" ) {
int i2 = (int) args[ 0 ];
int i1 = (int) args[ 1 ];
double ireresult = Modulo( i1, i2 );
result = ireresult;
}
}
else if ( IsFunction( token ) )
{
double d0 = args[ 0 ];
std::string capToken = UpperCase( token );
double mult = token.find( "-" ) != std::string::npos ? -1 : 1;
if ( capToken.find( "SIN" ) != std::string::npos ) {
result = sin( d0 );
}
else if ( capToken.find( "COS" ) != std::string::npos ) {
result = cos( d0 );
}
else if ( capToken.find( "TAN" ) != std::string::npos ) {
result = tan( d0 );
}
else if ( capToken.find( "LN" ) != std::string::npos ) {

```

```

result = log( d0 );
}
else if ( capToken.find( "LOG" ) != std::string::npos ) {
result = log10( d0 );
}
else if ( capToken.find( "EXP" ) != std::string::npos ) {
result = exp( d0 );
}
else if ( capToken.find( "^" ) != std::string::npos ) {
double d2 = args[ 0 ]; double d1 = args[ 1 ];
if ( d1 < 0 ) mult = -1.0;
result = pow( (double) d1, d2);
}
else if ( capToken.find( "POW" ) != std::string::npos ) {
double d2 = args[ 0 ]; double d1 = args[ 1 ];
result = pow( d1, d2);
}
else if ( capToken.find( "SQRT" ) != std::string::npos ) {
result = sqrt( d0 );
}
result *= mult;
}
// Push the returned results, if any, back onto the stack
if ( result == (long) result )
{
result = long( result );
}

std::stringstream s; s << result;
stack.push( s.str() );
}
}

```

```

if ( stack.size() == 1 )
{
    result = stack.top();

    double res = strtod( result.c_str(), NULL );
    if ( res == (long) res )
    {
        long lres = (long) res;
        std::stringstream s;
        s << lres;
        result = s.str();
    }

    return true;
}

return false; // (Error) The user input has too many values
}

// Convert infix expression format into reverse Polish notation
bool ExpressionParser::InfixToRPN( std::vector<std::string>& inputs )
{
    typedef std::list<std::string>::const_iterator tok_iter;

    std::list<std::string> infix;
    std::stack<std::string> stack;
    std::queue<std::string> outputQueue;
    Tokenize( infix, " " );
    bool success = true;
    for ( tok_iter it = infix.begin(); it != infix.end(); it++ )
    {
        std::string token = *it;

```

```
if ( IsNumber( token ) ||  
IsFloat( token ) ||  
IsPi( token ) ||  
IsE( token ) )  
{  
outputQueue.push( token );  
}  
else if ( IsFunction( token ) )  
{  
stack.push( token );  
}  
else if ( IsComma( token ) )  
{  
std::string stackToken = stack.top();  
while ( stackToken != "(" )  
{  
outputQueue.push( stackToken );  
stack.pop(); stackToken = stack.top();  
}  
if ( stackToken == "(" )  
{  
success = true;  
}  
else  
{  
success = false;  
}  
}  
else if ( IsOperator( token ) )  
{  
while( !stack.empty() &&  
IsOperator( stack.top() ) &&
```

```

( ( OpLeftAssoc( token ) && OpPrecedence( token ) == OpPrecedence( stack.top() ) )
||
( OpPrecedence( token ) < OpPrecedence( stack.top() ) ) ) ) {
std::string stackToken = stack.top();
stack.pop(); outputQueue.push( stackToken );
}
stack.push( token );
}
else if ( token == "(" )
{
stack.push(token);
}
else if ( token == ")" )
{
while ( !stack.empty() && stack.top() != "(" )
{
outputQueue.push( stack.top() );
stack.pop();
}
if ( !stack.empty() )
{
std::string stackToken = stack.top();
if ( stackToken != "(" )
success = false;
}
else
{
return false;
}
stack.pop();
if ( !stack.empty() )
{
std::string stackToken = stack.top();

```



```

if ( IsFunction( stackToken ) )
{
    outputQueue.push( stackToken );
    stack.pop();
}
}
}
}

while ( !stack.empty() )
{
    outputQueue.push( stack.top() );
    stack.pop();
}

while ( !outputQueue.empty() ) {
    std::string token = outputQueue.front();
    inputs.push_back( token );
    outputQueue.pop();
}

return success;
}

```

main.cpp

```

#include "ExpressionParser.h"
#include <assert.h>
#include <sstream>

```

```

// Print iterators in a generic way
template<typename T, typename InputIterator>
void Print( const std::string& message,
const InputIterator& itbegin,
const InputIterator& itend,

```

```

const std::string& delimiter)
{
std::cout << message << std::endl;
std::copy(itbegin, itend,
std::ostream_iterator<T>(std::cout, delimiter.c_str()));
std::cout << std::endl;
}

```

```

std::pair<std::string, double> tests[] =
{
std::make_pair<std::string, double>( “((2*(6-1))/2)*4”, 20.0 ),
std::make_pair<std::string, double>( “-11 ^ -7”, 5.13158e-08 ),
std::make_pair<std::string, double>( “-1*- sin( Pi / -2)”, -1.0 ),
std::make_pair<std::string, double>( “-8 + 5”, -3.0 ),
std::make_pair<std::string, double>( “2—2”, 4.0 ),
std::make_pair<std::string, double>( “34.5*(23+1.5)/2”, 422.625 ),
std::make_pair<std::string, double>( “3/2 + 4*(12+3)”, 61.5 ),
std::make_pair<std::string, double>( “pow(2, 3)”, 8 ),
std::make_pair<std::string, double>( “((2*(6-1))/2)*4”, 20 ),
std::make_pair<std::string, double>( “ln(2)+3^5”, 243.693147181 ),
std::make_pair<std::string, double>( “-8+3”, -5 ),
std::make_pair<std::string, double>( “2.5^3”, 15.625 ),
std::make_pair<std::string, double>( “SQRT(4 )”, 2 ),
std::make_pair<std::string, double>( “5 + (-1 + 2 )”, 6 ),
std::make_pair<std::string, double>( “-(2+3)*(4*-10-1)+100”, 305 ),
std::make_pair<std::string, double>( “(2+3)*-(4*10-1)+100”, -95 ),
std::make_pair<std::string, double>( “1 - (-2^2) - 1”, 4 ),
std::make_pair<std::string, double>( “1 - (—2^2) — 1”, -2 ),
std::make_pair<std::string, double>( “(4*-10-1)”, -41 ),
std::make_pair<std::string, double>( “-(2+3)*(4*-10-1)”, 205 ),
std::make_pair<std::string, double>( “-3/2 + 4*-( 12+3)”, -61.5 ),
std::make_pair<std::string, double>( “-3/2 + 4*(-12-3)”, -61.5 ),

```

```
std::make_pair<std::string, double>( “-3/2 + -4*(-12-3)”, 58.5 ),  
std::make_pair<std::string, double>( “1—cos(PI)”, 0.0 ),  
};
```

```
double string_to_double( const std::string& s )  
{  
    std::istringstream i(s);  
    double x;  
    if (!(i >> x))  
        return 0;  
    return x;  
}
```

```
int main(int argc, char** argv)  
{  
    std::string result;  
    size_t test_size = sizeof( tests ) / sizeof( tests[ 0 ] );  
  
    for ( int i = 0; i < test_size; ++i )  
    {  
        std::string originalInput = tests[ i ].first;  
        double expected_result = tests[ i ].second;  
  
        Print<char, std::string::iterator>(  
            “Input expression:”,  
            originalInput.begin(),  
            originalInput.end(), ”” );  
  
        ExpressionParser parser( originalInput );  
  
        if ( !parser.MatchingParentheses() )  
        {
```

```

std::cout << "Error: mismatched parentheses or empty input"
<< std::endl;
return 1;
}

std::vector<std::string> RPN;
if ( parser.InfixToRPN( RPN ) )
{
Print<std::string, std::vector<std::string>::const_iterator>(
"RPN tokens: ", RPN.begin(),
RPN.end(),
" " );

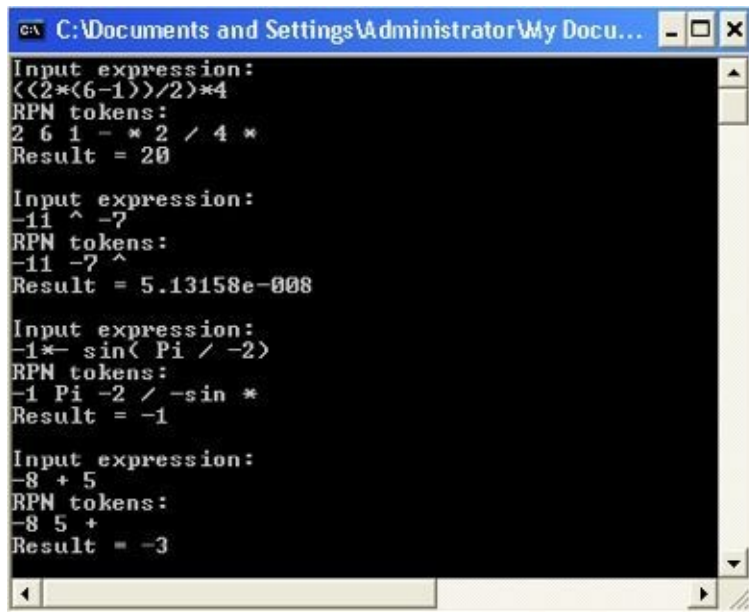
std::string str_result;

if ( parser.Evaluate( RPN, str_result ) )
{
std::cout << "Result = " << str_result << std::endl;
std::cout << std::endl;
double result = string_to_double( str_result );

// Compare expected vs actual to within a precision: 1
assert( fabs(result-expected_result) < 0.001 );
}
}
else
{
std::cout << "Error: mismatched parentheses" << std::endl;
}
}
return 0;
}

```

The result of running through each of the example test inputs and then comparing their expected output with the actual output is here:



```
C:\Documents and Settings\Administrator\My Docu...
Input expression:
<<2*(6-1)>>/2)*4
RPN tokens:
2 6 1 - * 2 / 4 *
Result = 20

Input expression:
-11 ^ -7
RPN tokens:
-11 -7 ^
Result = 5.13158e-008

Input expression:
-1*- sin( Pi / -2)
RPN tokens:
-1 Pi -2 / -sin *
Result = -1

Input expression:
-8 + 5
RPN tokens:
-8 5 +
Result = -3
```

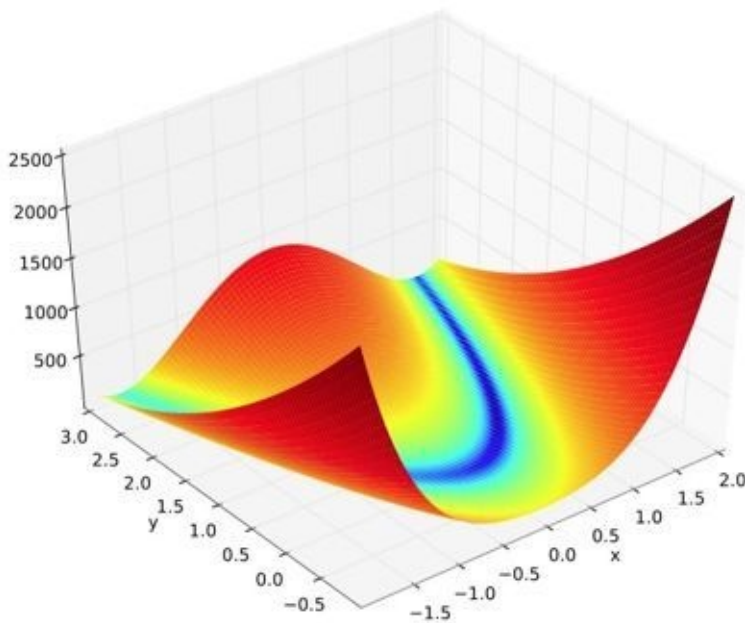
Chapter Summary

As a result of completing this lesson, you now have:

- A working example of a mathematical expression parser based on the shunting-yard algorithm.
- A better understanding of how to convert pseudocode (code written for human understanding, not a computer) into working C++ code.

Chapter 7: A Genetic Algorithm Function Optimizer

An example is presented here on how an algorithm employing standard genetic operators crossover, mutation and selection can be applied to optimize a standard mathematical function, such as the Rosenbrock function:



The Rosenbrock function is a non-convex function used to test the efficiency and effectiveness of optimization algorithms introduced by Howard H. Rosenbrock in 1960. The Rosenbrock function is defined by:

$$f(x, y) = (1 - x)^2 + 100(y - x^2)^2$$

As shown in the diagram, the global minimum lies inside a long, narrow, parabolic shaped flat valley. Locating the valley vicinity is fairly straightforward. Converging to the global minimum, located at:

$$(x, y) = (1, 1) \text{ where } f(1, 1) = 0$$

is trickier. This application employs standard genetic operators crossover, mutation and selection as applied to chromosome representations of floating-point numbers. There are a number of possible ways of representing real numbers using binary arrays. In this application, the IEEE 754 standard is one such method, where the floating-point number is represented in 32 bits. Experiments using Gray codes are also presented. In short, The first bit represents the sign (0 = plus, 1 = minus); the following 8 bits store the biased exponent (bias=127) and the remaining 23 bits represent the fractional part (mantissa). An in-depth description of how these operators work is beyond the scope of this post and there is a wealth of material already available. The code samples given here should be reasonably self-explanatory.

The 32-bit binary encodings are implemented using the **Chromosome** class as a means of housing the binary values and performing other simple get/set tasks and so forth. The **Population** class maintains a population of **Chromosome** objects and applies the standard genetic operators crossover, mutation and selection on them. It also generates an initial arbitrary population of chromosomes with random values and has other helper functions with which to convert between binary and decimal representations of the binary strings:

The **GeneticAlgorithm** class runs the genetic algorithm operators and applies them to the population of chromosomes with the intention of progressively improving the overall solution fitness.

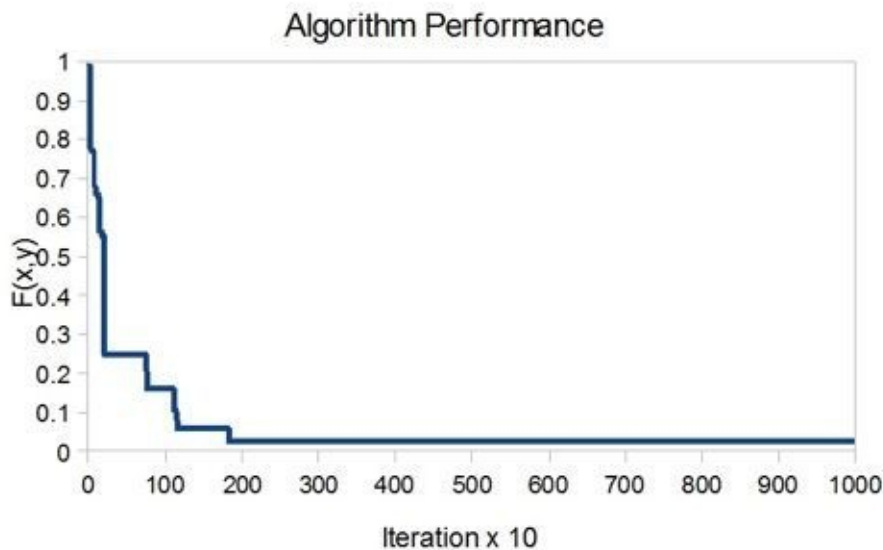
Running the Genetic Algorithm using IEEE 754 encodings

I have run a number of simple experiments to see how well the algorithm converges over 10,000 iterations. For the first experiment (using the IEEE 754 bit string representation for floating-point numbers) the algorithm parameters used were as follows:

```
encoding_type = 1; // IEEE 754
crossover_rate = 50;
mutation_rate = 5;
population_size = 100;
tournament_size = population_size / 4;
```

I use tournament selection as the preferred ‘survival of the fittest’ mechanism. The code

has a simple logger enabling you to dump results to a text file. In this example I enabled it return the best fitness found so far at every generation. A graph showing the best value found at every 10th iteration:



The global minimum for this particular function is 0.0 and in this run the algorithm got it down to 0.02. As you can see, no further reduction takes place for the next 800 or so iterations.

Feel free to try out this code and improve upon it, or apply it to your own area of interest.

Running the Genetic Algorithm using Gray codes

As a follow up to this experiment, I have included an additional means of interpreting the binary chromosomes presented to the genetic algorithm. For the next experiment I used a Gray encoding, whereby two successive values differ by only one bit. For a 32-bit Gray code, I used the first bit as the sign bit, so that a '0' corresponds to a plus and a '1' corresponds to a minus.

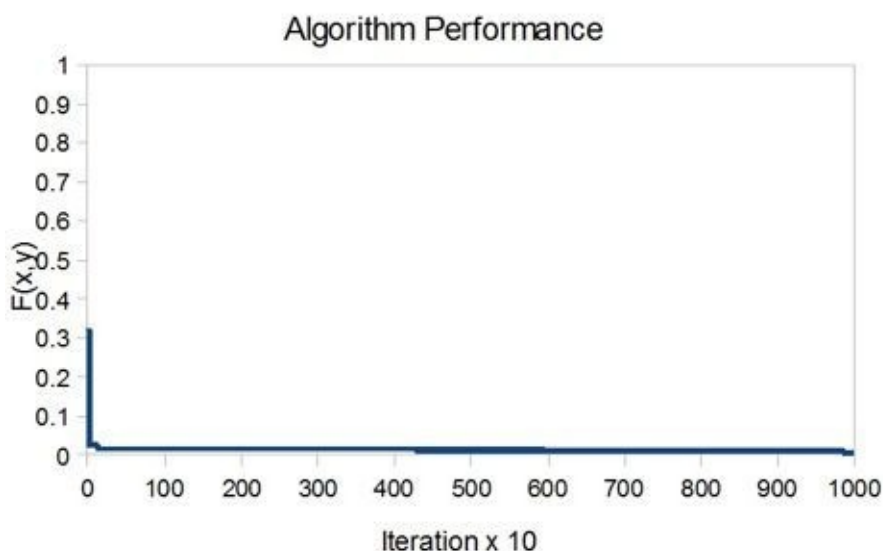
For the remaining bits, I extract the decimal value and then divide this by $2^n - 1$ to get the real number in the interval 0 to 1.

After doing a some experiments with parameter values I found that a nice combination seemed to be:

```
encoding_type = 0; // Gray code
crossover_rate = 70;
mutation_rate = 5;
population_size = 100;
tournament_size = population_size / 4;
```

My perception is that the algorithm performance using Gray codes seemed to converge somewhat more easily than those of IEEE 754.

However, this observation is only on the basis of the few simple experiments I have done so far. The algorithm in this case managed to get the value of $f(x,y)$ down to 0.00538112 ($x = 0.932924, y = 0.873316$) within the same number of iterations. A plot for this run is shown below.



Full Code Listings

Chromosome.h

