**C Programming Portfolio Documentation**

**Contents:**

* Title page/Contents: page 1
* Portfolio Q1: page 2
* Portfolio Q1 code listing: page 3
* Portfolio Q1 screen shots: page 4
* Portfolio Q2: page 5
* Portfolio Q2 code listing: page 6/7
* Portfolio Q2 screen shots: page 8/9
* Portfolio Q3: page 10
* Portfolio Q3 code listing: page 11
* Portfolio Q3 screen shots: page 12
* Portfolio Q4: page 13/14
* Portfolio Q4 code listing: page 15
* Portfolio Q4 screen shots: page 18
* Portfolio Q5: Page 21
* Portfolio Q5 code listing: page 22
* Portfolio Q5 screen shots: page 29

**PORTFOLIO QUESTION 1 of 6 SIGN OFF WEEK 4. (10 marks)**

Note. You must use the algorithm as outlines below.

Write a program that repeatedly request an integer from the user and displays the integer as a binary number in 32 bits. It should terminate when the value 9999 is entered. A typical run might look like:

Enter integer: 16

00000000000000000000000000010000

Enter integer: -1

11111111111111111111111111111111

Enter integer: 9999

You MUST use one of the shift operators (<< or >>) and the bitwise AND operator & (not &&). The basic idea is to AND a mask with the number. To give you an idea of the algorithm, suppose integers are 8 bits long rather than 32 bits and the variable data contains the number and the variable mask the mask.

Number 01110010

Mask 10000000

AND'ed = 00000000

If the result of AND'ing data with mask is non-zero print a ‘1’ else print a ‘0’. Now shift the mask one bit right (>>) and repeat the AND.

number 01110010

mask 01000000

AND'ed = 01000000

Continue until the mask has been shifted eight bits right.

**Code Listing:**

//\*\*integer2binary.cpp

//\*\*Integer conversion to binary

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#define MAX\_LINE\_LEN 32

void displayBinaryB(unsigned, size\_t);

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

char line[MAX\_LINE\_LEN];

int b;

printf("Enter integer: ");

while (1) {

do {

printf("\n> ");

fgets(line, MAX\_LINE\_LEN, stdin);

} while (sscanf(line, "%d", &b) != 1);

if(b == 9999)

break;

displayBinaryB(b,MAX\_LINE\_LEN);

}

system ("PAUSE");

return 0;

}

void displayBinaryB(unsigned b, size\_t numBits) {

if (numBits == 0) return;

printf("%d", (b >> (numBits-1)) & 1);

displayBinaryB(b, numBits - 1);

}

**Screen shots:**

This program coverts an integer to binary and displays it in 32 bits.

This sets the maximum binary numbers to 32 and is called later in the program.

This displays a message when the program is run.

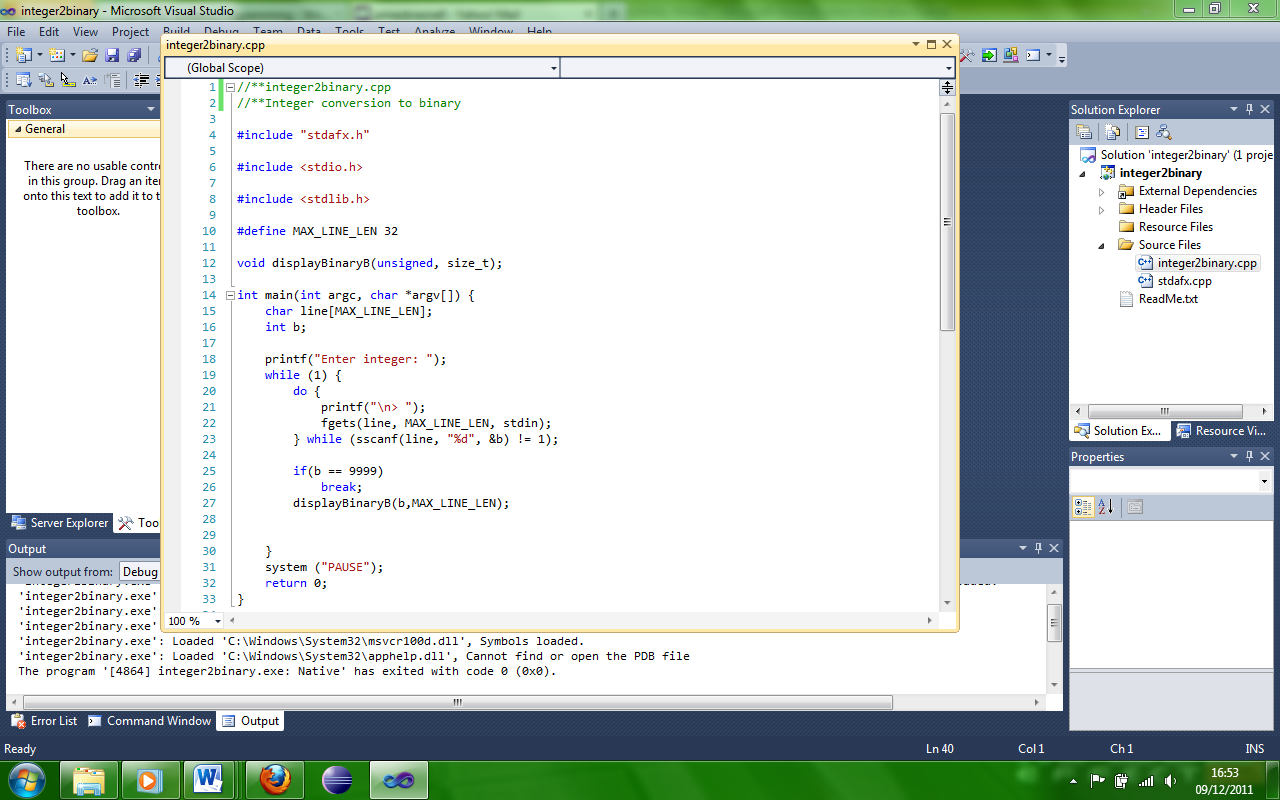
This scans the users input for the integer.