```
#pragma once
```

```
const int chrSize = 64;
```

```
class Chromosome
```

```
{
```

```
public:
```

```
Chromosome(void);  
~Chromosome(void);
```

```
void SetFitness( const double& value );  
void SetChromosome( const int& index, const unsigned char& value );  
unsigned char GetChromosome( const int& index );  
double GetFitness() const;  
int size() const;  
  
void Print( const int& index ) const;
```

```
private:
```

```
unsigned char chr[ chrSize ];  
double fitness;  
};
```

```
GeneticAlgorithm.h
```

```
#pragma once
```

```
#include "Population.h"
```

```
#include "Log.h"
```

```
class GeneticAlgorithm
```

```
{
```

```
public:
```

```
GeneticAlgorithm(void);
```

```
~GeneticAlgorithm(void);
```

```
void Initialize( const int& enc,  
const int& crate,  
const int& mrate,  
const int& psize,  
const int& iter,  
const int& csize,  
const int& tsize,  
const std::string& path );  
void Run();
```

```
private:
```

```
void CreatePopulation();  
double Evaluate();  
void Crossover();  
void Mutate();  
void Select();
```

```
void SetParameters( const int& enc,  
const int& crate,  
const int& mrate,  
const int& psize,  
const int& iter,  
const int& csize,  
const int& tsize );
```

```
void LogResult( const double& result,  
const int& iter,  
const int& count );
```

```
private:
```

```
int encoding;  
int mutationRate;  
int crossoverRate;  
int populationSize;  
int numberIterations;  
int chromosomeSize;  
int tournamentSize;
```

```
int bestFitnessIndex;  
double bestFitness;  
float best_x;  
float best_y;
```

```
Population pop;  
Log log;  
};
```

Log.h

```
#pragma once  
#include <fstream>
```

```
class Log  
{  
public:
```

```
Log();  
Log( char* );  
~Log();  
void Write( char* txt );  
void Open( const char* );
```

```
private:  
std::ofstream m_stream;  
};
```

Population.h

```
#pragma once  
#include <vector>  
#include <string>  
#include "Chromosome.h"
```

```
const double infinity = 9999999999999;
```

```
class Population
```

```
{  
public:  
enum Encoding {  
GRAY = 0,  
IEEE_754,  
};
```

```
Population(void);
```

```
~Population(void);
```

```
void SetChromosomeEncoding( const int& type );
```

```
void SetChromosomeSize( const int& size );
```

```
void CreateRandomPopulation( const int& size );
```

```
void Crossover( const int& index1,
```

```
const int& index2,
```

```
const int& point );
```

```
void Crossover( const int& index1,
```

```
const int& index2,
```

```
const int& point1,  
const int& point2 );  
void Mutation( const int& index );  
double EvaluatePopulation( float& bx, float& by );  
double CalcChromosomeFitness( const int& index,  
float& xv,  
float& yv);  
double GetChromosomeFitness( const int& index ) const;  
void CopyChromosome( const int& source, const int& dest );
```

```
private:
```

```
Chromosome* CreateRandomChromosome();  
std::string GetXstring( Chromosome* chr );  
std::string GetYstring( Chromosome* chr );  
float GetFloat32_IEEE754( std::string Binary );  
float GetFloat32_Gray( std::string Binary );  
int Binary32ToHex( std::string Binary );  
double CalculateFitnessFunction( const float& x,  
const float& y );  
double CalcChromosomeFitness_IEEE754( const int& index,  
float& xv, float& yv);
```

```
double CalcChromosomeFitnessGray( const int& index,  
float& xv, float& yv );  
std::string Gray2Bin( std::string gray );  
long Bin2Dec( std::string bin );
```

```
private:
```

```
std::vector< Chromosome* > pop;  
int chrSize;  
int encoding;  
};
```

Chromosome.cpp

```
#include "Chromosome.h"
```

```
#include <iostream>
```

```
// Constructor
```

```
Chromosome::Chromosome(void) {}
```

```
// Destructor
```

```
Chromosome::~~Chromosome(void) {}
```

```
// Set chromosome element
```

```
void Chromosome::SetChromosome( const int& index, const unsigned char& value )  
{  
if ( index < 0 || index >= chrSize ) return;  
chr[ index ] = value;  
}
```

```
// Get chromosome element
```

```
unsigned char Chromosome::GetChromosome( const int& index )  
{  
unsigned char element = chr[ index ];  
return element;  
}
```

```
// Set fitness of chromosome
```

```
void Chromosome::SetFitness( const double& value )  
{  
fitness = value;  
}
```



```

// Get chromosome fitness
double Chromosome::GetFitness() const
{
    return fitness;
}

// Get number of elements in the chromosome
int Chromosome::size() const
{
    return chrSize;
}

// Output the chromosome and its fitness
void Chromosome::Print( const int& index ) const
{
    std::string str;
    for ( int i = 0; i < chrSize; i++ ) {
        unsigned char value = chr[ i ];
        str.append( value == 0 ? "0" : "1" );
    }
    std::cout << index << "\t" << str.c_str() << "\t" << fitness << std::endl;
}

```

GeneticAlgorithm.cpp

```

#include "GeneticAlgorithm.h"
#include <sstream>
#include <stdlib.h>
#include <math.h>

const std::string filepath = "C:\\dump\\best.txt";

GeneticAlgorithm::GeneticAlgorithm(void)

```

```
{  
// Give it some default parameters  
encoding = 0;  
mutationRate = 5;  
crossoverRate = 90;  
populationSize = 100;  
numberIterations = 10000;  
chromosomeSize = 64;  
tournamentSize = populationSize / 5;  
bestFitness = infinity;  
}
```

```
GeneticAlgorithm::~~GeneticAlgorithm(void)  
{}
```

```
// Initialize parameters, generate population etc  
void GeneticAlgorithm::Initialize( const int& enc,  
const int& crate,  
const int& mrate,  
const int& psize,  
const int& iter,  
const int& csize,  
const int& tsize,  
const std::string& path )  
{  
SetParameters( enc, crate, mrate, psize, iter, csize, tsize );  
CreatePopulation();  
log.Open( path.c_str() );  
}
```

```
// Run the genetic algorithm  
void GeneticAlgorithm::Run()
```

```

{
for ( int i = 0; i < numberIterations; i++ ) {
LogResult( Evaluate(), i, 10 );
Select();
Crossover();
Mutate();
}
}

// Evaluate fitnesses of population chromosomes
double GeneticAlgorithm::Evaluate()
{
float bx = -1;
float by = -1;
double best = pop.EvaluatePopulation( bx, by );
if ( best < bestFitness ) {
bestFitness = best;
best_x = bx;
best_y = by;
}
return bestFitness;
}

// Apply crossover to selected chromosome pairs
void GeneticAlgorithm::Crossover()
{
for ( int i = 0; i < populationSize; i++ ) {
int r = rand() % 100;

if ( r < crossoverRate ) {
int index1 = rand() % populationSize;
int index2 = rand() % populationSize;

```

```

while ( index1 == index2 ) {
index2 = rand() % populationSize;
}

// Get crossover points
// Point1: 0 - 31
int point1 = rand() % chromosomeSize / 2;

// Point1: 32 - 64
int point2 = chromosomeSize / 2 +
rand() % ( chromosomeSize / 2 );

while ( point1 == point2 ) {
point2 = chromosomeSize / 2 +
rand() % ( chromosomeSize / 2 );
}

if ( point1 > point2 ) {
int tmp = point1;
point1 = point2;
point2 = tmp;
}

// Do 1-point crossover
pop.Crossover( index1, index2, point1, point2 );
}
}
}

// Mutate selected chromosomes
void GeneticAlgorithm::Mutate()
{

```

```
for ( int i = 0; i < populationSize; i++ ) {  
    int r = rand() % 100;  
    if ( r < mutationRate ) {  
        pop.Mutation( i );  
    }  
}  
}
```

// Select population chromosomes according to fitness

// Using a pairwise tournament selection mechanism

void GeneticAlgorithm::Select()

{

// For each pair of chromosomes selected

int i = 0;

while (i < tournamentSize) {

// Get the chromosome pair for tournament selection

int index1 = rand() % populationSize;

int index2 = rand() % populationSize;

while (index1 == index2) {

index2 = rand() % populationSize;

}

double fitness1 = fabs(pop.GetChromosomeFitness(index1));

double fitness2 = fabs(pop.GetChromosomeFitness(index2));

// We seek to find [x,y] that minimizes this function

// The bigger the value returned, the lower its fitness

if (fitness1 > fitness2) {

// Copy chromosome 1 elements into chromosome 2

pop.CopyChromosome(index2, index1);

}

```

else {
// Copy chromosome 2 elements into chromosome 1
pop.CopyChromosome( index1, index2 );
}
i++;
}
}

// Set mutation rate, population size etc
void GeneticAlgorithm::SetParameters( const int& enc,
const int& crate,
const int& mrate,
const int& psize,
const int& iter,
const int& csize,
const int& tsize )
{
mutationRate = mrate;
crossoverRate = crate;
populationSize = psize;
numberIterations = iter;
chromosomeSize = csize;
tournamentSize = tsize;
pop.SetChromosomeEncoding( enc );
}

// Create initial random population of chromosomes
void GeneticAlgorithm::CreatePopulation()
{
pop.CreateRandomPopulation( populationSize );
}

```

```
// Log the value to a text file
void GeneticAlgorithm::LogResult( const double& result,
const int& iter,
const int& count )
{
if ( iter % count == 0 ) {
std::stringstream ss;
ss << iter << "\t" << result << "\t" << best_x << "\t" << best_y;
log.Write( (char*) ss.str().c_str() );
}
}
```

Log.cpp

```
#include "Log.h"
```

```
Log::Log()
```

```
{}
```

```
Log::Log( char* filename )
```

```
{
```

```
m_stream.open( filename, std::fstream::app );
```

```
}
```

```
void Log::Write( char* txt )
```

```
{
```

```
m_stream << txt << std::endl;
```

```
}
```

```
Log::~~Log()
```

```
{
```

```
m_stream.close();
```

```
}
```

```
// Open the log file for writing
```

```
void Log::Open( const char* filename )
```

```
{
```

```
m_stream.open( filename, std::fstream::app );
```

```
}
```

Population.cpp

```
#include "Population.h"
```

```
#include <bitset>
```

```
#include <iostream>
```

```
#include <math.h>
```

```
#include <stdlib.h>
```

```
const int divisor_32 = 4294967295; // 2 ^ 32 - 1
```

```
const int divisor_31 = 2147483647; // 2 ^ 31 - 1
```

```
Population::Population(void)
```

```
{
```

```
// Set default chromosome size
```

```
chrSize = 64;
```

```
}
```

```
Population::~~Population(void)
```

```
{
```

```
int size = (int) pop.size();
```

```
for ( int i = 0; i < size; i++ ) {
```

```
Chromosome* chr = pop.at( i );
```

```
if ( chr ) {
```

```
delete chr;
```



```

}
}
pop.clear();
}

// Set the type of chromosomal encoding used: IEEE 754, Gray etc
void Population::SetChromosomeEncoding( const int& type )
{
    encoding = type;
}

// Create initial arbitrary population of chromosomes
void Population::CreateRandomPopulation( const int& size )
{
    for ( int i = 0; i < size; i++ ) {
        Chromosome* chr = CreateRandomChromosome();
        pop.push_back( chr );
    }
}

// Apply one-point crossover to selected chromosome pair
void Population::Crossover( const int& index1, const int& index2,
    const int& point )
{
    if ( point < 0 || point >= chrSize ) return;
    Chromosome* chr1 = pop.at( index1 );
    Chromosome* chr2 = pop.at( index2 );

    for ( int i = point; i < chrSize; i++ ) {
        unsigned char v1 = chr1->GetChromosome( i );
        unsigned char v2 = chr2->GetChromosome( i );
        chr1->SetChromosome( index1, v2 );
    }
}

```

```
chr2->SetChromosome( index1, v1 );  
}  
}
```

// Apply one-point crossover to selected chromosome pair

```
void Population::Crossover( const int& index1, const int& index2,  
const int& point1, const int& point2 )  
{  
if ( point1 < 0 || point1 >= chrSize ) return;  
if ( point2 < 0 || point2 >= chrSize ) return;  
int p1 = point1;  
int p2 = point2;  
if ( p1 > p2 ) {  
int tmp = p1;  
p1 = p2;  
p2 = tmp;  
}  
}
```

```
Chromosome* chr1 = pop.at( index1 );  
Chromosome* chr2 = pop.at( index2 );
```

// Do crossover on x portion of chromosome:

// either before or after point p1

```
int start = 0;  
int end = p1;  
if ( rand() % 100 < 50 ) {  
start = p1;  
end = chrSize / 2;  
}  

```

```
for ( int i = start; i < end; i++ ) {  
unsigned char v1 = chr1->GetChromosome( i );
```

```
unsigned char v2 = chr2->GetChromosome( i );  
chr1->SetChromosome( i, v2 );  
chr2->SetChromosome( i, v1 );  
}
```

```
// Do crossover on x portion of chromosome
```

```
// either before or after point p2
```

```
start = 0;
```

```
end = p2;
```

```
if ( rand() % 100 < 50 ) {
```

```
start = p2;
```

```
end = chrSize;
```

```
}
```

```
for ( int i = start; i < end; i++ ) {
```

```
chr1->SetChromosome( i, chr2->GetChromosome( i ) );
```

```
chr2->SetChromosome( i, chr1->GetChromosome( i ) );
```

```
}
```

```
}
```

```
// Apply mutation to selected chromosome: x part or y part
```

```
void Population::Mutation( const int& index )
```

```
{
```

```
Chromosome* chr = pop.at( index );
```

```
bool xpart = rand() % 100 < 50 ? true : false;
```

```
int start = xpart ? 0 : chrSize / 2;
```

```
int end = xpart ? chrSize / 2 : chrSize;
```

```
for ( int i = start; i < end; i++ ) {
```

```
unsigned char value = chr->GetChromosome( i );
```

```
value = value == 0 ? 1 : 0;
```

```
chr->SetChromosome( i, value );
```

```
}
```

}

// Evaluate the population fitnesses

double Population::EvaluatePopulation(float& bx, float& by)

{

double totalFitness = 0.0;

//double aveFitness = 0.0;

double bestFitness = infinity;

int bestFitnessIndex = 0;

for (int i = 0; i < (int) pop.size(); i++) {

float x, y;

double fitness = CalcChromosomeFitness(i, x, y);

Chromosome* chr = pop.at(i);

chr->SetFitness(fitness);

// Output the chromosome (optional - comment out if desired)

chr->Print(i);

totalFitness += fitness;

// Store best solution

if (i == 0) bestFitness = fitness;

if (fitness < bestFitness) {

bestFitness = fitness;

bestFitnessIndex = i;

bx= x;

by= y;

}

}

std::cout << "\n\n";

//aveFitness = totalFitness / pop.size();

```
return bestFitness;
```

```
}
```

```
// Create an arbitrary random chromosome
```

```
Chromosome* Population::CreateRandomChromosome()
```

```
{
```

```
Chromosome* chr = new Chromosome();
```

```
// Randomize chromosome elements
```

```
for ( int i = 0; i < chr->size(); i++ )
```

```
{
```

```
unsigned char value = rand() % 100 < 50 ? 0 : 1;
```

```
chr->SetChromosome( i, value );
```

```
}
```

```
return chr;
```

```
}
```

```
// Calculate chromosome fitness according to chromosome encoding used
```

```
double Population::CalcChromosomeFitness(
```

```
const int& index,
```

```
float& xv,
```

```
float& yv)
```

```
{
```

```
if ( encoding == GRAY )
```

```
{
```

```
return CalcChromosomeFitnessGray( index, xv, yv );
```

```
}
```

```
else
```

```
{
```

```
return CalcChromosomeFitness_IEEE754( index, xv, yv );  
}  
}
```

// Get fitness of selected chromosome according to IEEE 754 coding

```
double Population::CalcChromosomeFitness_IEEE754(
```

```
const int& index,
```

```
float& xv,
```

```
float& yv )
```

```
{
```

```
// Get the chromosome elements
```

```
Chromosome* chr = pop.at( index );
```

```
// Put the first 32 bits (X) into a string
```

```
std::string xstr = GetXstring( chr );
```

```
// Get x value by converting from IEEE 754 into decimal
```

```
float x = GetFloat32_IEEE754( xstr );
```

```
// Put the next 32 bits (Y) into a string
```

```
std::string ystr = GetYstring( chr );
```

```
// Get y value by converting from IEEE 754 into decimal
```

```
float y = GetFloat32_IEEE754( ystr );
```

```
double fitness = CalculateFitnessFunction( x, y );
```

```
// Return the chromosome fitness
```

```
xv = x;
```

```
yv = y;
```

```
return fitness;
```

```

}

// Get fitness of selected chromosome according to Gray coding
double Population::CalcChromosomeFitnessGray(
const int& index,
float& xv,
float& yv )
{
// Get the chromosome elements
Chromosome* chr = pop.at( index );

// Put the first 32 bits (X) into a string
std::string xstr = GetXstring( chr );

// Get x value by converting from Gray into decimal
float x = GetFloat32_Gray( xstr );

// Put the next 32 bits (Y) into a string
std::string ystr = GetYstring( chr );

// Get y value by converting from Gray into decimal
float y = GetFloat32_Gray( ystr );
double fitness = CalculateFitnessFunction( x, y );

// Return the chromosome fitness
xv = x; yv = y;

return fitness;
}

// Set the size of the chromosome
void Population::SetChromosomeSize( const int& size )

```

```

{
chrSize = size;
}

// Get the 'X' string portion of the chromosome
std::string Population::GetXstring( Chromosome* chr )
{
std::string xstr;

for ( int i = 0; i < chrSize / 2; i++ )
{
unsigned char value = chr->GetChromosome( i );
xstr.append( value == 0 ? "0" : "1" );
}

return xstr;
}

// Get the 'Y' string portion of the chromosome
std::string Population::GetYstring( Chromosome* chr )
{
std::string ystr;
int start = chrSize / 2;

for ( int i = start; i < chrSize; i++ )
{
unsigned char value = chr->GetChromosome( i );
ystr.append( value == 0 ? "0" : "1" );
}

return ystr;
}

```



```

// Convert 32-bit binary into the decimal
// IEEE 754 encoding for 32 bit binary string
float Population::GetFloat32_IEEE754( std::string Binary )
{
    int HexNumber = Binary32ToHex( Binary );
    bool negative = !(HexNumber & 0x80000000);
    int exponent = (HexNumber & 0x7f800000) >> 23;
    int sign = negative ? -1 : 1;

    // Subtract 127 from the exponent
    exponent -= 127;

    // Convert mantissa into decimal using last 23 bits
    int power = -1;
    float total = 0.0;

    for ( int i = 0; i < 23; i++ )
    {
        int c = Binary[ i + 9 ] - '0';
        total += (float) c * (float) pow( 2.0, power );
        power--;
    }
    total += 1.0;
    float value = sign * (float) pow( 2.0, exponent ) * total;
    return value;
}

// Convert 32-bit gray encoding into the decimal equivalent
float Population::GetFloat32_Gray( std::string Binary )
{
    // Get sign bit
    int sign = Binary.at( 0 ) == '0' ? 1 : -1;

```

```

// Get remaining string
std::string rem = Binary.substr( 1 );
std::string bin = Gray2Bin( rem );
long bin_val = Bin2Dec( bin );
float val = (float) bin_val / (float) divisor_31;
val *= sign;

```

```

return val;
}

```

```

// Convert the 32-bit binary encoding into hexadecimal
int Population::Binary32ToHex( std::string Binary )
{
std::bitset<32> set(Binary);
int hex = set.to_ulong();
return hex;
}

```

```

// Calculate the overall fitness, f(x, y)
double Population::CalculateFitnessFunction( const float& x,
const float& y )
{
// Rosenbrock:  $(1-x)^2 + 100(y-x*x)^2$ 
// (see http://en.wikipedia.org/wiki/Rosenbrock\_function)

double fitness = ( pow( (double)( 1.0 - x ), 2 ) ) +
100 * ( pow( (double) ( y - ( x * x ) ), 2 ) ) ;

return fitness;
}
// Get fitness of selected chromosome

```

```

double Population::GetChromosomeFitness( const int& index ) const
{
// Get the chromosome
Chromosome* chr = pop.at( index );

return chr->GetFitness();
}

// Copy the contents of the source chromosome into destination
void Population::CopyChromosome( const int& source, const int& dest )
{
// Get the chromosomes
Chromosome* chr1 = pop.at( source );
Chromosome* chr2 = pop.at( dest );

// Copy elements and fitness of source chromosome
// into the destination chromosome
for ( int i = 0; i < chrSize; i++ ) {
// Get source chromosome element
unsigned char value = chr1->GetChromosome( i );
// Write this element value into the destination element
chr2->SetChromosome( i, value );
}

// Set the destination chromosome fitness
double fitness = chr1->GetFitness();
chr2->SetFitness( fitness );
}

// Convert gray encoding into binary
std::string Population::Gray2Bin( std::string gray )
{

```

```

int len = gray.length();
std::string bin = gray;

for ( int i = len - 2; i >= 0; i— )
{
    int bk = bin.at( i + 1 ) - '0';
    int gi = gray.at( i + 1 ) - '0';

    if ( bk == 0 )
    {
        bin[ i ] = gi == 1 ? '1' : '0';
    }
    else
    {
        bin[ i ] = gi == 1 ? '0' : '1';
    }
}

return bin;
}

// Binary to int
long Population::Bin2Dec( std::string bin )
{
    char * ptr;
    long parsed = strtol( bin.c_str(), & ptr, 2 );

    return parsed;
}

```

main.cpp

```
#include "Log.h"
#include <sstream>
#include "GeneticAlgorithm.h"

const int encoding_type = 0;
const int crossover_rate = 70;
const int mutation_rate = 5;
const int population_size = 100;
const int number_iterations = 10000;
const int chromosome_size = 64;
const int tournament_size = population_size / 4;

int main()
{
// Run the GA!
GeneticAlgorithm ga;

ga.Initialize( encoding_type,
crossover_rate,
mutation_rate,
population_size,
number_iterations,
chromosome_size,
tournament_size,
"C:\\dump\\best.txt" );
ga.Run();

return 0;
}
```

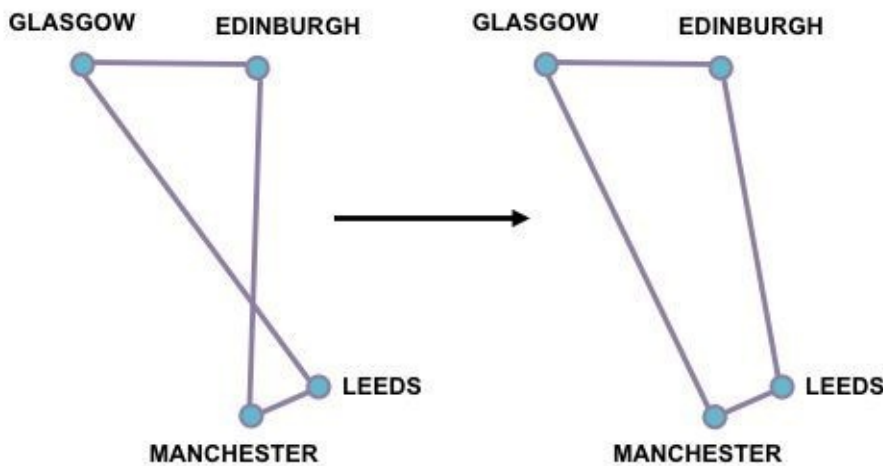
Chapter Summary

To summarize, this master lesson provides:

- A practical example of how to develop a genetic algorithm application in C++ and apply it to a difficult optimization problem to generate better solutions over successive iterations.
- A demonstration of how it is possible to improve the performance of a genetic algorithm by experimenting with binary encodings such as IEEE 754 and Gray and by varying the rates of genetic operator crossover, mutation and selection.

Chapter 8: Application of 2-opt to the Traveling Salesman Problem

A 2-opt is a simple local search algorithm for solving 'traveling salesman' type problems. Its operation is simple: it modifies a route that crosses over itself, reordering it so there are no longer any crossovers and therefore a shorter overall route:



This can be a useful technique for finding reasonable solutions to traveling salesman problems with hundreds or even thousands of node locations.

The Mechanism by which the 2-opt swaps a given route between nodes i and k to create a new route:

1. add route[0] to route[$i-1$] in order to the new route
2. add route[i] to route[k] in reverse order to the new route
3. add route[$k+1$] to end in order to the new route;

And as applied to the example route above, following steps 1-3:

route[0 to 3] : Glasgow > Edinburgh > Manchester > Leeds where $i = 1$; $k = 2$. New route:

1. Glasgow
2. Glasgow > (Manchester > Edinburgh)
3. Glasgow > Manchester > Edinburgh > (Leeds)

The complete 2-opt swap heuristic making use of the above mechanism to search every possible valid combination:

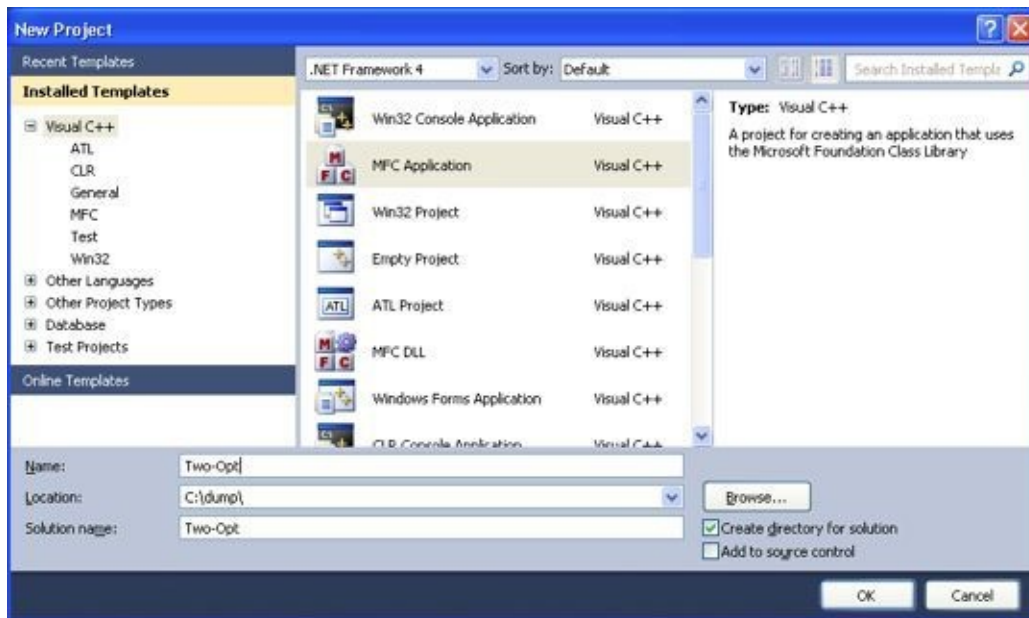
```
While ( no further improvement made )  
{  
start_again:  
best_distance = calculateTotalDistance(existing_route)  
for (i = 0; i < number of nodes eligible to be swapped - 1; i++) {  
  for (k = i + 1; k < number of nodes eligible to be swapped; k++) {  
    new_route = 2optSwap(existing_route, i, k)  
    new_distance = calculateTotalDistance(new_route)  
    if (new_distance < best_distance) {  
      existing_route = new_route  
      goto start_again  
    }  
  }  
}  
}
```

The creation of a Visual Studio 2010 dialog-based project to implement the optimization algorithm and present the output graphically is described, together with some results from applying the program to a standard traveling salesman test problem.

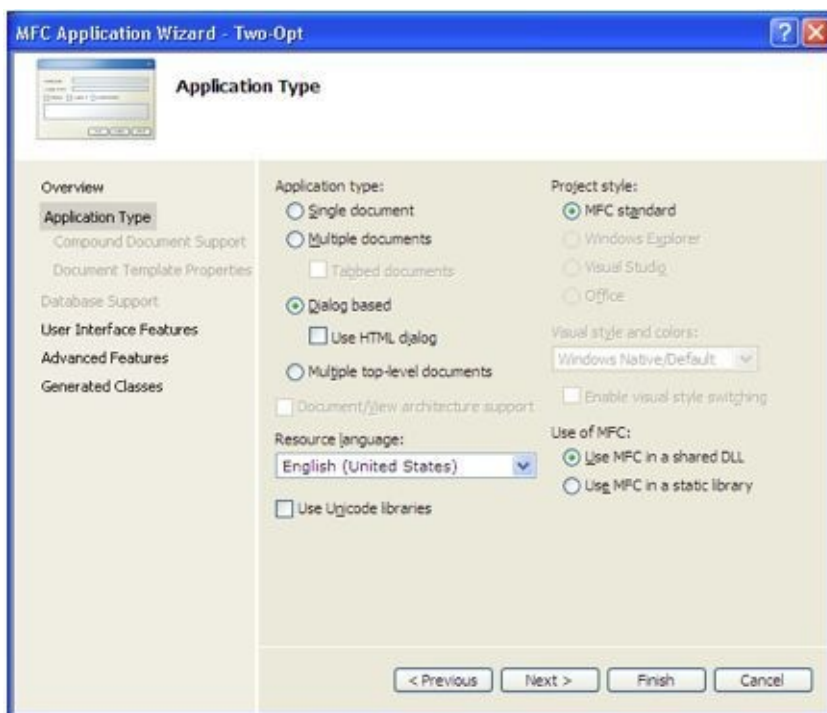
Just follow these steps:

Step 1: Create a standard dialog-based application:

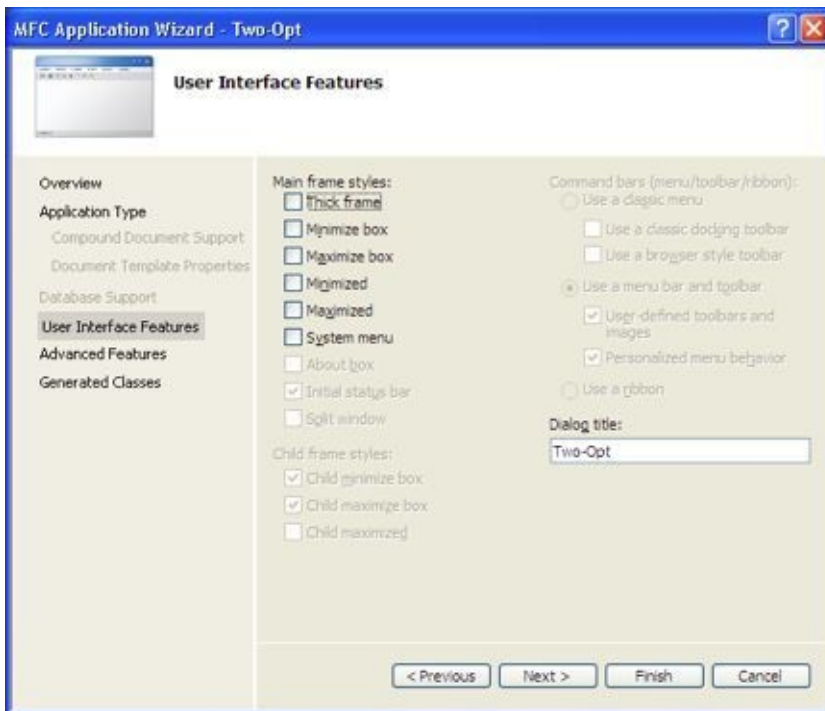
In Visual Studio select File > New > Project. Select MFC Application:



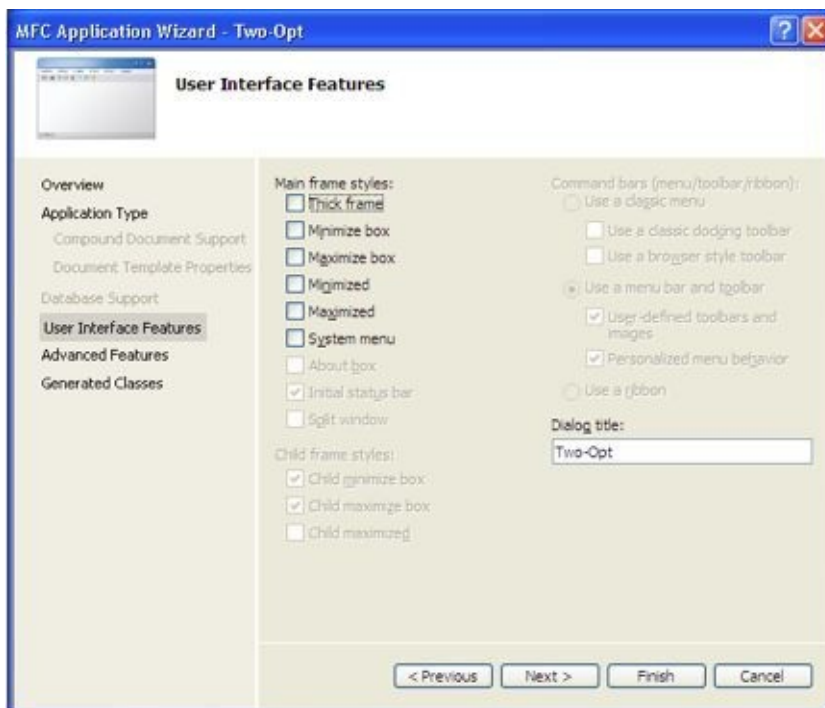
Uncheck all boxes and select a dialog-based application, then click Next:



Again uncheck all boxes:



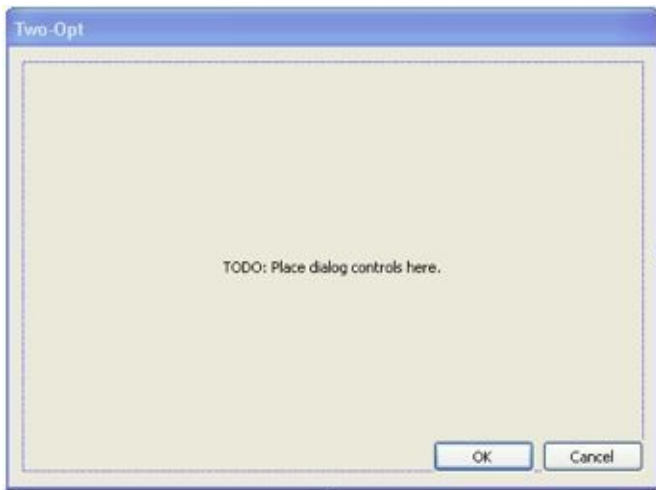
Do the same for the advanced features page:



Click Next and then click Finish.

Step 2: Create the dialog controls

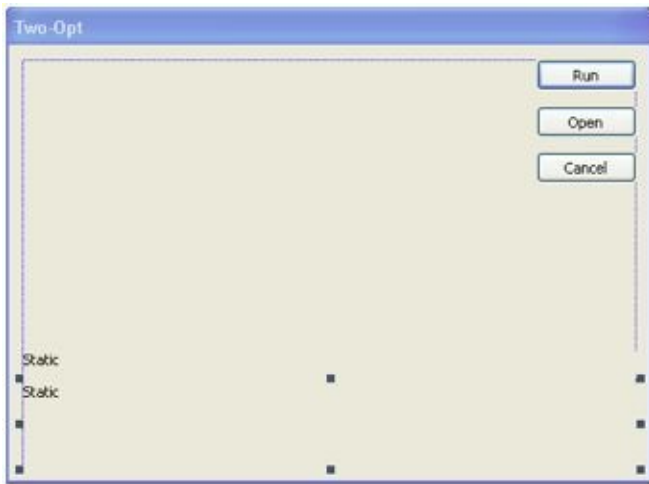
In the Resource View see how the basic dialog has been created:



Remove the “TODO: Place dialog controls here.” static control in the middle. Right-click the OK button and select Properties and rename its Caption property to Run. Change the ID property from IDOK to IDC_RUN Create one more button: from the Toolbox, select the Button control and drag it on to the dialog. Set its Caption property to Open and set its ID to IDC_OPEN. Reposition your three buttons so they are located in the top right part of the dialog:



From the Toolbox drag and resize two static controls on to the dialog:



Let the top static text control be captioned as “Total distance:” and set its ID property to IDC_DISTANCE. Then set the bottom static text Caption property as “Tour:” and set its ID to IDC_TOUR:



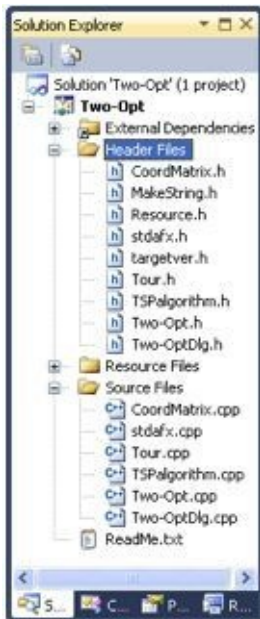
This is everything needed for creating the user interface.

Step 3: Add the additional source files

In the solution explorer, right click the project folder and select Add > New Item. Do this to create the following C++ header and source files:

1. CoordMatrix.h, CoordMatrix.cpp
2. Tour.h, Tour.cpp
3. TSPalgorithm.h, TSPalgorithm.cpp
4. MakeString.h

Your solution explorer will be updated as follows:



Insert the code for each of the source files.

Full Code Listings

CoordMatrix.h

```
#pragma once
#include <vector>
#include <map>
#include <fstream>
#include <tuple>
#include <string>
```

```
class CoordMatrix
{
public:
```

```
enum EDGE_WEIGHT_TYPE
{
```

ATT = 0

};

CoordMatrix();

~CoordMatrix();

void Initialize(std::string filepath);

int CalcPseudoEuclidDistance(

const int& x1,

const int& x2,

const int& y1,

const int& y2);

double Distance(const int& c1, const int& c2);

int size() const;

int GetX(const int& index) const;

int GetY(const int& index) const;

int GetMinX() const;

int GetMinY() const;

int GetMaxX() const;

int GetMaxY() const;

std::string GetFileTitle() const;

private:

void tokenize(std::vector<std::string>& tokens,

const std::string& text,

const std::string& delimiter);

void SetDistanceMatrix();

void Reset();

private:

```
std::vector< std::pair<int, int> > coords;  
std::map< std::pair<int,int>, double > distMatrix;  
std::string title;  
  
int minx, miny;  
int maxx, maxy;  
int edge_weight_type;  
};
```

CoordMatrix.cpp

```
#include "StdAfx.h"  
#include "CoordMatrix.h"  
#include <iostream>  
#include <fstream>  
#include <sstream>  
#include <cmath>  
  
const int infinity = 99999999;  
#define round(n) ((int)((n)+0.5))
```

```
CoordMatrix::CoordMatrix()  
{  
minx = infinity; miny = infinity;  
maxx = -infinity; maxy = -infinity;  
edge_weight_type = ATT;  
title = "";  
}
```

```
CoordMatrix::~~CoordMatrix() {}
```

```
// Initialise cost / distance matrix from input file
```

```
void CoordMatrix::Initialize( std::string filepath )
```

```
{
```

```
Reset();
```

```
minx = infinity;
```

```
miny = infinity;
```

```
maxx = infinity * -1;
```

```
maxy = infinity * -1;
```

```
std::vector< std::pair<int, int> >().swap( coords );
```

```
std::ifstream file;
```

```
file.open( filepath.c_str() );
```

```
if ( !file )
```

```
{
```

```
std::cout << "Error in opening text/tsp file\n";
```

```
return;
```

```
}
```

```
int line = 0;
```

```
while( !file.eof() )
```

```
{
```

```
line++;
```

```
char buf[ 255 ];
```

```
file.getline( buf, 128 );
```

```
std::vector<std::string> tokens;
```

```
std::string text( buf );
```

```
tokenize( tokens, text, " " );
```

```
if ( tokens.at( 0 ) == "NAME" )
```



```
{  
title = tokens.at( 2 );  
}  
if ( tokens.at( 0 ) == "EDGE_WEIGHT_TYPE" )  
{  
if ( tokens.at( 2 ) == "ATT" )  
{  
edge_weight_type = ATT;  
}  
}  
if (line < 7 || ( tokens.size() == 1 && text != "EOF"))  
continue;  
if ( text == "EOF" )  
break;
```

```
int node = atoi( tokens.at( 0 ).c_str() );  
int x = atoi( tokens.at( 1 ).c_str() );  
int y = atoi( tokens.at( 2 ).c_str() );
```

```
std::pair<int,int> n( x, y );  
coords.push_back( n );
```

```
if ( x > maxx ) maxx = x;  
if ( y > maxy ) maxy = y;  
if ( x < minx ) minx = x;  
if ( y < miny ) miny = y;  
}
```

```
file.close();  
SetDistanceMatrix();  
}
```

```
// Get Euclidean distance between two cities
```

```

double CoordMatrix::Distance( const int& c1, const int& c2 )
{
    if ( c1 < c2 )
    {
        std::pair<int,int> p( c1, c2 );
        return distMatrix[ p ];
    }
    else
    {
        std::pair<int,int> p( c2, c1 );
        return distMatrix[ p ];
    }
}

```

// Reset the distance matrix

```

void CoordMatrix::Reset()
{
    std::vector< std::pair<int, int> >().swap( coords );
    std::map< std::pair<int,int>, double >().swap( distMatrix );
}

```

// Tokenize the input string

```

void CoordMatrix::tokenize( std::vector<std::string>& tokens,
    const std::string& text,
    const std::string& delimiter )
{
    size_t next_pos = 0;
    size_t init_pos = text.find_first_not_of( delimiter, next_pos );
    while ( next_pos != std::string::npos &&
        init_pos != std::string::npos ) {
        next_pos = text.find( delimiter, init_pos );
        std::string token = text.substr( init_pos, next_pos - init_pos );
    }
}

```

```
tokens.push_back( token );  
init_pos = text.find_first_not_of( delimiter, next_pos );  
}  
}
```

```
// Get number of items contained in matrix
```

```
int CoordMatrix::size() const  
{ return (int) coords.size(); }
```

```
// Get selected x coordinate
```

```
int CoordMatrix::GetX( const int& index ) const  
{  
    std::pair<int,int> n = coords.at( index );  
    return n.first;  
}
```

```
// Get selected y coordinate
```

```
int CoordMatrix::GetY( const int& index ) const  
{  
    std::pair<int,int> n = coords.at( index );  
    return n.second;  
}
```

```
// Get minimum x coordinate
```

```
int CoordMatrix::GetMinX() const  
{ return minx; }
```

```
// Get minimum y coordinate
```

```
int CoordMatrix::GetMinY() const  
{ return miny; }
```

```
// Get maximum x coordinate
```

```

int CoordMatrix::GetMaxX() const
{ return maxx; }

// Get maximum y coordinate
int CoordMatrix::GetMaxY() const
{ return maxy; }

// Get the file name
std::string CoordMatrix::GetFileTitle() const
{ return title; }

// Set up distance matrix between node pairs
void CoordMatrix::SetDistanceMatrix()
{
for ( int i = 0; i < (int) coords.size() - 1; i++ )
{
std::pair<int,int> p1 = coords.at( i );
int x1 = p1.first; int y1 = p1.second;
for ( int j = i + 1; j < (int) coords.size(); j++ )
{
std::pair<int,int> p2 = coords.at( j );
int x2 = p2.first; int y2 = p2.second;
double dist = 0.0;

if ( edge_weight_type == ATT )
{
dist = CalcPseudoEuclidDistance( x1, x2, y1, y2 );
}
else
{
dist = (double) sqrt( (double) ( x1 - x2 ) * ( x1 - x2 ) +
(double) ( y1 - y2 ) * ( y1 - y2 ) );
}
}
}
}

```

```

}
std::pair< int, int > p( i, j );
distMatrix[ p ] = dist;
}
}
}

// For edge weight type 'ATT'
// Pseudo Euclidean distance
int CoordMatrix::CalcPseudoEuclidDistance( const int& x1,
const int& x2,
const int& y1,
const int& y2 )
{
int dij = 0;
int xd = x1 - x2; int yd = y1 - y2;
double rij = sqrt( (xd*xd + yd*yd) / 10.0 );
int tij = round( rij );
if ( tij < rij )
dij = tij + 1;
else
dij = tij;

return dij;
}

```

Tour.h

```

#pragma once
#include "CoordMatrix.h"
#include <vector>
#include <set>

```

```

class Tour
{
public:
    Tour();
    Tour( const Tour& t );
    Tour& operator=( const Tour& t );

    ~Tour();

    void DoTwoOpt( const int& c1, const int& c2,
const int& c3, const int& c4 );

    double TourDistance() const;
    int TourSize() const;
    int GetCity( const int& index );
    void SetCity( const int& index, const int& value );
    void SetMatrix( CoordMatrix* mat );
    void CreateRandomTour();
    void CreateTour();
    void CreateNearestNeighbourTour();
    void Reset();
    void SetCities( const std::vector<int>& v );

private:
    double Distance( const int& c1, const int& c2 ) const;
    int GetNearestNeighbour( const int& c, std::set<int>& cset );

private:
    std::vector< int > cities;
    CoordMatrix* matrix;
};

```

Tour.cpp

```
#include "StdAfx.h"
```

```
#include "Tour.h"
```

```
#include <algorithm>
```

```
#include <functional>
```

```
Tour::Tour(){}
```

```
Tour::~~Tour(){}
```

```
Tour::Tour( const Tour& t )
```

```
{ cities = t.cities; }
```

```
Tour& Tour::operator=( const Tour& t )
```

```
{
```

```
if ( this != &t )
```

```
cities = t.cities;
```

```
return *this;
```

```
}
```

```
void Tour::DoTwoOpt( const int& c1, const int& c2,
```

```
const int& c3, const int& c4 )
```

```
{
```

```
if ( c3 == c1 || c3 == c2 || c4 == c1 || c4 == c2 )
```

```
return;
```

```
int c1old = cities.at( c1 );
```

```
int c2old = cities.at( c2 );
```

```
int c3old = cities.at( c3 );
```

```
int c4old = cities.at( c4 );
```

```
double old_distance = TourDistance();  
int tmp = cities.at( c2 );  
cities[ c2 ] = cities.at( c3 );  
cities[ c3 ] = tmp;  
double new_distance = TourDistance();
```

```
if ( new_distance > old_distance )  
{  
    cities[ c1 ] = c1old;  
    cities[ c2 ] = c2old;  
    cities[ c3 ] = c3old;  
    cities[ c4 ] = c4old;  
}  
}
```

```
// Get total distance of tour
```

```
double Tour::TourDistance() const  
{  
    double dist = 0.0;  
    int size = (int) cities.size();  
    int c1, c2;  
    for ( int i = 0; i < size - 1; i++ ) {  
        c1 = cities.at( i );  
        c2 = cities.at( i + 1 );  
        dist += Distance( c1, c2 );  
    }  
    c1 = cities.at( size - 1 );  
    c2 = cities.at( 0 );  
    dist += Distance( c1, c2 );  
    return dist;  
}
```



```
// Generate arbitrary tour  
void Tour::CreateRandomTour()  
{  
Reset();  
for ( int i = 0; i < matrix->size(); i++ )  
cities.push_back( i );  
  
random_shuffle( cities.begin() + 1, cities.end() );  
}
```

```
// Generate arbitrary tour  
void Tour::CreateTour()  
{  
Reset();  
for ( int i = 0; i < matrix->size(); i++ )  
cities.push_back( i );  
}
```

```
// Get distance bewteen selected cities  
double Tour::Distance( const int& c1,  
const int& c2 ) const  
{ return matrix->Distance( c1, c2 ); }
```

```
// Set pointer to cost / distance matrix object  
void Tour::SetMatrix( CoordMatrix* mat )  
{ matrix = mat; }
```

```
// Reset existing tour data  
void Tour::Reset()  
{ std::vector< int >().swap( cities ); }
```

```
// Return the number of cities in the tour
```

```
int Tour::TourSize() const
{ return (int) cities.size(); }
```

```
// Get the tour city, given the index passed to it
```

```
int Tour::GetCity( const int& index )
{
    int node = cities.at( index );
    return node;
}
```

```
// Get tour from the set of nearest neighbours
```

```
void Tour::CreateNearestNeighbourTour()
{
    Reset();
    std::set<int> city_set;
    std::set<int>::iterator it;

    for ( int i = 0; i < matrix->size(); i++ )
        city_set.insert( i );

    int city = 0;
    for ( int i = 1; i <= matrix->size(); i++ )
    {
        cities.push_back( city );
        it = city_set.find( city );

        city_set.erase( it );
        city = GetNearestNeighbour( city, city_set );
    }
}
```

```
int Tour::GetNearestNeighbour( const int& c, std::set<int>& cset )
```

```

{
int city = 0;
double min_dist = 99999999;
std::set<int>::iterator cit;
for ( cit = cset.begin(); cit != cset.end(); cit++ ) {
int n = *cit;
if ( n == c ) continue;
double dist = Distance( c, n );
if ( dist < min_dist ) {
city = n;
min_dist = dist;
}
}
return city;
}

void Tour::SetCities( const std::vector<int>& v )
{ cities = v; }

void Tour::SetCity( const int& index, const int& value )
{ cities[ index ] = value; }

```

TSPalgorithm.h

```

#pragma once
#include "Tour.h"
#include <string>

class Observer;

class TSPalgorithm
{

```

public:

TSPAlgorithm(void);

~TSPAlgorithm(void);

void Initialize(const int& iterations, CoordMatrix* mat);

void TwoOptSwap(const int& i, const int& k);

void Run();

int GetTourCity(const int& index);

void AddObserver(Observer* ob);

void Notify(const int& dist);

private:

void TwoOpt();

private:

Tour tour, new_tour;

int iterations;

std::vector<Observer*> ob_set;

};

class Observer

{

public:

virtual void UpdateDisplay(const float& dist) = 0;

};

TSPAlgorithm.cpp

#include "StdAfx.h"

#include "TSPAlgorithm.h"

```
TSPAlgorithm::TSPAlgorithm()
```

```
{}
```

```
TSPAlgorithm::~~TSPAlgorithm()
```

```
{}
```

```
// Set up TSP problem to run
```

```
void TSPAlgorithm::Initialize(
```

```
const int& iter,
```

```
CoordMatrix* mat )
```

```
{
```

```
tour.SetMatrix( mat );
```

```
new_tour.SetMatrix( mat );
```

```
iterations = iter;
```

```
}
```

```
// Run the optimization algorithm
```

```
void TSPAlgorithm::Run()
```

```
{
```

```
tour.Reset();
```

```
tour.CreateNearestNeighbourTour();
```

```
TwoOpt();
```

```
}
```

```
// Get the tour node, given the index
```

```
int TSPAlgorithm::GetTourCity( const int& index )
```

```
{
```

```
return tour.GetCity( index );  
}
```

```
// Notify observers of changes
```

```
void TSPAlgorithm::Notify( const int& dist )  
{  
for ( std::vector<Observer*>::iterator it = ob_set.begin();  
it != ob_set.end();  
it++ )  
{  
(*it)->UpdateDisplay( dist );  
}  
}
```

```
// Add observer
```

```
void TSPAlgorithm::AddObserver( Observer* ob )  
{  
ob_set.push_back( ob );  
}
```

```
// Do all 2-opt combinations
```

```
void TSPAlgorithm::TwoOpt()  
{  
int size = tour.TourSize();  
new_tour = tour;  
  
bool improve = true;
```

```
double best_distance, new_distance;
```

```
while ( improve )
```

```
{
```

```
best_distance = tour.TourDistance();
```

```
improve = false;
```

```
for ( int i = 0; i < size - 1; i++ )
```

```
{
```

```
for ( int k = i + 1; k < size; k++ )
```

```
{
```

```
TwoOptSwap( i, k );
```

```
new_distance = new_tour.TourDistance();
```

```
if ( new_distance < best_distance )
```

```
{
```

```
improve = true;
```

```
tour = new_tour;
```

```
best_distance = new_distance;
```

```
Notify( tour.TourDistance() );
```

```
i = size;
```

```
break;
```

```
}
```

```
}
```

```
}
```

```
}
```

```
}
```

```
void TSPalgorithm::TwoOptSwap( const int& i, const int& k )
```

```

{
for ( int c = 0; c <= i - 1; ++c )
{
new_tour.SetCity( c, tour.GetCity( c ) );
}

int dec = 0;

for ( int c = i; c <= k; ++c )
{
new_tour.SetCity( c, tour.GetCity( k - dec ) );
dec++;
}

for ( int c = k + 1; c < tour.TourSize(); ++c )
{
new_tour.SetCity( c, tour.GetCity( c ) );
}

}

```

MakeString.h

```

#include <fstream>
#include <sstream>
#include <string>

class make_string
{
public:
template <typename T>

```



```

make_string& operator<<( T const & datum )
{
buffer_ << datum;
return *this;
}

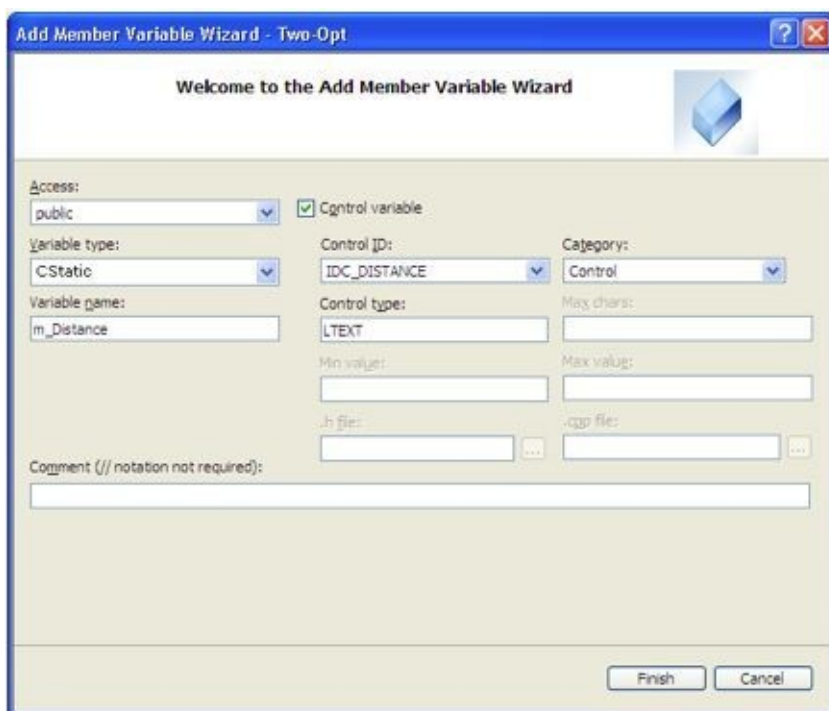
operator std::string () const
{ return buffer_.str(); }

private:
std::ostringstream buffer_;
};

```

Step 4: add the static control variables

In the Resource View, right-click the “Total Distance” static control you added and select Add Variable. Give it the variable name m_Distance:



Repeat this step for the “Tour” static control, giving it the variable name **m_Tour**. Notice that following this step will insert the **CStatic** controls in the dialog header class CTwoOptDlg.h and insert the DDX controls in CTwoOptDlg.cpp:

```

void CTwoOptDlg::DoDataExchange(CDataExchange* pDX)
{

```

```
CDialogEx::DoDataExchange(pDX);  
DDX_Control(pDX, IDC_DISTANCE, m_Distance);  
DDX_Control(pDX, IDC_TOUR, m_Tour);  
}
```

Step 5: Implement the button event handling logic

In the Resource View, double-click the Run button to generate the event handler code in CTwoOptDlg.cpp for when this button is clicked.

Once this is generated, insert the following code inside **OnBnClickedRun** to invoke the algorithm and display updates:

```
void CTwoOptDlg::OnBnClickedRun()  
{  
    alg.Run();  
    hasRun = true;  
    RedrawWindow( 0, 0, RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE  
);  
}
```

Do the same for the Open button: Double click the Open button to generate its event handler code, and insert the following code that will enable the user to open the desired *.tsp file:

```
RedrawWindow( 0, 0, RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE  
);  
}void CTwoOptDlg::OnBnClickedOpen()  
{  
    CPaintDC dc(this);  
    CRect rect;  
    GetClientRect(&rect);
```

```

// Open the file opening dialog
CFileDialog fileDlg(
TRUE,
NULL,
NULL,
OFN_HIDEREADONLY | OFN_FILEMUSTEXIST,
(LPCTSTR) “.tsp Files(*.tsp)|*.tsp|”,
this );

fileDlg.m_ofn.lpstrTitle = (LPCSTR)“Loading TSP Problem”;

INT_PTR result = fileDlg.DoModal();
CString path = fileDlg.GetPathName();

if ( result == IDOK )
{
// Reset matrix data contents
CT2CA pszConvertedAnsiString( path );
std::string path_str( pszConvertedAnsiString );
mat.Initialize( path_str );
hasRun = false;

(AfxGetMainWnd( ))->SetWindowText(mat.GetFileTitle().c_str());
}

```

Step 6: Update the OnPaint method

Update **OnPaint** method to draw the current best solution found by the algorithm:

```

void CTwoOptDlg::OnPaint()
{

```

```

// device context for painting
CPaintDC dc(this);

if (IsIconic())
{
    SendMessage(
        WM_ICONERASEBKGND,
        reinterpret_cast<WPARAM>(dc.GetSafeHdc()), 0);

    // Center icon in client rectangle
    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;

    // Draw the icon
    dc.DrawIcon(x, y, m_hIcon);
}
else
{
    if ( mat.size() < 1 ) return;

    CRect rect;
    GetClientRect(&rect);

    int rectx1 = rect.left + 20;
    int rectx2 = rect.right - 150;
    int recty1 = rect.top + 20;
    int recty2 = rect.bottom - 140;

```

```
CPen pen( PS_SOLID, 1, RGB( 0, 0, 0 ) );
```

```
dc.SelectStockObject(BLACK_BRUSH);
```

```
int tour_size = mat.size();
```

```
for ( count = 0; count < tour_size; count++ )
```

```
{
```

```
xc1 = mat.GetX( count );
```

```
yc1 = mat.GetY( count );
```

```
xn1 = (float) ( xc1 - mat.GetMinX() ) /
```

```
(float) ( mat.GetMaxX() - mat.GetMinX() );
```

```
yn1 = (float) ( yc1 - mat.GetMinY() ) /
```

```
(float) ( mat.GetMaxY() - mat.GetMinY() );
```

```
xcoord1 = rectx1 + (int) (float) ( xn1 * abs( rectx1 - rectx2 ) );
```

```
ycoord1 = recty2 - (int) (float) ( yn1 * abs( recty1 - recty2 ) );
```

```
dc.Ellipse( xcoord1 - 2, ycoord1 - 2, xcoord1 + 2, ycoord1 + 2 );
```

```
if ( hasRun && count < tour_size - 1 )
```

```
{
```

```
cc1 = alg.GetTourCity( count );
```

```
cc2 = alg.GetTourCity( count + 1 );
```

```
xc1 = mat.GetX( cc1 );
```

```
yc1 = mat.GetY( cc1 );
```

```
xc2 = mat.GetX( cc2 );
```

```
yc2 = mat.GetY( cc2 );
```

```
xn1 = (float) ( xc1 - mat.GetMinX() ) /
```

```
(float) ( mat.GetMaxX() - mat.GetMinX() );
```

```
yn1 = (float) ( yc1 - mat.GetMinY() ) /
```

```
(float) ( mat.GetMaxY() - mat.GetMinY() );  
xn2 = (float) ( xc2 - mat.GetMinX() ) /  
(float) ( mat.GetMaxX() - mat.GetMinX() );  
yn2 = (float) ( yc2 - mat.GetMinY() ) /  
(float) ( mat.GetMaxY() - mat.GetMinY() );
```

```
xcoord1 = rectx1 + (int)(float)(xn1 * abs( rectx1 - rectx2));  
ycoord1 = recty2 - (int)(float)(yn1 * abs( recty1 - recty2));  
xcoord2 = rectx1 + (int)(float)(xn2 * abs( rectx1 - rectx2));  
ycoord2 = recty2 - (int)(float)(yn2 * abs( recty1 - recty2));
```

```
dc.MoveTo( xcoord1, ycoord1 );  
dc.LineTo( xcoord2, ycoord2 );  
}  
}
```

```
if ( hasRun )  
{  
cc1 = alg.GetTourCity( tour_size - 1 );  
cc2 = alg.GetTourCity( 0 );
```

```
xc1 = mat.GetX( cc1 );  
yc1 = mat.GetY( cc1 );
```

```
xc2 = mat.GetX( cc2 );  
yc2 = mat.GetY( cc2 );
```

```
xn1 = (float) ( xc1 - mat.GetMinX() ) /  
(float) ( mat.GetMaxX() - mat.GetMinX() );  
yn1 = (float) ( yc1 - mat.GetMinY() ) /  
(float) ( mat.GetMaxY() - mat.GetMinY() );  
xn2 = (float) ( xc2 - mat.GetMinX() ) /
```

```

(float) ( mat.GetMaxX() - mat.GetMinX() );
yn2 = (float) ( yc2 - mat.GetMinY() ) /
(float) ( mat.GetMaxY() - mat.GetMinY() );

```

```

xcoord1 = rectx1 + (int) (float) ( xn1 * abs( rectx1 - rectx2 ) );
ycoord1 = recty2 - (int) (float) ( yn1 * abs( recty1 - recty2 ) );
xcoord2 = rectx1 + (int) (float) ( xn2 * abs( rectx1 - rectx2 ) );
ycoord2 = recty2 - (int) (float) ( yn2 * abs( recty1 - recty2 ) );
dc.MoveTo( xcoord1, ycoord1 );
dc.LineTo( xcoord2, ycoord2 );
}

```

```

CDialogEx::OnPaint();
}
}

```

Step 7: Add methods to initialize observer and refresh display

Insert this code to CTwoOptDlg.cpp to initialize the observer pattern and output the details of the best tour found:

```

void CTwoOptDlg::Init( TSPalgorithm* tsp )
{
    alg.AddObserver( this );
}

```

```

void CTwoOptDlg::UpdateDisplay( const float& dist )
{
    hasRun = true;

```

```

std::string s = make_string() << "Total distance: "
<< dist;

```

```
m_Distance.SetWindowText( s.c_str() );
```

```
int tour_size = mat.size();
```

```
std::string tour = make_string() << "Tour:\n";
```

```
std::string t;
```

```
for ( int i = 0; i < tour_size; i++ )
```

```
{
```

```
t = make_string() << alg.GetTourCity( i ) + 1
```

```
<< "-";
```

```
tour.append( t );
```

```
if ( i > 0 && i % 35 == 0 )
```

```
{
```

```
tour.append( "\n" );
```

```
}
```

```
}
```

```
t = make_string() << alg.GetTourCity( 0 ) + 1;
```

```
tour.append( t );
```

```
m_Tour.SetWindowText( tour.c_str() );
```

```
RedrawWindow(
```

```
0,
```

```
0,
```

```
RDW_INVALIDATE | RDW_UPDATENOW | RDW_ERASE );
```

```
}
```

Step 8: Ensure necessary headers are present in CTwoOptDlg.cpp and update OnInitDialog

Make sure the following headers are included in CTwoOptDlg.cpp:

```
#include "stdafx.h"  
#include "Two-Opt.h"  
#include "Two-OptDlg.h"  
#include "MakeString.h"  
#include <fstream>  
#include <sstream>  
#include <string>  
#include "afxdialogex.h"
```

```
BOOL CTwoOptDlg::OnInitDialog()  
{  
    CDialogEx::OnInitDialog();  
    SetIcon(m_hIcon, TRUE); // Set big icon  
    SetIcon(m_hIcon, FALSE); // Set small icon
```

```
    Init( &alg );  
    alg.Initialize( 10000, &mat );  
    hasRun = false;  
    font.CreatePointFont(100, _T("Arial"));  
    m_Distance.SetFont(&font);  
    m_Tour.SetFont(&font);  
    GetDlgItem(IDC_DISTANCE)->SetFont(&font);  
    GetDlgItem(IDC_TOUR)->SetFont(&font);
```

```
    return TRUE; // return TRUE unless you set the focus to a control  
}
```

Step 9: Add additional variables and methods to the CTwoOptDlg class

This includes the **TSPalgorithm** and **CoordMatrix** objects and methods to initialize the observer and update the display. You can just copy and insert the complete CTwoOptDlg.h code listing shown here:

```
#pragma once
#include "TSPalgorithm.h"
#include "CoordMatrix.h"
#include "afxwin.h"

// CTwoOptDlg dialog
class CTwoOptDlg : public CDialogEx, Observer
{
// Construction
public:
CTwoOptDlg(CWnd* pParent = NULL);

// Dialog Data
enum { IDD = IDD_TWOOPT_DIALOG };
protected:
virtual void DoDataExchange(CDataExchange* pDX);

// Implementation
protected:
HICON m_hIcon;

// Generated message map functions
virtual BOOL OnInitDialog();
afx_msg void OnPaint();
afx_msg HCURSOR OnQueryDragIcon();
DECLARE_MESSAGE_MAP()
public:
afx_msg void OnBnClickedOk();
afx_msg void OnBnClickedOpen();
```

```

afx_msg void OnBnClickedRun();

void Init( TSPAlgorithm* tsp );
void UpdateDisplay( const float& dist );
private:
TSPAlgorithm alg;
CoordMatrix mat;
bool hasRun;
int cc1, cc2, xc1, xc2, yc1, yc2;
float xn1, xn2, yn1, yn2;
int xcoord1, xcoord2, ycoord1, ycoord2;
int x1, y1, count;
CStatic m_Distance, m_Tour;
LOGFONT lf;
CFont font;
};

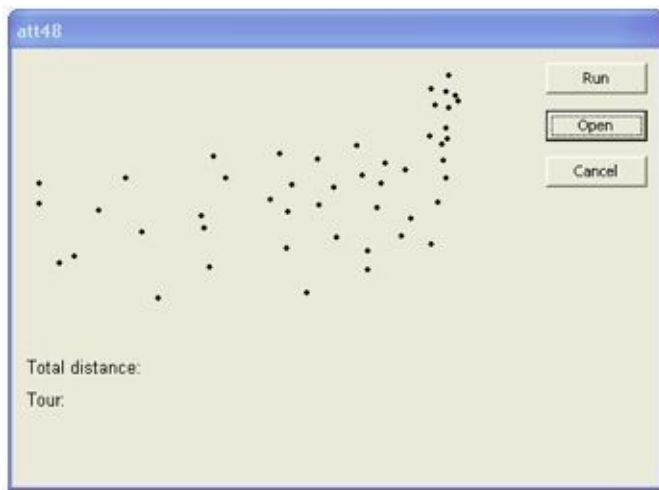
```

Once the project is completed, using it is simple: download a test problem as a *.tsp file such as Att48.tsp, representing 48 capital cities of the USA:

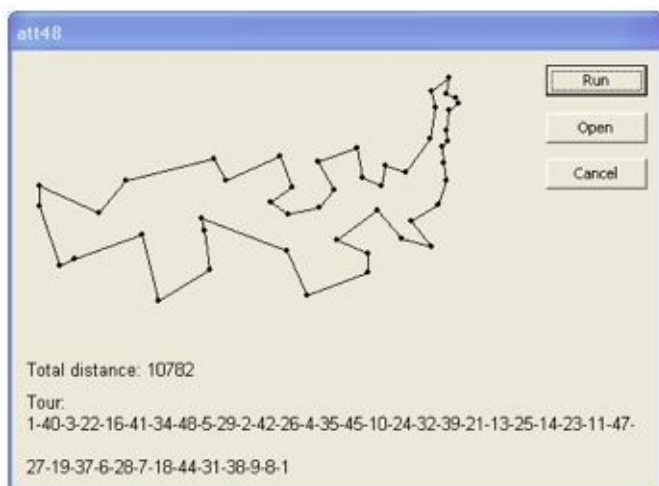
<http://www.technical-recipes.com/Downloads/att48.tsp>

Click Open to load this test problem and then click Run.

To try out the algorithm, first click Open to load the test problem. This will represent the geographical locations of cities as a series of nodes:



Upon pressing Run, the 2-opt algorithm is applied to run through all two-edge combinations and find the best solution obtained from this process. The program will periodically update the graphical layout as the algorithm finds progressively better solutions.



Chapter Summary

To summarize this master lesson, you have learned how to:

- Apply the 2-opt algorithm in C++ to generate near-optimal solutions to traveling salesman test problems.
- Create a MFC dialog application and use standard drawing objects to present the results graphically.

Chapter 9: Path finding in networks

A frequently occurring requirement in solving network optimization problems is to find all path combinations that contain no cycles between a selected pair of communicating end nodes. An additional factor in finding all combinations of paths is that the algorithm should be able to handle both directed or undirected graphs. The National Institute of Standards and Technology (NIST) online Dictionary of Algorithms and Data Structures describes this particular problem as “all simple paths” and recommends a depth-first search to find all non-cyclical paths between arbitrary pairs of nodes.

Problem definition: Find all simple paths from a starting vertex (source) to a destination vertex (sink) in a directed graph. In an undirected graph, find all simple paths between two vertices.

Network connectivity is modeled using a Graph class as the means to uniquely identify individual links, as well as add new links and check for the presence of existing links. It also involves setting a graph's directed/undirected status of added links.

The algorithm used for identifying and enumerating paths uses using a recursive depth-first search . The search avoids repeating vertices (nodes) by marking them with a boolean visited value as they are visited during the recursion, then unmarking the node just before returning from the recursive call.

DepthFirstSearch(Graph G, vertex v)

{

Set vertex v as visited;

For (each neighbor vertex w of v)

{

If (vertex w is unvisited)

{

DepthFirstSearch(w);

Add edge vw to tree T

}

}

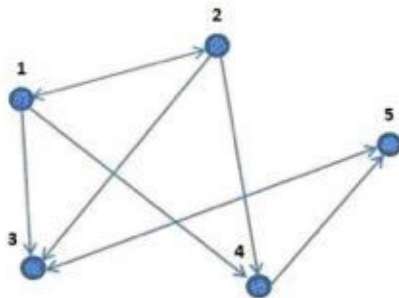
}

Practical application of the depth-first algorithm

Beginning from the start node, the depth first search algorithm examines all outgoing links and progresses by expanding the first child node of the search tree that appears. It searches progressively deeper until a target node is found, or until it encounters a node that has no children. The search then backtracks, returning to the most recent node it hasn't finished exploring.

Example 1: network topology with uni-directional links

Here is the graphical representation of a 5-node directed graph problem used in the example presented here:



In the main main program loop, the network was set as having directed edges which are inserted using calls to the Network object's **AddLink** method.

For example a directed edge exists between nodes [1,3], but not nodes [3,1], hence the single arrow between the node [1,3] pair. Directed edges exist between nodes [1,2] and nodes [2,1], hence the bi-directional link between them.

To find all possible combinations of paths between nodes [2,5] for example, we simply set the start and target nodes and feed the **GetAllPaths** method with them.

The -1 value passed to **GetAllPaths** signifies that we do not wish to filter any of the search results for maximum number of hops, but return all possible paths it finds.

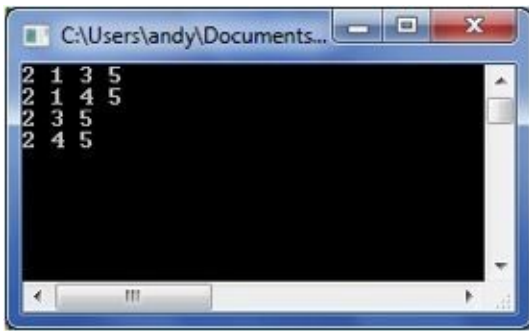
Example main.cpp usage:

```

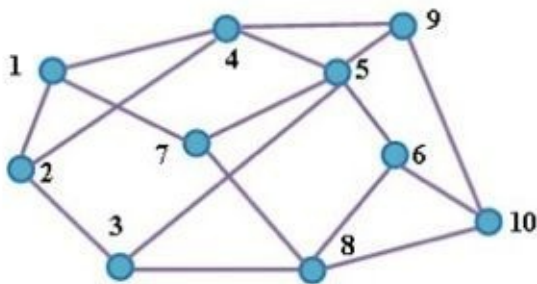
#include "Algorithm.h"
#include <iostream>
#include <vector>
const Link<int> linkset[] = { Link<int>( 1, 2 ),
Link<int>( 1, 3 ),
Link<int>( 1, 4 ),
Link<int>( 2, 1 ),
Link<int>( 2, 4 ),
Link<int>( 2, 3 ),
Link<int>( 3, 5 ),
Link<int>( 4, 5 ),
Link<int>( 5, 3 ) };
const size_t size = sizeof( linkset ) / sizeof( linkset[ 0 ] );
const std::vector<Link<int>> links (linkset, linkset + size );
int main()
{
// Create network from interconnections given
Network nw( links, false );
// Create the algorithm object
Algorithm algorithm( &nw );
// Get all paths between nodes 2 and 5
algorithm.GetAllPaths( &nw, 2, 5, -1, -1 );
// Output the set of paths found
nw.ShowPaths();
return 0;
}

```

This gives the following output for finding all paths between nodes 2 and 5:



Example 2: network topology with bi-directional links



Using a very simple tweak we can also construct network topologies with link interconnections that are bi-directional; that is, a link between [1,2] is the same as the link between [2,1]. In other words, adding link [2,1] when link [1,2] already exists will make no difference.

Simply set the bi-directional flag in the **Network** class constructor to true, and build the network the same way as shown in this main.cpp example:

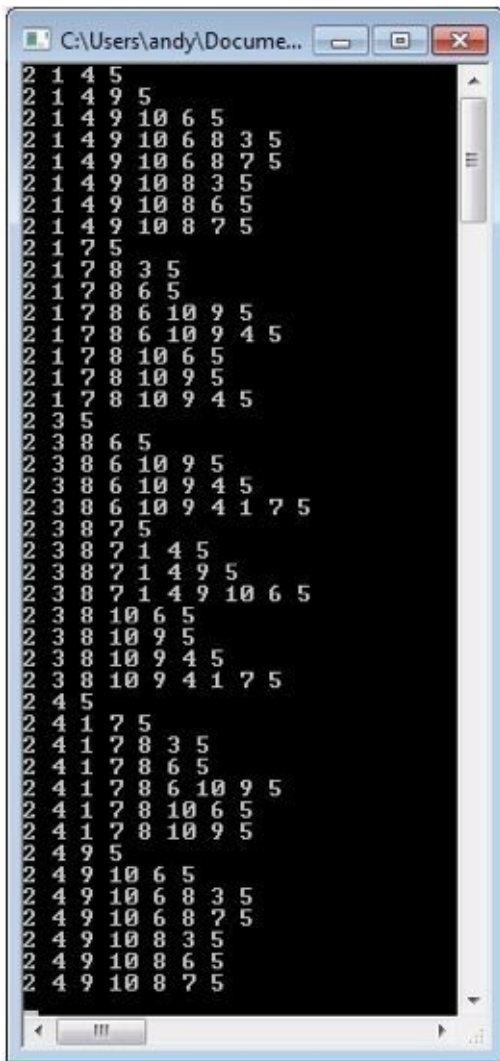
```
#include "Algorithm.h"
#include <iostream>
#include <vector>
const Link<int> linkset[] = { Link<int>( 1, 2 ),
Link<int>( 1, 4 ),
Link<int>( 1, 7 ),
Link<int>( 2, 4 ),
Link<int>( 2, 3 ),
Link<int>( 3, 5 ),
Link<int>( 3, 8 ),
Link<int>( 4, 5 ),
```