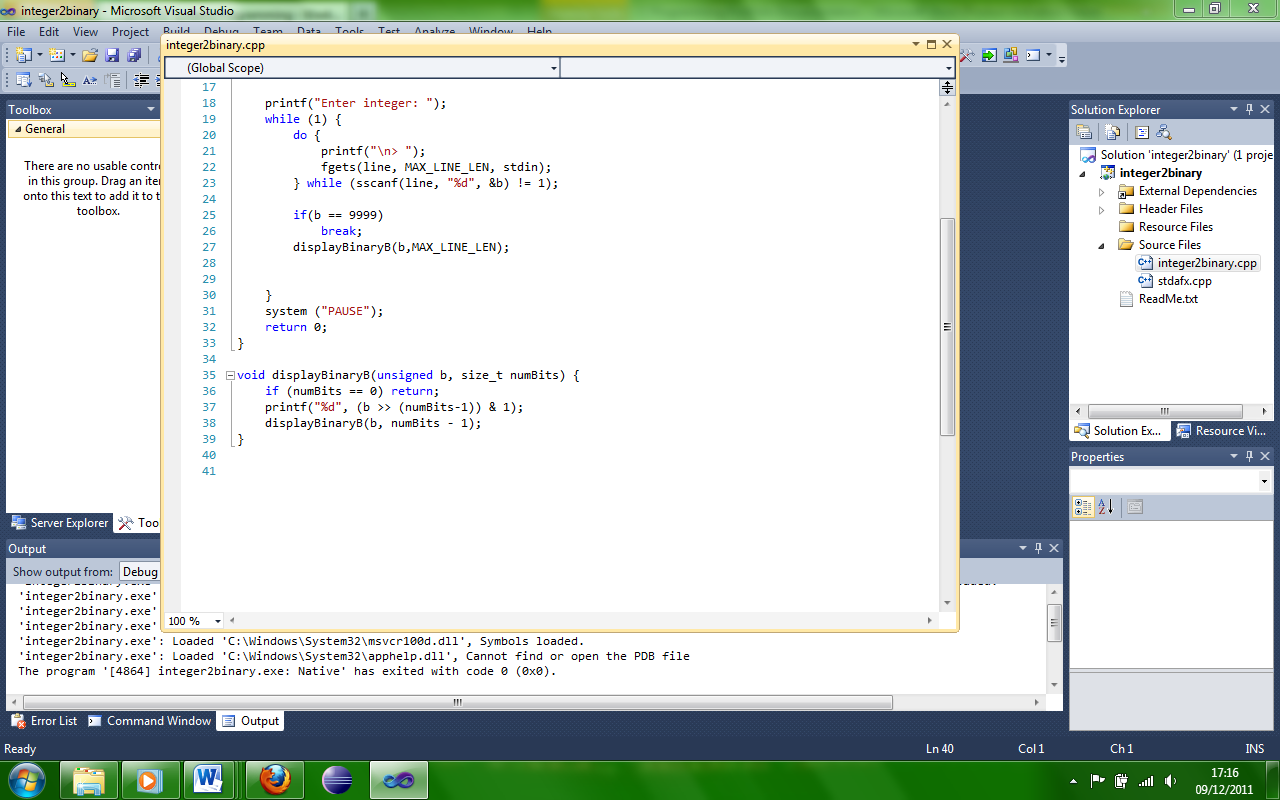
The if statement shuts down the program if the integer 9999 is input by the user.