```

Link<int>( 4, 9 ),
Link<int>( 5, 6 ),
Link<int>( 5, 7 ),
Link<int>( 5, 9 ),
Link<int>( 6, 8 ),
Link<int>( 6, 10 ),
Link<int>( 7, 8 ),
Link<int>( 8, 10 ),
Link<int>( 9, 10 ) };
const size_t size = sizeof( linkset ) / sizeof( linkset[ 0 ] );
const std::vector<Link<int>> links (linkset, linkset + size );
int main()
{
// Create network from interconnections given
Network nw( links, true );
// Create the algorithm object
Algorithm algorithm( &nw );
// Get all paths between nodes 2 and 5
algorithm.GetAllPaths( &nw, 2, 5, -1, -1 );
// Output the set of paths found
nw.ShowPaths();
return 0;
}

```

This gives the following output for finding all paths between nodes 2 and 5:



Example 3: filtering by the maximum number of hops

We may wish to restrict paths found to those whose maximum hop count is no larger than a set maximum, say 4 hops. By using another simple tweak we can achieve this. Simply set the **max_hops** argument that is passed to **Algorithm::GetAllPaths** to some value greater than zero. The **min_hops** argument is kept as default -1:

```
algorithm.GetAllPaths( &nw, 2, 5, 4, -1 );
```

Re-running the program with the same network as in example 2 gives the following paths (restricted to a maximum of 4 hops):



Example 4: Labeling network nodes arbitrarily

A possible enhancement could be to allow the nodes to be labeled as string variables as well as integers. Consider the following 9-node network representing UK towns and cities:



Given that we want to label the nodes to give a more meaningful description, it is possible to maintain a two-way mapping between the integer used to uniquely identify each node created and the text label used to describe it. Supposing we wish to construct the above network and find all possible paths between Bristol and Edinburgh for an undirected

representation of the graph. The usage would be as follows:

```
#include "Algorithm.h"
#include <iostream>
#include <vector>

const Link<std::string> linkset[] =
{ Link<std::string>( "Bristol", "London" ),
  Link<std::string>( "Bristol", "Cardiff" ),
  Link<std::string>( "Oxford", "Cardiff" ),
  Link<std::string>( "Glasgow", "Manchester" ),
  Link<std::string>( "Manchester", "Leeds" ),
  Link<std::string>( "Manchester", "Cardiff" ),
  Link<std::string>( "Glasgow", "Edinburgh" ),
  Link<std::string>( "Edinburgh", "Newcastle" ),
  Link<std::string>( "Leeds", "London" ),
  Link<std::string>( "Newcastle", "Leeds" ) };

const size_t size = sizeof( linkset ) / sizeof( linkset[ 0 ] );

const std::vector<Link<std::string>> links (linkset, linkset + size );

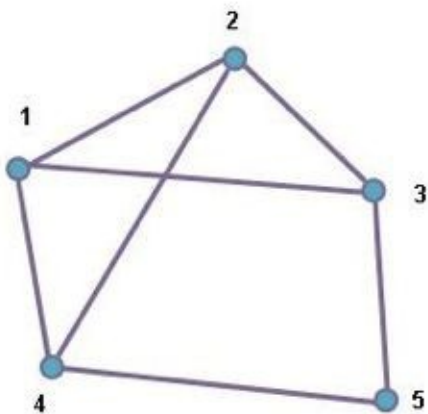
int main()
{
    // Create network from interconnections given
    Network nw( links, true );
    // Create the algorithm object
    Algorithm algorithm( &nw );
    // Get all paths between nodes Bristol and Bath
    algorithm.GetAllPaths( &nw, "Bristol", "Edinburgh", -1, -1 );
    // Output the set of paths found
    nw.ShowPaths();
    return 0;
}
```

This gives the following output:

```
C:\Documents and Settings\User\My Documents\Visual Studio 2010...
Bristol Cardiff Manchester Glasgow Edinburgh
Bristol Cardiff Manchester Leeds Newcastle Edinburgh
Bristol London Leeds Manchester Glasgow Edinburgh
Bristol London Leeds Newcastle Edinburgh
```

Example 5: Finding all paths between the same node

You may wish to find all paths between the same node. Take the following network with bi-directional links:



To find all paths that start from the node 2 and return to node 2, an example is shown here:

```
#include "Algorithm.h"
#include <iostream>
#include <vector>
const Link<int> linkset[] = { Link<int>( 1, 2 ),
Link<int>( 1, 3 ),
Link<int>( 1, 4 ),
Link<int>( 2, 1 ),
Link<int>( 2, 4 ),
```

```

Link<int>( 2, 3 ),
Link<int>( 3, 5 ),
Link<int>( 4, 5 ),
Link<int>( 5, 3 ) };

const size_t size = sizeof( linkset ) / sizeof( linkset[ 0 ] );
const std::vector<Link<int>> links (linkset, linkset + size );

int main()
{
// Create network from interconnections given
Network nw( links, true );

// Create the algorithm object
Algorithm algorithm( &nw );

// Get all paths between nodes 2 and 5
algorithm.GetAllPaths( &nw, 2, 2, -1, -1 );

// Output the set of paths found
nw.ShowPaths();

return 0;
}

```

This gives the list of all found paths to and from the node labelled 2:

```

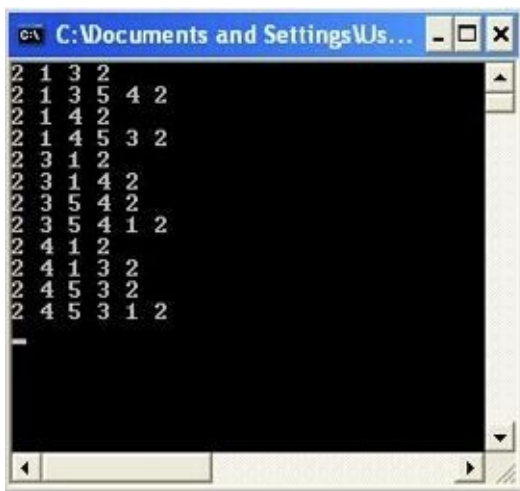
C:\Documents and Settings\WJs...
2 1 2
2 1 3 2
2 1 3 5 4 2
2 1 4 2
2 1 4 5 3 2
2 3 2
2 3 1 2
2 3 1 4 2
2 3 5 4 2
2 3 5 4 1 2
2 4 2
2 4 1 2
2 4 1 3 2
2 4 5 3 2
2 4 5 3 1 2

```

If you would rather not have paths such as 2 – 1 – 2 then specify a minimum hop count of 3:

```
algorithm.GetAllPaths( &nw, 2, 2, -1, 3 );
```

Which would give the following path list:



Full Code listings

Graph.h

```
#pragma once
```

```
#include "PriorityQueue.h"
```

```
#include <vector>
```

```
#include <memory>
```

```
class Edge
```

```
{
```

```
public:
```

```
int v, w;
```

```
double wt;
```

```
Edge(int v = -1, int w = -1, double wt = 0.0) :
```

```
v(v), w(w), wt(wt) { }
```

```
};
```



```
typedef std::shared_ptr<Edge> EdgePtr;
```

```
template <class Edge>
```

```
class DenseGRAPH
```

```
{
```

```
private:
```

```
int Vcnt, Ecnt;
```

```
bool digraph;
```

```
std::vector <std::vector <EdgePtr>> adj;
```

```
public:
```

```
DenseGRAPH(int V, bool digraph = false) :
```

```
adj(V), Vcnt(V), Ecnt(0), digraph(digraph)
```

```
{
```

```
for ( int i = 0; i < V; i++ ) {
```

```
adj[ i ].resize( V );
```

```
}
```

```
}
```

```
int V() const { return Vcnt; }
```

```
int E() const { return Ecnt; }
```

```
bool directed() const { return digraph; }
```

```
void insert( EdgePtr e )
```

```
{
```

```
int v = e->v;
```

```
int w = e->w;
```

```
if (adj[v][w] == 0) Ecnt++;
```

```
adj[v][w] = e;
```

```
if (!digraph)
```

```
adj[w][v] = e;
```

```
}
```

```
void remove(EdgePtr e)
```

```
{
```

```
int v = e->v; int w = e->w;
```

```
if (adj[v][w] != 0) Ecnt—;
```

```
adj[v][w] = 0;
```

```
if (!digraph) adj[w][v] = 0;
```

```
}
```

```
EdgePtr edge(int v, int w) const
```

```
{
```

```
return adj[v][w];
```

```
}
```

```
class adjIterator;
```

```
friend class adjIterator;
```

```
};
```

```
template <class Edge>
```

```
class DenseGRAPH<Edge>::adjIterator
```

```
{
```

```
private:
```

```
const DenseGRAPH &G;
```

```
int i, v;
```

```
public:
```

```
adjIterator(const DenseGRAPH<Edge> &G, int v) :
```

```
G(G), v(v), i(0) {}
```

```
EdgePtr beg() { i = -1; return nxt(); }
```

```
EdgePtr nxt()
```

```

{
for (i++; i < G.V(); i++)
if (G.edge(v, i))
return G.adj[v][i];
return 0;
}

bool end() const { return i >= G.V(); }
};

```

PriorityQueue.h

```

#include <memory>
#include <vector>

template <class keyType>
class PriorityQueue
{
private:
int d, N;
std::vector<int> pq, qp;
const std::vector<keyType> &a;

void Exchange(int i, int j)
{
int t = pq[i];
pq[i] = pq[j];
pq[j] = t;
qp[pq[i]] = i;
qp[pq[j]] = j;
}

```

```

void fixUp(int k)
{
while ( k > 1 && a[pq[(k+d-2)/d]] > a[pq[k]] )
{
Exchange(k, (k+d-2)/d);
k = (k+d-2)/d;
}
}

```

```

void fixDown(int k, int N)
{
int j;
while ((j = d*(k-1)+2) <= N)
{
for (int i = j+1; i < j+d && i <= N; i++)
if (a[pq[j]] > a[pq[i]])
j = i;
if (!(a[pq[k]] > a[pq[j]]))
break;
Exchange(k, j);
k = j;
}
}

```

```

public:
PriorityQueue(int N, const std::vector<keyType> &a, int d = 3) :
a(a), pq(N+1, 0), qp(N+1, 0), N(0), d(d) {}

```

```

int empty() const { return N == 0; }
void insert(int v)
{
pq[++N] = v;
qp[v] = N;
}

```

```
fixUp(N);
```

```
}
```

```
int getmin()
```

```
{
```

```
Exchange( 1, N );
```

```
fixDown(1, N-1);
```

```
return pq[N—];
```

```
}
```

```
void lower(int k)
```

```
{
```

```
fixUp(qp[k]);
```

```
}
```

```
};
```

```
Network.h
```

```
#pragma once
```

```
#include “Graph.h”
```

```
#include <vector>
```

```
#include <map>
```

```
#include <string>
```

```
typedef std::vector<int> Path;
```

```
typedef std::vector<Path> PathSet;
```

```
typedef std::vector<Path>::const_iterator Paths;
```

```
template< class T >
```

```
class Link
```

```
{
```

```
private:
```

```

T from;
T to;
double weight;
public:
Link( const T& from, const T& to, const double &weight = 1.0 ) :
from( from ), to( to ), weight( weight ) {}
const T& From() const
{ return from; }
const T& To() const
{ return to; }
const double& Weight() const
{ return weight; }
};

```

```

class Network
{
private:
DenseGRAPH<Edge>* graph;
PathSet pathset;
std::map<std::string, int > label2index;
std::map<int, std::string > index2label;
public:
Network() {}
Network( const std::vector<Link<std::string> >& links,
const bool& bi = false );
Network( const std::vector<Link<int> >& links,
const bool& bi = false );
~Network();
void AddPath( const Path& p );
void ShowPaths() const;
int NodeIndex( const std::string& label ) const;
std::vector<int> GetAdjNodeIDs( const int& n ) const;

```

```
};
```

Network.cpp

```
#include "Network.h"
```

```
#include <iostream>
```

```
#include <sstream>
```

```
#include <iterator>
```

```
#include <algorithm>
```

```
#include <set>
```

```
#include <stack>
```

```
Network::~Network()
```

```
{
```

```
if ( graph != NULL ) {
```

```
delete graph;
```

```
graph = NULL;
```

```
}
```

```
}
```

```
Network::Network( const std::vector<Link<std::string>>& links,  
const bool& bi_connected )
```

```
{
```

```
typedef std::vector<Link<std::string>>::const_iterator v_iter;
```

```
label2index.clear();
```

```
index2label.clear();
```

```
std::set<std::string> unique_id_str;
```

```
for ( v_iter it = links.begin(); it != links.end(); ++it ) {
```

```
const Link<std::string> l = (*it);
```

```
std::string first = l.From();
```

```
std::string second = l.To();
```

```

unique_id_str.insert( first );
unique_id_str.insert( second );
}

int i = 0;
for ( std::set<std::string>::const_iterator it = unique_id_str.begin();
it != unique_id_str.end();
++it )
{
std::string label = (*it);
index2label[ i ] = label;
label2index[ label ] = i;
i++;
}
std::set<int> unique_ids;
const int nodes = unique_id_str.size();
graph = new DenseGRAPH<Edge>( nodes + 1, !bi_connected );
for ( v_iter it = links.begin(); it != links.end(); ++it )
{
const Link<std::string> l = (*it);
std::string first = l.From();
std::string second = l.To();
int s = label2index.find( first )->second;
int d = label2index.find( second )->second;
double weight = l.Weight();
EdgePtr e( new Edge( s, d, weight ) );
graph->insert( e );
}
}

Network::Network( const std::vector<Link<int>>& links,
const bool& bi_connected )

```



```

{
typedef std::vector<Link<int>>::const_iterator v_iter;
label2index.clear();
index2label.clear();
std::set<std::string> unique_id_str;

for ( v_iter it = links.begin(); it != links.end(); ++it )
{
const Link<int> l = (*it);
int first = l.From();
int second = l.To();
std::string strFirst = static_cast<std::ostringstream*>(
&(std::ostringstream() << first ) )->str();
std::string strSecond = static_cast<std::ostringstream*>(
&(std::ostringstream() << second ) )->str();

unique_id_str.insert( strFirst );
unique_id_str.insert( strSecond );
}

int i = 0;

for ( std::set<std::string>::const_iterator it = unique_id_str.begin();
it != unique_id_str.end();
++it )
{
std::string label = (*it);
index2label[ i ] = label;
label2index[ label ] = i;
i++;
}
std::set<int> unique_ids;

```

```

const int nodes = unique_id_str.size();
graph = new DenseGRAPH<Edge>( nodes + 1, !bi_connected );
for ( v_iter it = links.begin(); it != links.end(); ++it ) {
const Link<int> l = (*it);
int first = l.From();
int second = l.To();
std::string strFirst = static_cast<std::ostringstream*>(
&(std::ostringstream() << first ))->str();
std::string strSecond = static_cast<std::ostringstream*>(
&(std::ostringstream() << second ))->str();

int s = label2index.find( strFirst )->second;
int d = label2index.find( strSecond )->second;
double weight = l.Weight();

EdgePtr e( new Edge( s, d, weight ) );
graph->insert( e );
}
}

void Network::AddPath( const Path& p )
{
pathset.push_back( p );
}

// Get the set of nodes adjacent to this node
std::vector< int > Network::GetAdjNodeIDs( const int& n ) const
{
std::vector< int > adj;
DenseGRAPH<Edge>::adjIterator A(*graph, n );
for ( EdgePtr e = A.beg(); !A.end(); e = A.nxt() ) {

```

```

const int s = e->v;
const int d = e->w;
if ( !graph->directed() && ( s == n || d == n ) ) {
adj.push_back( s == n ? d : s );
}
else if ( s == n ) {
adj.push_back( d );
}
}
return adj;
}
// Output the set of paths found
void Network::ShowPaths() const
{
for ( Paths paths = pathset.begin();
paths != pathset.end();
++paths )
{
Path path = *paths;
for ( std::vector<int>::const_iterator it = path.begin();
it != path.end();
++it )
{
int n = (*it);
std::string node_str = index2label.find( n )->second;
std::cout << node_str << " ";
}
std::cout << std::endl;
}
}

int Network::NodeIndex( const std::string& label ) const

```

```
{  
int n = label2index.find( label )->second;  
return n;  
}
```

Algorithm.h

```
#include "Network.h"  
#ifndef ALGORITHM_H  
#define ALGORITHM_H
```

```
class Algorithm
```

```
{
```

```
public:
```

```
Algorithm() {}
```

```
Algorithm( Network* network );
```

```
void GetAllPaths( Network* network,  
const std::string& start,  
const std::string& target,  
const int& max_hops,  
const int& min_hops );
```

```
void GetAllPaths( Network* network,  
const int& start,  
const int& target,  
const int& max_hops,  
const int& min_hops );
```

```
private:
```

```
void DepthFirst( Network* network,  
Path& visited,  
const int& end,
```

```

const int& max_hops,
const int& min_hops );

void SetNetwork( Network* network );
bool ContainsNode( Path& nodes, const int& node );
Network *nw;
int startNodeId;
};

#endif /* ALGORITHM_H */

```

Algorithm.cpp

```

#include <iostream>
#include <sstream>
#include <iterator>
#include "Algorithm.h"

Algorithm::Algorithm( Network* network )
{
SetNetwork( network );
}

void Algorithm::SetNetwork( Network* network )
{
nw = network;
}

// Algorithm to recursively search for all paths between
// chosen source - destination nodes
void Algorithm::DepthFirst(
Network* network,

```

```

Path& visited,
const int& end,
const int& max_hops,
const int& min_hops)
{
    const int back = visited.back();
    std::vector< int > adjNode = network->GetAdjNodeIDs( back );

    // Examine adjacent nodes
    for ( std::vector<int>::const_iterator node_it = adjNode.begin();
        node_it != adjNode.end();
        node_it++ )
    {
        const int node = (*node_it);

        bool startEqualTarget = ContainsNode( visited, node ) &&
startNodeId == end &&
node == startNodeId;

        if ( ContainsNode( visited, node ) && !startEqualTarget )
            continue;

        if ( node == end )
        {
            visited.push_back( *node_it );

            // Get hop count for this path
            const int size = (int) visited.size();
            const int hops = size - 1;

            if ( ( max_hops < 1 || hops <= max_hops ) && hops >= min_hops )
            {

```

```

Path path( visited.begin(), visited.begin() + size );
network->AddPath( path );
}
visited.erase( visited.begin() + hops );
break;
}
}

```

```

// in breadth-first, recursion needs to come after visiting adjacent nodes
for ( std::vector<int>::const_iterator node_it = adjNode.begin();
node_it != adjNode.end();
node_it++ )
{
int node = (*node_it);

if ( ContainsNode( visited, node ) || node == end )
continue;
visited.push_back( node );
DepthFirst( network, visited, end, max_hops, min_hops );
int n = (int) visited.size() - 1;
visited.erase( visited.begin() + n );
}
}

```

```

bool Algorithm::ContainsNode( Path& nodes, const int& node )
{
for ( std::vector<int>::const_iterator nodes_it = nodes.begin();
nodes_it != nodes.end(); nodes_it++ )
{
if ( (*nodes_it) == node ) return true;
}
return false;
}

```

```
}
```

```
void Algorithm::GetAllPaths( Network* network,  
const std::string& start,  
const std::string& target,  
const int& max_hops,  
const int& min_hops )  
{  
int s = network->NodeIndex( start );  
int d = network->NodeIndex( target );  
startNodeId = s;  
Path visited;  
visited.push_back( s );  
DepthFirst( network, visited, d, max_hops, min_hops );  
}
```

```
void Algorithm::GetAllPaths( Network* network,  
const int& start,  
const int& target,  
const int& max_hops,  
const int& min_hops )  
{  
std::string strFirst = static_cast<std::ostringstream*>(  
&(std::ostringstream() << start ) )->str();  
std::string strSecond = static_cast<std::ostringstream*>(  
&(std::ostringstream() << target ) )->str();  
  
int s = network->NodeIndex( strFirst );  
int d = network->NodeIndex( strSecond );  
  
startNodeId = s;  
Path visited;
```



```
visited.push_back( s );
```

```
DepthFirst( network, visited, d, max_hops, min_hops );
```

```
}
```

Chapter Summary

To summarize this master lesson, you have:

- Implemented a recursive algorithm to solve the “all simple paths” problem in C++.
- Made further refinements and enhancements to present the results in different ways, such as labeling nodes with meaningful names.
- Applied further adjustments such as finding all paths going out from and returning to the same node or how to constrain the paths found to a maximum number of hops.

Chapter 10: The Standard Template Library

10.1 Introduction

The Standard Template Library (STL) as introduced by Hewlett Packard to the C++ community was revolutionary. Why? Because it gave C++ a dimension and capability it previously did not have. Before the library's inception, programmers focused on classes and Object Oriented Programming (OOP) in an attempt to model real-world behaviour.

Classes and OOP helped boost the popularity of C++ immensely. But by themselves, classes lacked efficiency in handling algorithms and data structures in a generic way. And OOP was never the holy grail it was frequently made out to be – in many cases it did not truly promote code reuse and modularity. Until template-based programming was incorporated into C++, developers and companies would hand-craft their own data structures and algorithms by writing custom libraries and routines from scratch.

Now let's fast forward to the development of a new feature to receive serious attention by Alexander Stepanov (see http://en.wikipedia.org/wiki/Alexander_Stepanov) and other researchers: templates. These facilitated a library framework that at last allowed algorithms and data structures (containers) to work with arbitrary data types.

STL now comprises of five kinds of components: containers, iterators, algorithms, function objects (functors) and allocators.

Using these components can dramatically make your programs solid and robust in many ways. Here are just a few examples:

1. Arrays of any data type which can arbitrarily grow or shrink and incorporate automatic memory management as needed.
2. Access to a large number of powerful algorithms for searching and sorting data.
3. Set operations and other numerical transformations.
4. Customized code that will frequently outperform code built from scratch.

It is no exaggeration to say that STL has made possible a revolution in the way software tasks get done. It has become an integral part of any C++ programmer's arsenal of techniques by allowing unwieldy custom code to be replaced with programs that can be applied across a wide range of generic tasks. STL concepts themselves will be discussed in more detail in the following sections.

10.2 Choosing an STL Container

Standard STL containers can be generalized as belonging to:

Standard sequence containers (**std::vector**, **std::list**, **std::deque** and **std::string**)

Standard associative containers (**std::map**, **std::set**, **std::multimap**, **std::multiset**)

Given the number of different STL containers at your disposal, it is worth considering some rules of thumb for choosing a STL container appropriate to your particular scenario. If you anticipate your collection needing to hold just a few items, then it is probably fine to use **std::vector** regardless of what you intend to do. **std::vector** remains a good choice if you intend to maintain a collection of items that will change very little when compared to the number of times they are accessed. Furthermore, the **std::vector** container will provide everything you expect from an array, allowing for a straightforward collection of items that can be randomly accessed or added and removed (at the back of the container) in a dynamic and type safe way:

```
#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <iterator>

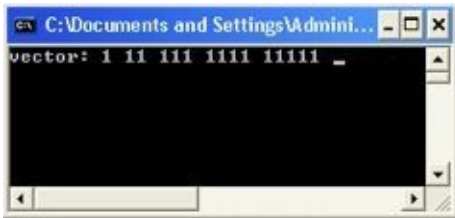
int main()
{
    const int values[] = { 1, 11, 111, 1111 };
    const size_t size = sizeof( values ) / sizeof( values[ 0 ] );
    std::vector<int> v( values, values + size );
    v.push_back( 11111 );

    std::cout << "vector: ";
    std::copy( v.begin(),
              v.end(),
              std::ostream_iterator<int>( std::cout, " " ) );
```

```
return 0;
```

```
}
```

This gives the following output:



Similar in respects to **std::vector** is **std::deque** (pronounced “deck”), short for double-ended queue. **std::deque** allows insertions at either end of the collection, ie **push_back** and **push_front**, thus allowing you to model a queue as well as a stack. But let’s move on to entirely new ground. It may be the case that the need for random access is not so important, but instead you need a container that can efficiently insert or remove items at arbitrary positions. An associative container such as **std::map** would be a definite no-no and **std::vector** would be more computationally expensive for inserting data items, especially in the middle. However, **std::list** is ideal for this kind of task. A practical application could be the tracking of all open Windows objects, or maintaining a list of processes running in the background, the the kind of situations whereby the number of insertions and deletions is likely to be substantial. A unique feature of **std::list** is its ability to splice: that is, transfer elements from one **std::list** to another.

```
#include <list>
```

```
#include <string>
```

```
#include <algorithm>
```

```
#include <iostream>
```

```
#include <iterator>
```

```
int main()
```

```
{
```

```
typedef std::list<std::string>::iterator list_it;
```

```
std::list<std::string> list1;
```

```
std::list<std::string> list2;
```

```
list1.push_front( “A” );
```

```
list1.push_back( “E” );
```

```
list1.push_back( “F” );
```

```

list1.insert( ++list1.begin(), "B" );
list2.push_front( "C" );
list2.push_front( "D" );

// Transfer (splice) elements from list2 into list1.
list_it it = list1.begin();
std::advance( it, 2 );
list1.splice( it, list2 );
std::cout << "list1: ";

std::copy( list1.begin(), list1.end(),
std::ostream_iterator<std::string>( std::cout, " " ) );

return 0;
}

```

The output is as shown:



So the **std::vector** and **std::list** containers are examples of sequence container uses that implement data item storage. Other sequence containers in this category include **std::deque**, **std::array** and **std::forward_list**. In each case, all the container items must belong to the same data type. There can never be a mix of data types such as `int` or **std::string** within the same container.

So what other uses do we have for these containers? Well, there may be times when you want to guarantee that the container stores unique values only and prevents the insertion of duplicate items. In a practical setting this might include maintaining a list of students enrolled on a course, or a collection of unique social security numbers. Therefore an associative container such as **std::set** would be a reliable means of guaranteeing the uniqueness of inserted data items, provided the ordering of these unique items is of no importance.

The **std::set** container permits easy insertion and lookup of unique values by way of the **insert** and **find** methods respectively. That is because insert will not add a new item if the value of that item is already in the set, and new elements when inserted are automatically sorted in ascending order. Furthermore **std::set**, like other associative containers, has certain performance guarantees: the time required to both find and insert an item is proportional to $\log N$, where N is the number of items.

```
#include <set>
#include <string>
#include <algorithm>
#include <iostream>
#include <iterator>

int main()
{
    // Initialise sample sets
    int vals1[] = { 4, 2, 1 };
    int vals2[] = { 1, 7, 5, 2 };
    int size1 = sizeof( vals1 ) / sizeof ( vals1[ 0 ] );
    int size2 = sizeof( vals2 ) / sizeof ( vals2[ 0 ] );

    std::set<int> s1( vals1, vals1 + size1 );
    std::set<int> s2( vals2, vals2 + size2 );
    // Find the intersection, providing it with an output iterator
    std::set<int> intersect;
    set_intersection( s1.begin(),
        s1.end(),
        s2.begin(),
        s2.end(),
        std::inserter(intersect,intersect.begin()) );
    std::cout << "intersection: ";
    std::copy( intersect.begin(),
        intersect.end(),
        std::ostream_iterator<int>( std::cout, " " ) );
```



```
return 0;
```

```
}
```

The output is as shown:



But what if you care about the position of inserted items? In this case **std::set** would not be appropriate. And in fact there is no STL container that does this at the moment, although **std::set** comes the closest because one of the types used to parameterize the **std::set** template is a comparison type. This comparison type determines the order in which new items are inserted. When this is not specified, it defaults to a less-than comparison. So to maintain a set of unique integer items preserved in the order in which they were inserted such as { 101, 111, 110, 001, 101 }, **std::set** (while ensuring the uniqueness of the numbers inserted) would not maintain this particular ordering. In cases like this, the solution will likely to involve more complex approaches such as creating your own class to combine container types in your own class. You might want **std::set** for storing unique items, and **std::vector** for keeping track of the order in which they were inserted:

```
#include <set>
```

```
#include <vector>
```

```
#include <string>
```

```
#include <algorithm>
```

```
#include <iostream>
```

```
typedef std::set<std::string>::iterator Iter;
```

```
class OrderedSet
```

```
{
```

```
private:
```

```
std::set<std::string> setItems;
```

```
std::vector<Iter> setOrder;
```

public:

OrderedSet() {}

void insert(const std::string& item)

{

setItems.insert(item);

Iter it = setItems.find(item);

setOrder.push_back(it);

}

int order(const std::string& item)

{

Iter it = setItems.find(item);

std::vector<Iter>::iterator order_it =

std::find(setOrder.begin(), setOrder.end(), it);

return order_it - setOrder.begin();

}

};

int main()

{

OrderedSet orderedSet;

orderedSet.insert("Bravo");

orderedSet.insert("Alpha");

orderedSet.insert("Charlie");

orderedSet.insert("Delta");

std::cout << "Order = "

<< orderedSet.order("Alpha")

<< std::endl;

```
return 0;
```

```
}
```

Moving on now, the **std::map** class allows us to store and obtain data items that are mapped to a key. This is useful when you need a means of quickly looking up a data item associated with a key value. The **std::map** container, while allowing for efficient insertion and retrieval as with **std::set**, differs from the **std::set** in that **std::map** stores pairs, rather than single values: **std::map<Key, T>**, where **Key** is used to reference the data object **T**.

A **std::set<Key>** is an ordered collection of keys, while a **std::map<Key, T>** is a collection of data objects that can be referenced by a key. A common way to use the **std::map<Key, T>** is as an associative-like array, whereby the user can look up a given data element using a specific key. Superficially this looks a lot like using a conventional array, but the important difference is that the index used to retrieve the array element no longer has to be an integer-like variable, but instead can be ANY type.

The secret is that **std::map<Key,T>** defines the **operator[]** function to perform the lookup to find the data element associated with the given key. If a match is found, the function returns the reference to its mapped value; otherwise a new data item is inserted with that key and the function returns the reference to its newly mapped value. To put it more simply, **std::map** is an “add or update” function (hat-tip: Scott Meyers). It’s worth pointing out a subtle efficiency issue here: if you’re adding brand new items, then the insert method is preferable to using **operator[]** since it will only create new items without replacing existing ones. But if you need to update an existing item in your **std::map**, then the **operator[]** would be the preferred option.

```
#include <map>
```

```
#include <string>
```

```
#include <algorithm>
```

```
#include <iostream>
```

```
int main()
```

```
{
```

```
typedef std::map<std::string, int> ResultsMap;
```

```
ResultsMap rmap;
```

```
rmap.insert( ResultsMap::value_type( “Physics”, 88 ) );
```

```
rmap.insert( ResultsMap::value_type( “Math”, 67 ) );
```

```

rmap.insert( ResultsMap::value_type( "English", 67 ) );

// operator[] usage ("add or update")
rmap[ "Math" ] = 90;
std::for_each( rmap.begin(), rmap.end(),
[] (std::pair<const std::string, int>& map)
{
    std::cout << map.first << " " << map.second << std::endl;
});

return 0;
}

```

Now let's go one step farther. The **std::multimap** container offers a way of mapping groups of data items to a key. This is in contrast to the **std::map** being a one-to-one mapping between keys and their associated data items. Rather than using the **find** method to look up a single item associated with its key, the **equal_range** method is more likely to be used to obtain the range of items associated with the group and iterate through these:

```

#include <map>
#include <string>
#include <algorithm>
#include <iostream>

int main()
{
    typedef std::multimap<std::string, int> ResultsMap;
    typedef std::multimap<std::string, int>::const_iterator ResultsIter;
    ResultsMap rmap;

    rmap.insert( ResultsMap::value_type( "Geography", 87 ) );
    rmap.insert( ResultsMap::value_type( "Physics", 57 ) );
    rmap.insert( ResultsMap::value_type( "Physics", 69 ) );

```

```
rmap.insert( ResultsMap::value_type( "Physics", 45 ) );
```

```
const std::string subject = "Physics";  
const std::pair<ResultsIter, ResultsIter> group =  
rmap.equal_range( subject );  
std::cout << subject << " =>";  
std::for_each( group.first, group.second,  
[](const std::pair<const std::string, int>& gr)  
{  
std::cout << gr.second << " "  
});  
  
return 0;  
}
```

10.3 Common STL Algorithms and their Usage

Replacing the traditional for loop

Besides improved readability, there exist some nice advantages to using **std::for_each** in place of the traditional for loop for iterating over sequences. It will allow you to write an algorithm on top of the **std::for_each** that will work with any iterator and reduces the risk of typo-introduced bugs. Specifically it applies a function object (functor) to each element within a range and returns a copy of the function object after it has been applied to all of the elements in the range. Here is an example of using **std::for_each** to append a set of strings contained in a **std::vector** container. This collects each 'txt' item stored in the in the **std::vector** of 'Token' structs, appending them to the 'txt' member of the Appender functor, which has the Word member function to return the complete concatenated string:

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
#include <algorithm>
```

```
struct Token
```

```
{
```

```
    std::string txt;
```

```
    Token( std::string s ) : txt( s ) {}
```

```
};
```

```
class Append
```

```
{
```

```
private:
```

```
    std::string txt;
```

```
public:
```

```
    Append() : txt("") {}
```

```
void operator() (const Token &x)
```

```
{ txt.append( x.txt + " " ); }
```

```

std::string Word() const
{ return txt; }
};

int main()
{
std::vector<Token> tokens;
tokens.push_back( Token( "Hello" ) );
tokens.push_back( Token( "and" ) );
tokens.push_back( Token( "welcome!" ) );

std::cout << std::for_each(
tokens.begin(),
tokens.end(),
Append() ).Word()
<< std::endl;

return 0;
}

```

Sorting and searching

The STL sort algorithm operates on a selected range of container elements, arranging them into a sorted order according to a user-defined predicate function object that defines how elements are compared. Here is an example of how to read a list of objects, sort them and output them to the screen:

```

#include <iostream>
#include <ostream>
#include <vector>
#include <algorithm>
#include <memory>

```

```

// A base class to hold, return and output a value

```

```
class Item
{
int val;

public:
Item( const int& val ) : val( val ) {}
virtual char* ItemName() = 0;
int GetVal() { return val; }
void OutputToStream( std::ostream& os )
{
os << ItemName() << "\t" << val << std::endl;
}
};
```

// Derived class - "old"

```
class OldItem : public Item
{
public:
OldItem( const int& val ) : Item( val ) { }
char* ItemName() { return (char*) "Old: "; }
};
```

// Derived class - "new"

```
classNewItem : public Item
{
public:
NewItem( const int& val ) : Item( val ) {}

char* ItemName()
{
return (char*) "New: ";
}
```



```

};

// Auto pointer to de-allocate Item objects in case you are forgetful.
typedef std::unique_ptr<Item> ItemPtr;

// Specify how we want to sort the data:
struct Ascending
{
    bool operator() ( const ItemPtr& start, const ItemPtr& end )
    {
        return start->GetVal() < end->GetVal();
    }
};

int main()
{
    std::vector< ItemPtr > itemVector;

    itemVector.push_back( static_cast<ItemPtr>( new OldItem( 43 ) ) );
    itemVector.push_back( static_cast<ItemPtr>( newNewItem( 34 ) ) );
    itemVector.push_back( static_cast<ItemPtr>( newNewItem( 443 ) ) );
    itemVector.push_back( static_cast<ItemPtr>( new OldItem( 433 ) ) );
    itemVector.push_back( static_cast<ItemPtr>( new OldItem( 343 ) ) );

    // Sort values in ascending order...
    std::sort(
        itemVector.begin(),
        itemVector.end(),
        Ascending() );

    std::for_each( itemVector.begin(), itemVector.end(), []( ItemPtr& item )
    {

```

```

item->OutputStream( std::cout );
});

return 0;
}

```

This gives the following output:



In fact, STL has an ample fund of algorithms to help you locate items stored in containers. These can be speedy, typically of order $\log N$ for **binary_search** or **equal_range**. And containers such as **std::map** are designed to keep the data sorted in a self-balancing binary search tree as they are inserted, which allows certain performance guarantees when it comes to looking these items up.

Copying, Filling, Replacing and Removing

Many tasks boil down to being able to copy objects around or remove them. Input and output iterators are often a necessary means to accomplish this since we need to specify what data we want to work with and then where to output the result.