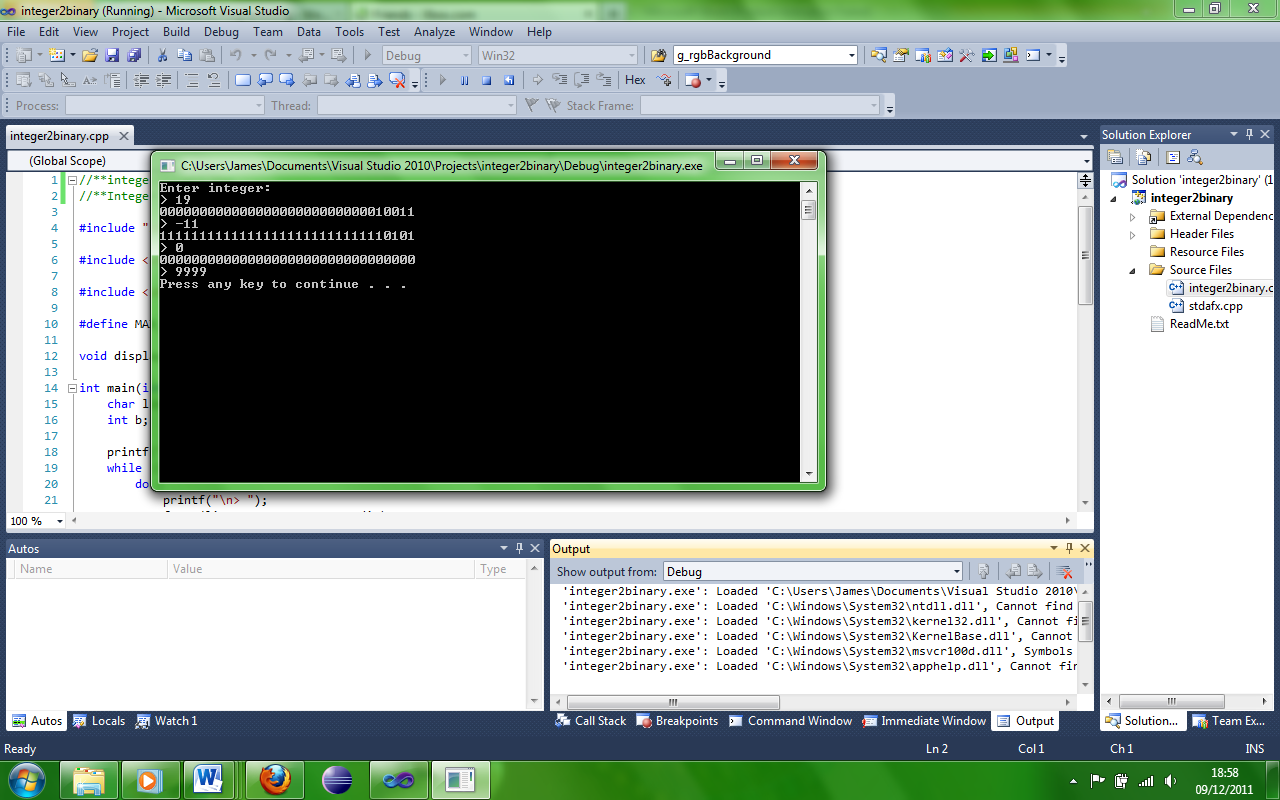
This pauses the program before it closes the window making it easier to see a programs output.

This part of the program handles the conversion of integer to binary and is called earlier in the program.

The program clearly converts and integer to binary as well as –integers, the program also ends when the integer 9999 is input by the user.







**PORTFOLIO QUESTION 2 of 6 SIGN OFF WEEK 5. (10 marks)**

Modify the workshop question from week 2 to accept the characters from a

data file rather than from the keyboard. Extend the program to convert all

the characters to upper case (Excluding numeric characters) and save the

characters to a new data file.

Use notepad to create the original file and to view the results of the output

data file.

***Extend the program*** *: hide the word ‘dog’ in the character data file and modify the application to detect the hidden word – detect the position of the word, - detect the number of occurrences of the word, - detect a number of different words ‘cat’, ‘rat’, etc.).*

***NOTE. ASSUME THAT ALL HIDDEN WORDS ARE 3 CHARACTERS IN***

***LENGTH.***

**Code Listing:**

// Week6.cpp : Defines the entry point for the console application.

//Author: James Braznell

//Created: 8/10/2011

//Versions: version 1, version2

//Version1 details: version 1 was workshop 2 which accepted characters and determined whether the character

//was an alphabetic uppercase character or alphabetic lower case character or a numeric. Displayed the character counts

//and produce a graph of asterisks to represent the counts.

//Version2 details: version 2 has all features present in version 1 but also accepts the characters from a data file

//and writes them to a second file in capitals if lower case.

#include "stdafx.h"

#include <stdio.h>

#include <stdlib.h>

#include<ctype.h>

void print(int c);

void main ()

{

int d,upper,digits,lower;

lower = upper = digits = 0;

char d\_char;

FILE \*inputFile = fopen("myfiles/myfile.txt","r");

FILE \*outputFile = fopen("myfiles/my\_output\_file1.txt", "w");

if (inputFile==NULL)

{

perror ("Error opening file");

}else{

do

{

d = fgetc (inputFile);

if(d == EOF)

{

break;

}

if (isupper(d))

upper++;

if (isdigit(d))

digits++;

if (islower(d))

{

lower++;

d = toupper(d);

}

d\_char = (char) d;

fputc(d, outputFile);

} while (d != EOF)

printf("\n Upper case (%d) :", upper);

print(upper);

printf ("\n");

printf("\n Lower case (%d) :", lower);

print(lower);

printf ("\n");

printf("\n Digit (%d) :", digits);

print(digits);

printf ("\n\n");

fclose(inputFile);

fclose(outputFile);

system ("PAUSE");

}

}

void print(int c)

{

int i;

for (i = 0; i < c; i++)

printf("\*");

}

**Screen shots:**

The program accepts characters from a data file and determines whether the character was upper case, lower case or a number. It displays the character counts and produces a graph of asterisks to represent counts. It also outputs the characters to a separate file with the lower case characters changed to upper case.