This example copies the elements of a **std::vector** container to a standard output where **ostream_iterator** is an adaptor of an output iterator type. The iterator operations are defined such that assigning through the iterator prints to standard output, with each print followed by a whitespace character:

```

#include <vector>
#include <algorithm>
#include <iterator>
#include <iostream>

```

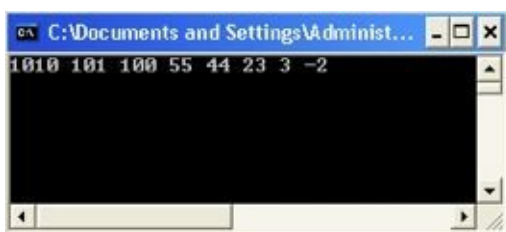
```

class descending
{
public:
bool operator()( int x, int y ) const
{ return x > y; }
};

int main()
{
int values[] = { 100, 101, -2, 23, 1010, 44, 55, 3 };
const size_t size = sizeof( values ) / sizeof( values[ 0 ] );
std::vector<int> v( values, values + size );
std::sort( v.begin(), v.end(), descending() );
std::copy( v.begin(),
v.end(),
std::ostream_iterator<int>( std::cout, " " ) );
return 0;
}

```

This gives the following output:



Mutating and Transforming

The **std::transform** algorithm applies a function to each object in a given range and copies the result of the function to a new range which is pointed to by an output iterator. This example shows how to use a function adaptor that takes two input ranges and increases the values of the first range by the second range.

```
#include <iostream>
```

```

#include <vector>
#include <algorithm>
#include <iterator>

template<typename T, typename InputIterator>
void Print(std::ostream& ostr,
InputIterator itbegin,
InputIterator itend,
const std::string& delimiter)
{
std::copy(itbegin, itend,
std::ostream_iterator<T>(ostr, delimiter.c_str()));
}

template<class T>
class IncreaseByVal
{
public:
T operator()( T a, T b ) const
{ return a + b; }
};

int main()
{
int x[] = { 1, 2, 3, 4, 5 };
int y[] = { 1, 2, 3, 4, 5 };

std::transform( x,
x + sizeof( x ) / sizeof( x[ 0 ] ),
y,
x,
IncreaseByVal<int>() );

Print<int, int[]>( std::cout,

```

```
x,  
x + sizeof( x ) / sizeof( x[ 0 ] ),  
” ” );  
return 0;  
}
```

10.4 More STL Tips, Tricks and Examples

Set Algorithms

Here is a practical example: using union and set intersection to calculate the Jaccard Index, a means of measuring the similarity and/or diversity of sample sets.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <iterator>
#include <string>

class Token
{
private:
    std::string str;

public:
    Token() {}
    Token( std::string val ) : str( val ) {}
    std::string Word() const
    { return str; }
};

struct Ascending
{
    bool operator() ( const Token& start, const Token& end )
    {
        return start.Word() < end.Word();
    }
};
```

```
int main()
{
    std::string str1[] = { "The", "crazy", "cat", "sat", "on", "the", "mat" };
    std::string str2[] = { "The", "cat", "sat", "on", "the", "red", "mat" };

    std::vector<Token>::iterator tok_intersect, tok_union;

    int size1 = sizeof( str1 ) / sizeof( str1[ 0 ] );
    int size2 = sizeof( str2 ) / sizeof( str2[ 0 ] );

    std::vector<Token> tokens1( str1, str1 + size1 );
    std::vector<Token> tokens2( str2, str2 + size2 );
    std::vector<Token> tokens( size1 + size2 );

    std::sort( tokens1.begin(), tokens1.end(), Ascending() );
    std::sort( tokens2.begin(), tokens2.end(), Ascending() );

    tok_intersect = std::set_intersection(
        tokens1.begin(),
        tokens1.end(),
        tokens2.begin(),
        tokens2.end(),
        tokens.begin(),
        Ascending() );

    int intersect_size = int( tok_intersect - tokens.begin() );

    tok_union = std::set_union (
        tokens1.begin(),
        tokens1.end(),
        tokens2.begin(),
```

```
tokens2.end(),  
tokens.begin(),  
Ascending() );
```

```
int union_size = int( tok_union - tokens.begin() );
```

```
double JaccardIndex = (double) intersect_size / (double) union_size;
```

```
std::cout << "Jaccard Index = "  
<< JaccardIndex  
<< std::endl;
```

```
return 0;  
}
```

Tokenizing and splitting strings

Tokenization is the process of breaking up text into separate items ('tokens') according to some delimiter, typically a whitespace or other punctuation character.

This example takes the sample text and inserts each tokenized word into a vector array:

```
#include <iostream>  
#include <string>  
#include <sstream>  
#include <algorithm>  
#include <iterator>  
#include <vector>
```

```
template<typename T, typename InputIterator>  
void Print( std::ostream& ostr,  
InputIterator itbegin,  
InputIterator itend,
```



```

const std::string& delimiter )
{
std::copy( itbegin,
itend,
std::ostream_iterator<T>( ostr, delimiter.c_str() ) );
}

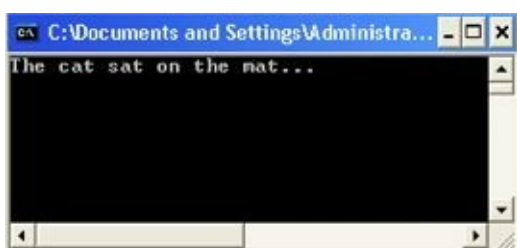
int main()
{
typedef std::vector<std::string>::iterator str_iter;

std::string txt = "The cat sat on the mat...";
std::istringstream iss( txt );
std::vector<std::string> tokens;
std::copy(
std::istream_iterator<std::string>(iss),
std::istream_iterator<std::string>(),
std::back_inserter<std::vector<std::string>>( tokens ) );

// Print the vector std::string items
Print<std::string, str_iter>( std::cout,
tokens.begin(),
tokens.end(),
" " );
}

```

This gives the tokenized output as shown:



Using containers as multi-dimensional arrays

Now here is an example of how to initialize and set values from a three-dimensional array implemented using the **std::vector** container. The STL container approach has the added benefit that unlike pointers to objects, the memory used in creating the array will be automatically removed when going out of scope. Just use standard array notation to retrieve array values. For example:

```
#include <vector>
#include <stdlib.h>

const size_t HEIGHT = 5;
const size_t WIDTH = 3;
const size_t DEPTH = 7;

int main()
{
    std::vector<std::vector<std::vector<double>>> array3D;

    // Allocate array dimensions by doing the resizing: HEIGHT x WIDTH
    array3D.resize( HEIGHT );
    for ( int i = 0; i < static_cast<int>( HEIGHT ); ++i )
    {
        array3D[ i ].resize( WIDTH );

        for ( int j = 0; j < static_cast<int>( WIDTH ); ++j )
        {
            array3D[ i ][ j ].resize( DEPTH );
        }
    }

    // Initialize with some values
    array3D[ 1 ][ 2 ][ 5 ] = 3.2;
```

```
array3D[ 2 ][ 0 ][ 3 ] = 4.1;
```

```
return 0;
```

```
}
```

How to sort a `std::map`

By definition you cannot sort a `std::map` container by value, since a `std::map` sorts its elements by key. However we can get around this by dumping `std::map` key-value pairs into a `std::vector` first, then sorting that `std::vector` with a less-than functor afterwards. This is illustrated by the following example which creates mappings between `std::pair` and double pairs. The STL map to create a one-to-one mapping between a pair of link node IDs (using a `std::pair`) and the weight value connecting these nodes (using a double):

```
std::map<std::pair<int,int>, double> link_map;
```

The first step is to physically create the mappings:

```
link_map[ std::pair<int,int>(0,1) ] = 1.1;
```

```
link_map[ std::pair<int,int>(1,2) ] = 0.1;
```

```
link_map[ std::pair<int,int>(2,3) ] = 6.2;
```

```
link_map[ std::pair<int,int>(3,4) ] = 3.4;
```

```
link_map[ std::pair<int,int>(4,5) ] = 5.7;
```

```
link_map[ std::pair<int,int>(5,6) ] = 2.2;
```

```
link_map[ std::pair<int,int>(0,8) ] = 1.8;
```

Then the `std::pair` node IDs and their respective weights are printed as follows. Notice that since the `std::pair` is used as the map key, the `std::map` items are automatically sorted according to this, not the weight values:

```
0 1 1.1
```

```
0 8 1.8
```

```
1 2 0.1
```

2 3 6.2

3 4 3.4

4 5 5.7

5 6 2.2

Now you need some means of obtaining the mappings so they are sorted according to their weights, not their **std::pair** key values. Since you can't sort the **std::map** itself, instead use a function to insert these mappings into a **std::vector** and then sort the **std::vector** items. This involves an additional functor which defines how the items are sorted:

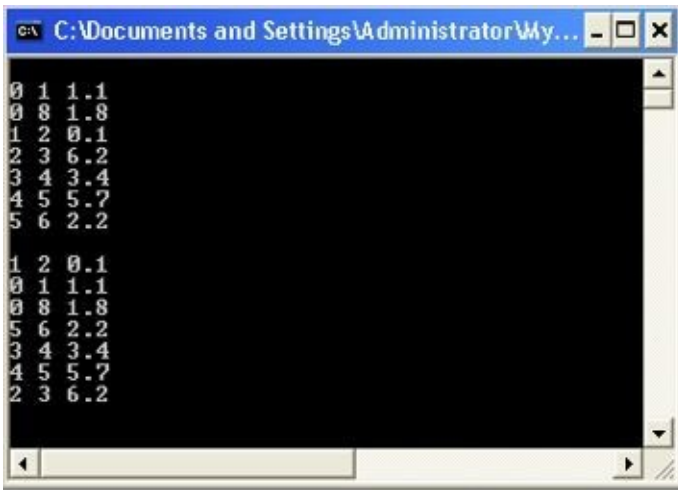
```
template<class T>
struct less_second : std::binary_function<T,T,bool>
{
    inline bool operator()( const T& lhs, const T& rhs )
    { return lhs.second < rhs.second; }
};

// Initialize and sort vector with data_t items from link_map
std::vector<data_t> sorted_edges()
{
    std::vector< data_t > vec(link_map.begin(), link_map.end());
    std::sort(vec.begin(), vec.end(), less_second<data_t>());
    return vec;
}
```

Also required is **data_t**, an additional typedef used to define the mapping:

```
typedef std::pair<std::pair<int,int>, double> data_t;
```

So that when we print out the vector of **data_t** items, we can see that they are sorted according to the second map parameter (**double**), and not the **std::pair** representing node IDs:



```
0 1 1.1
0 8 1.8
1 2 0.1
2 3 6.2
3 4 3.4
4 5 5.7
5 6 2.2

1 2 0.1
0 1 1.1
0 8 1.8
5 6 2.2
3 4 3.4
4 5 5.7
2 3 6.2
```

Full code listing

```
#include <map>
#include <vector>
#include <iostream>
#include <algorithm>

typedef std::pair<std::pair<int,int>, double> data_t;
std::map<std::pair<int,int>, double> link_map;
std::map<std::pair<int,int>, double>::iterator link_it;
template<class T>
struct less_second : std::binary_function<T,T,bool>
{
    inline bool operator()( const T& lhs, const T& rhs )
    {
        return lhs.second < rhs.second;
    }
};

// Initialize and sort vector with data_t items from link_map
std::vector<data_t> sorted_edges()
{
    std::vector< data_t > vec(
        link_map.begin(),
        link_map.end() );
```

```

std::sort(
vec.begin(),
vec.end(),
less_second<data_t>());
return vec;
}

void PrintLinkMap()
{
std::cout << std::endl;

std::for_each( link_map.begin(), link_map.end(),
[] ( std::pair< const std::pair<int,int>, double >& lmap )
{
std::cout << lmap.first.first << " "
<< lmap.first.second << " "
<< lmap.second << std::endl;
});
}

void PrintSortedEdges( const std::vector<data_t>& vec )
{
std::cout << std::endl;

std::for_each( vec.begin(), vec.end(), [] ( const data_t& dt )
{
std::cout << dt.first.first << " "
<< dt.first.second << " "
<< dt.second
<< std::endl;
});
}

int main()
{

```

```
// Create mapping between node pairs and weights
```

```
link_map[ std::pair<int,int>(0,1) ] = 1.1;
```

```
link_map[ std::pair<int,int>(1,2) ] = 0.1;
```

```
link_map[ std::pair<int,int>(2,3) ] = 6.2;
```

```
link_map[ std::pair<int,int>(3,4) ] = 3.4;
```

```
link_map[ std::pair<int,int>(4,5) ] = 5.7;
```

```
link_map[ std::pair<int,int>(5,6) ] = 2.2;
```

```
link_map[ std::pair<int,int>(0,8) ] = 1.8;
```

```
PrintLinkMap();
```

```
// Sort the mapping according to link weight
```

```
std::vector<data_t> vec = sorted_edges();
```

```
PrintSortedEdges( vec );
```

```
return 0;
```

```
}
```

Permanently removing STL container elements

Like all STL algorithms, `remove` receives a pair of iterators to identify the range of container elements over which it needs to operate, as shown in its declaration:

```
template< class ForwardIterator, class T >
```

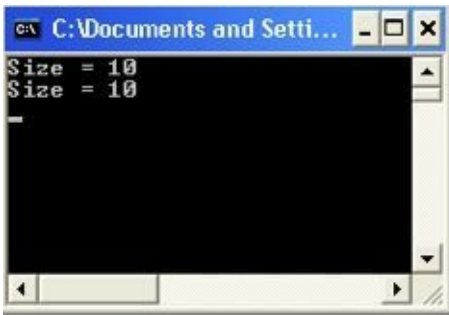
```
ForwardIterator remove( ForwardIterator first,
```

```
ForwardIterator last,
```

```
const T& value );
```

However, `remove` does not know about the container that holds the iterator elements, only the iterators. So `remove` is not able to go from the iterator to the container holding that iterator, and therefore is not physically able to remove elements from that container. That `remove` does not actually remove in the literal sense is a source of confusion for many! Try it and see how removing elements from a container does not actually change the total number of elements in that container:

This gives the following output:



Using STL to generate permutations

Put simply, given an input string how to we generate all of its possible permutations? A string permutation is found by reordering the position of its elements. We know that a string of length n will have $n!$ (n factorial) permutations. For example, the string “xyz” will have the permutations:

xyz

xzy

yxz

yzx

zxy

zyx

You can see this with the following code:

```
#include <iostream>
```

```
#include <algorithm>
```

```
#include <iterator>
```

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

```
int main()
```

```
{
```

```
char str[] = “abcd”;
```

```
const size_t len = strlen( str );
```

```
do
```

```
{
```



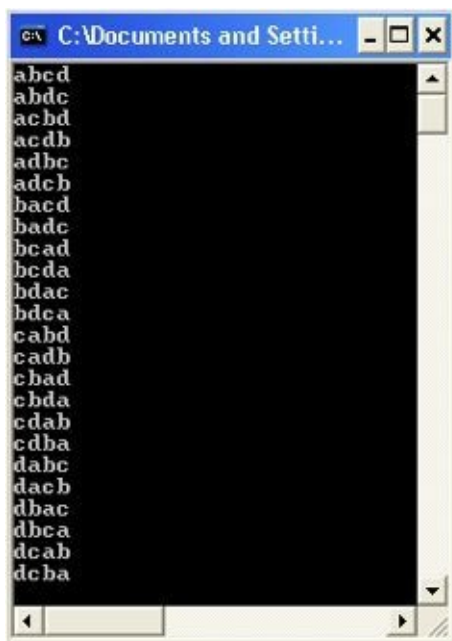
```

std::copy( str,
str + len,
std::ostream_iterator<char>( std::cout ) );

std::cout << std::endl;
}
while ( std::next_permutation( str, str + len ) );
}

```

This gives the following output:



Counting the occurrences of words in a text file

Now here is an example of the power of using STL's associative arrays as a word counter. This code reads the contents of a text file while keeping a running total of the number of occurrences of each word. Discounting the code that outputs the results and strips out the punctuation characters we do not wish to count uses just a few lines of code. This deceptively simple line of code...

```

counter[ tok ]++;

```

... is used to look up the counter value for a given word and increment its value. It navigates the red-black tree used by the standard map to find the appropriate tree node,

selects the **T** portion of the pair **<Key,T>**, and then increments it.

Note that the word counter has a default constructor to set the counter value to zero. Without this, the counter is set to some arbitrary value contained in the memory when the **counter[tok]++** line of code is run for the first time. Not the zero value we want.

```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <algorithm>
#include <map>
#include <string>
#include <vector>
#include <iterator>

template <class T>
struct less_second : std::binary_function<T, T, bool>
{
    bool operator()( const T& lhs, const T& rhs )
    { return lhs.second.value < rhs.second.value; }
};

class WordCounter
{
public:
    int value;
    WordCounter() : value( 0 ) {}
    void operator++ (int) { value++; }
};

// Remove unwanted characters from a string
bool filter(char c)
{
    return isalpha( c ) == 0;
}
```

```

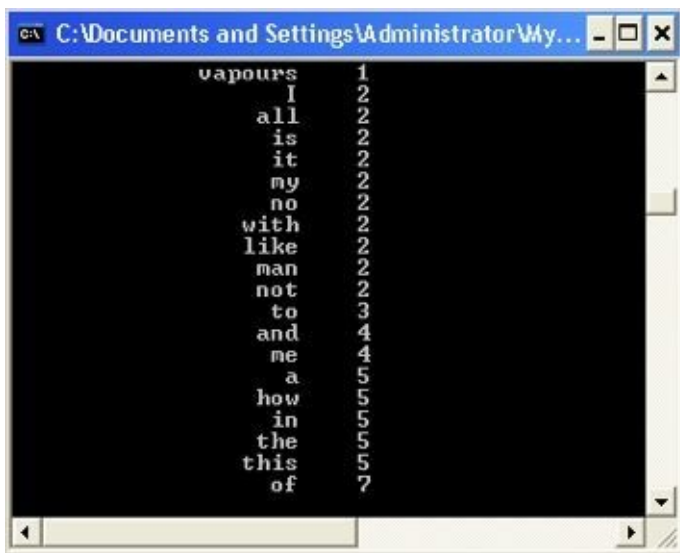
const std::string path = "Hamlet.txt";
int main()
{
typedef std::pair<std::string, WordCounter > word_mapping;
std::map<std::string, WordCounter> counter;
std::ifstream input;
input.open( path.c_str() );
if ( !input )
{
std::cout << "Error in opening file\n";
return 0;
}
std::string tok;
while ( true )
{
input >> tok;
// Remove ?, !, etc characters etc from string
tok.resize( std::remove_if( tok.begin(),
tok.end(),
filter ) - tok.begin() );

if ( tok == "" || tok == "\t" || tok == "\n" )
continue;
if ( input )
{
counter[ tok ]++;
}
else break;
}
std::map< std::string, WordCounter, std::less<std::string> >::iterator it;
// And then sort and display the result:
std::vector< word_mapping > result( counter.begin(), counter.end() );

```

```
std::sort( result.begin(), result.end(), less_second<word_mapping>() );
std::for_each( result.begin(), result.end(), []( const word_mapping& wmp )
{
std::cout << std::setw (20) << wmp.first << "\t"
<< wmp.second.value << std::endl;
});
return 0;
}
```

The console output is shown here:



```
C:\Documents and Settings\Administrator\My...
vapours      1
I            2
all          2
is           2
it           2
my           2
no           2
with         2
like         2
man          2
not          2
to           3
and          4
me           4
a            5
how          5
in           5
the          5
this         5
of           7
```

Conclusion

I generally find programming books dense and difficult to assimilate. Though the content may be excellent they are frequently let down through poor presentation and excessive wordiness. The likelihood of successfully working through a thousand page tome of dry exposition within a week or two is slim. The chances are you will burn out within a few days. I know because I have tried and failed with this approach a number of times.

This ebook attempts to present a difficult subject as a series of lessons in ways that are comprehensible to the intelligent layperson. A traditional ‘professional’ book could have been about five times as big, while still missing important information. One could go on about how difficult Windows programming is, or how incomprehensible the error messages in Boost appear, but by using plenty of examples and screenshots, this ebook shows you some quick ways of getting to grips with such things.

For those reasonably familiar with programming in C or C++ in the traditional scenario of creating console applications, the first two lessons will be effective in making the transition to writing Windows applications easier. You will also gain an understanding of how to create your own customized controls, build a dialog-based web browser and play back media files using DirectShow, amongst other things.

Lesson three is an accessible introduction to getting started with the Boost libraries in a variety of development environments. Examples of using Boost for six different libraries are given. Having this valuable skill will increase your productivity by providing tailor-made solutions to many common programming problems.

Also included are lessons to get you started with using graph algorithms, which are particularly useful in areas like telecommunications. Here we show you how to model networks, as well as applying techniques with which to ascertain connectivity, find shortest paths and determine minimal spanning trees.

There are times when as a developer you may be asked to code a complex algorithm. A number of the lessons given here show you how to prototype algorithms and de-mystify the process. Examples include the shunting-yard algorithm used for evaluating mathematical expressions, the “all simple paths” problem for finding routes in networks and a Sudoku puzzle-solver. In all these examples we verify that the computational results are as expected.

When you take advantage of the Standard Template Library (STL) you make your life simpler and more efficient. The final lesson on STL will dramatically improve your programming productivity and style, allowing your code to be far more reliable and reusable.

Learning C++ is no small task. I hope this ebook has proved to be a practical resource that you can actually read, and more importantly, put into practice.