Here the program sets up numerous attributes including the opening of the input and output files.

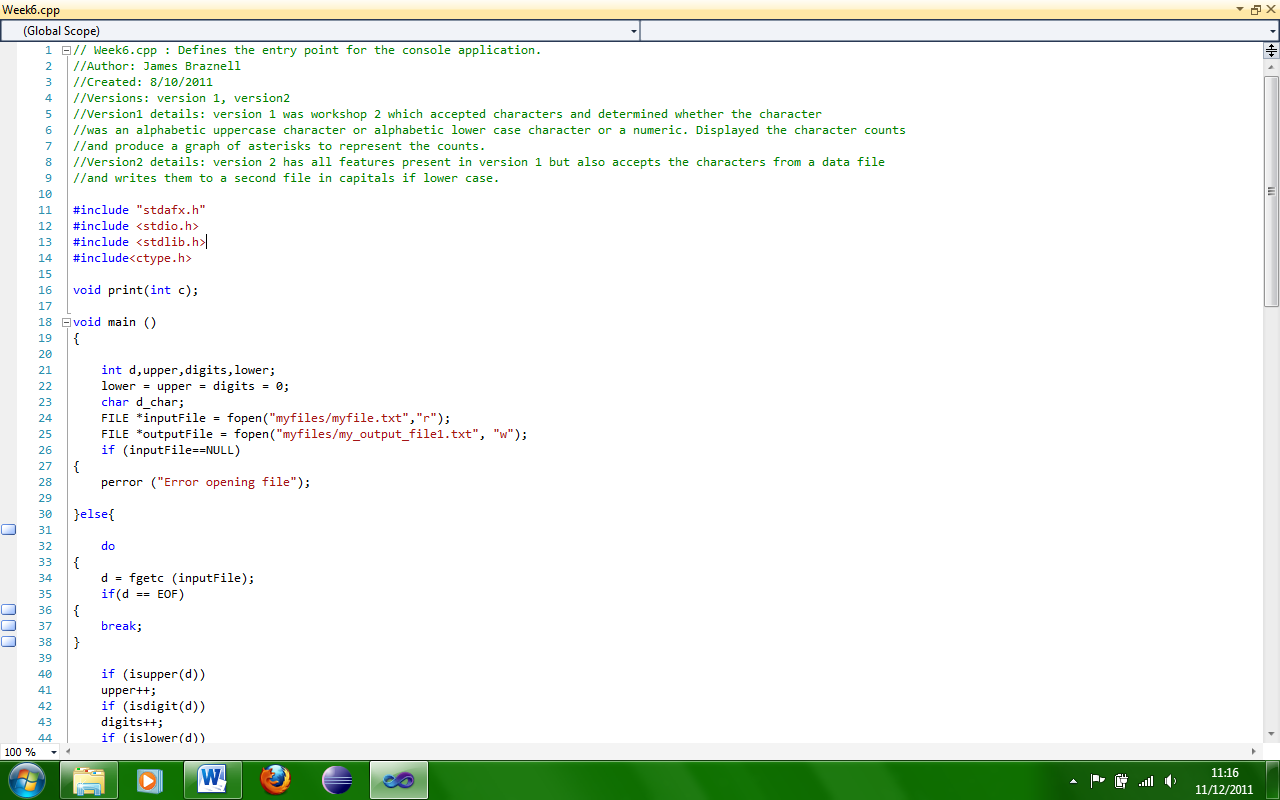
Here the program gets characters from the input file.

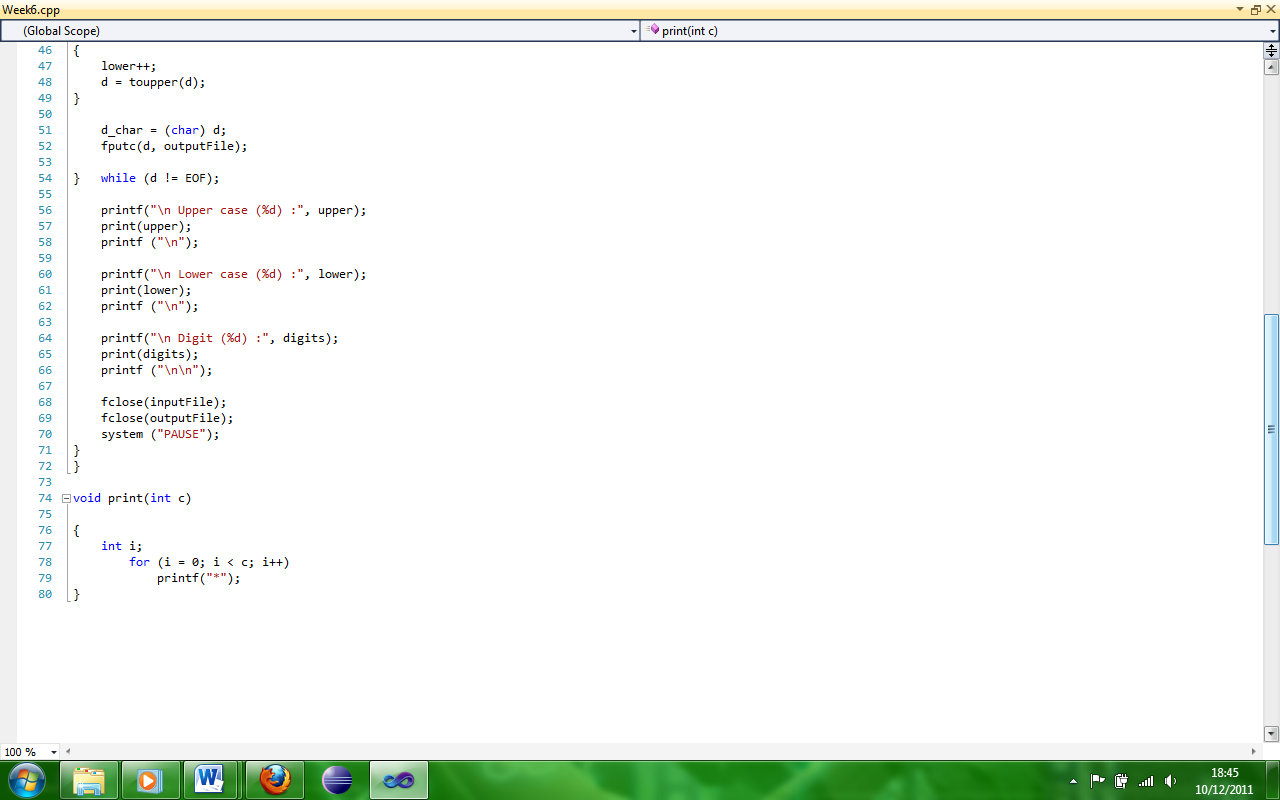
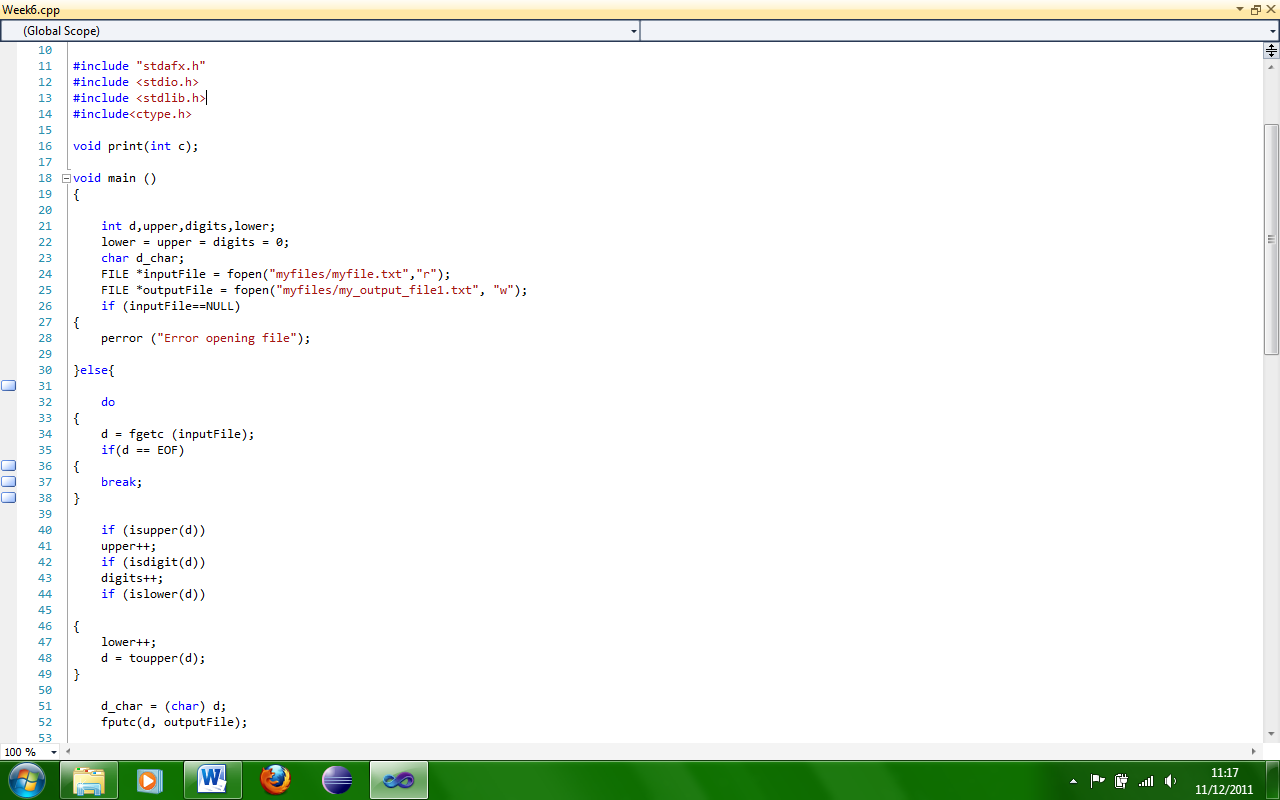
The program then checks these characters to see if they are upper case, lower case or numbers, then adds a count for each of them whenever one of the respective character types are found. Also when a lower case character is found it changes these to upper case when outputted.

Here the program prints out the names of the character types with the counts of the types as well as asterisks to more clearly show the counts.

Here the program closes the input and output files as well as pauses the program before it closes the window making it easier to see a programs output.

Here the asterisks are added for each count of character types and prints them out in the programs output.

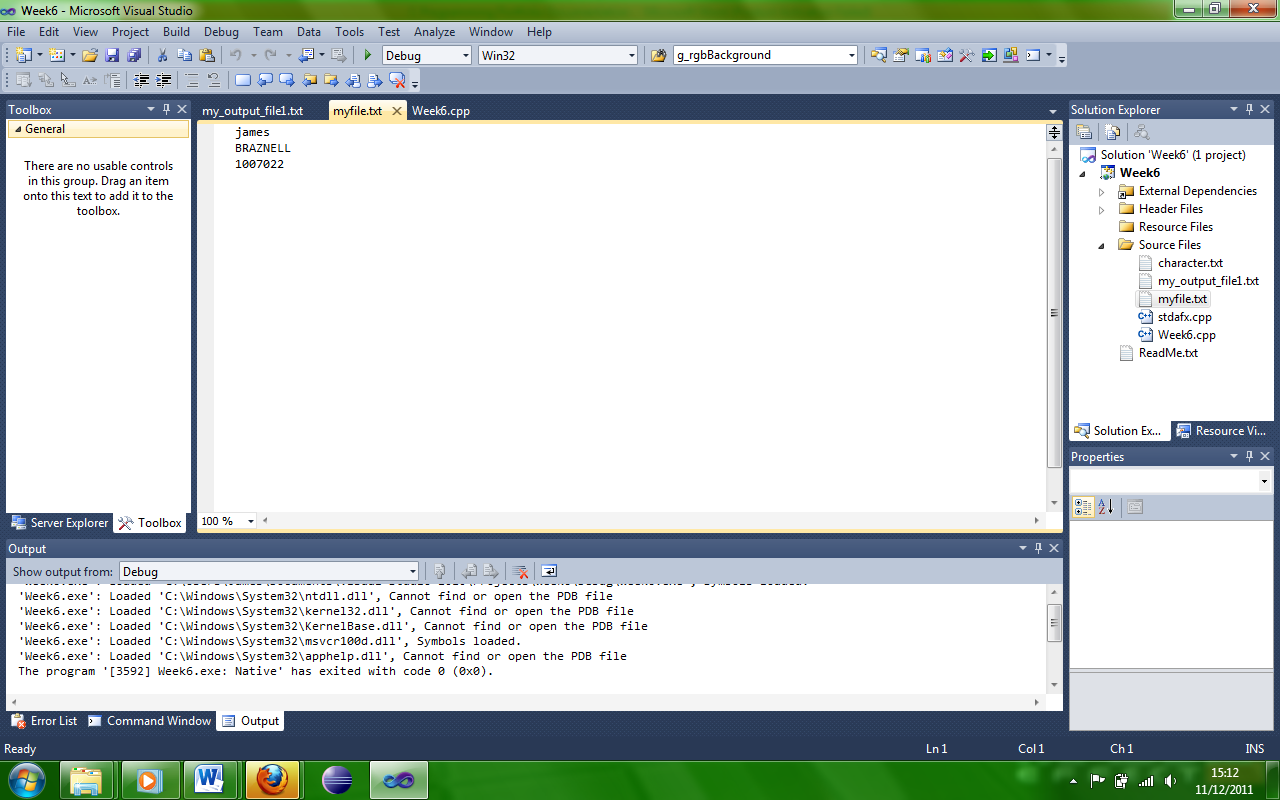


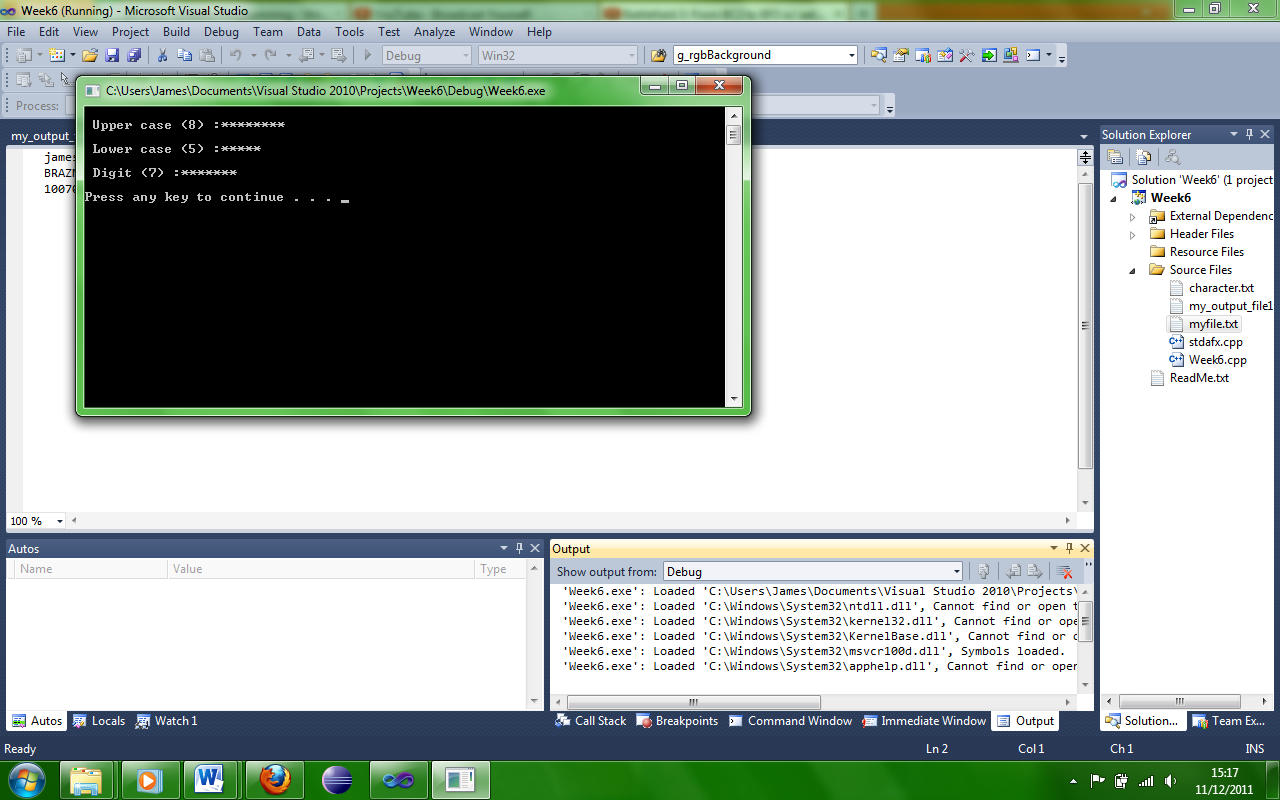


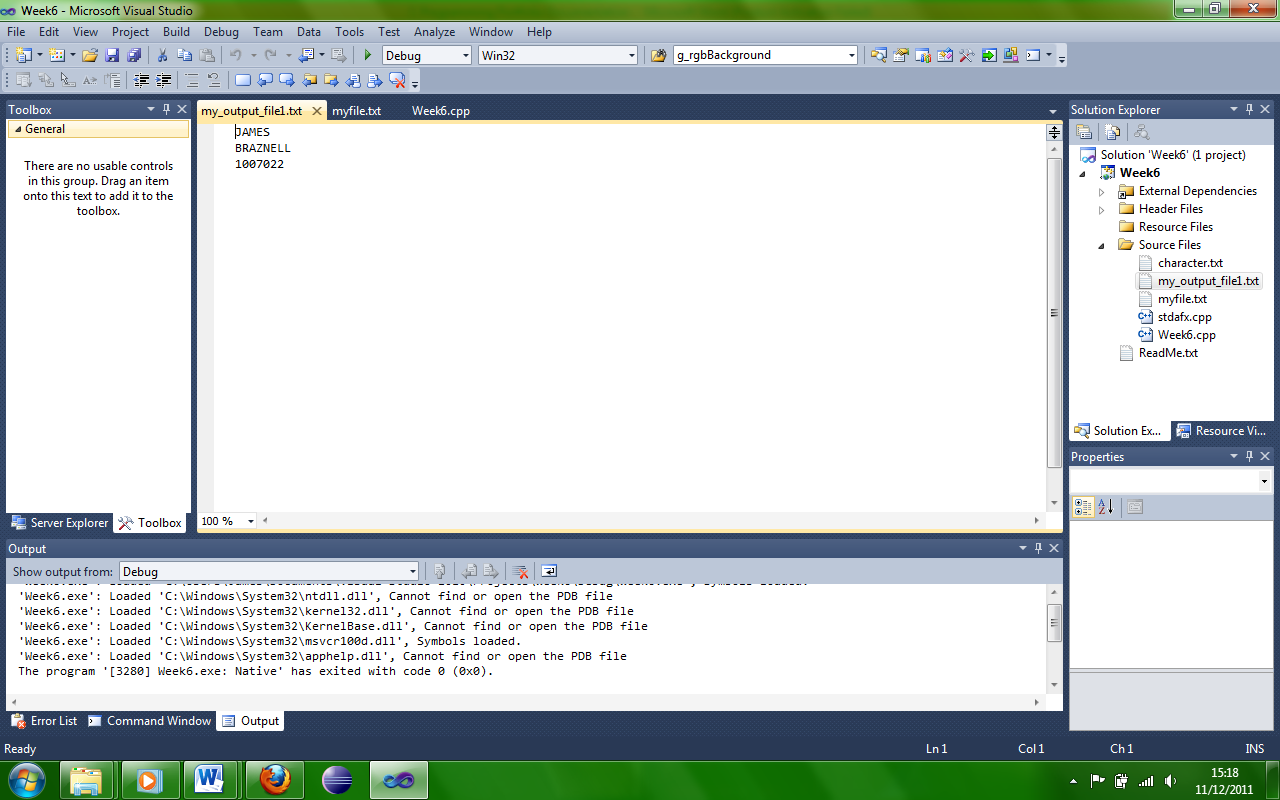
Here is the input file and clearly has lower and upper case characters as well as numbers.

Here is the programs output and shows the number of counts a character type in number as well as asterisks creating a crude “graph” to help show the counts.

Here is the output file and the program, has clearly outputted the characters from the input file and changed the lower case characters to upper case.







**PORTFOLIO QUESTION 3 of 6. SIGN OFF WEEK 6. 10 marks**

**This question is to be signed off at the start of your normal workshop class only. The program should be included in the portfolio due in during week 13. For each portfolio question you need to include, question, design (limited), code listing, screen shots to show testing, brief comments.**

*If you fail to demonstrate or have failed to meet the minimum pass grade you have the opportunity re-demonstrate your work again the following week* ***(Note this MUST be the following week).*** *If a pass grade is achieved a capped D grade will be awarded.*

(1) Write a function with the following prototype. **Note**. You MUST use the prototype below.

char \*mystrcat(char \*s, char \*p);

The function appends string p to the end of string s and returns a pointer to the start of string s.

Remember that strings are terminated by binary zero (‘\0’).

Suppose string s contains “abcd” and string p contains “xyz”. The result would be pointer to the string containing “abcdxyz”.

**Marking Criteria:**

1. Program works : Appends two strings use of the keyboard for input. (4 marks)

2. Check code : Used prototype as defined in the task. (3 marks)

char \*mystrcat(char \*s, char \*p);

3. User function includes use of pointers. (3 marks)

**Code Listing:**

// portfolio 3.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <stdio.h>

#include <string.h>

char \*mystrcat(char \*j,char \*p); // Prototype

int main(void)

{

char string1[150], string2[150];

char \*b;

while(1){

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nEnter string or EXIT to quit : ");

scanf("%s",&string1);

if(!strcmp(string1,"EXIT"))

break;

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nEnter string : ");

scanf("%s",&string2);

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nstrings are %s %s ",string1,string2);

printf("\nstrings after adding together %s ",mystrcat(string1,string2));

printf("\n\n");

}

}

char \*mystrcat(char \*j,char \*b)

{

return strcat(j,b);

}

**Screen shots:**

This program takes two strings from the users input and appends them together.

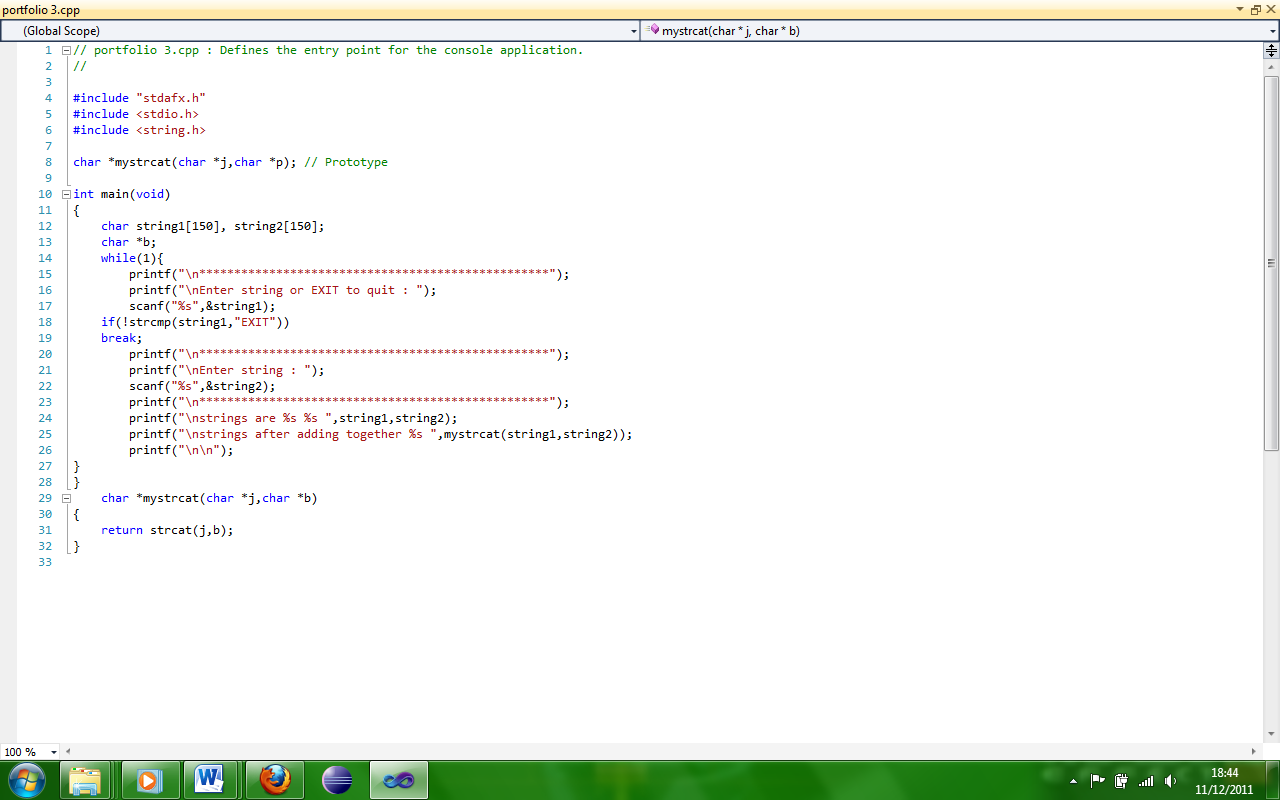
Here the prototype char \*mystrcat(char \*s, char \*p); is used to be called later.

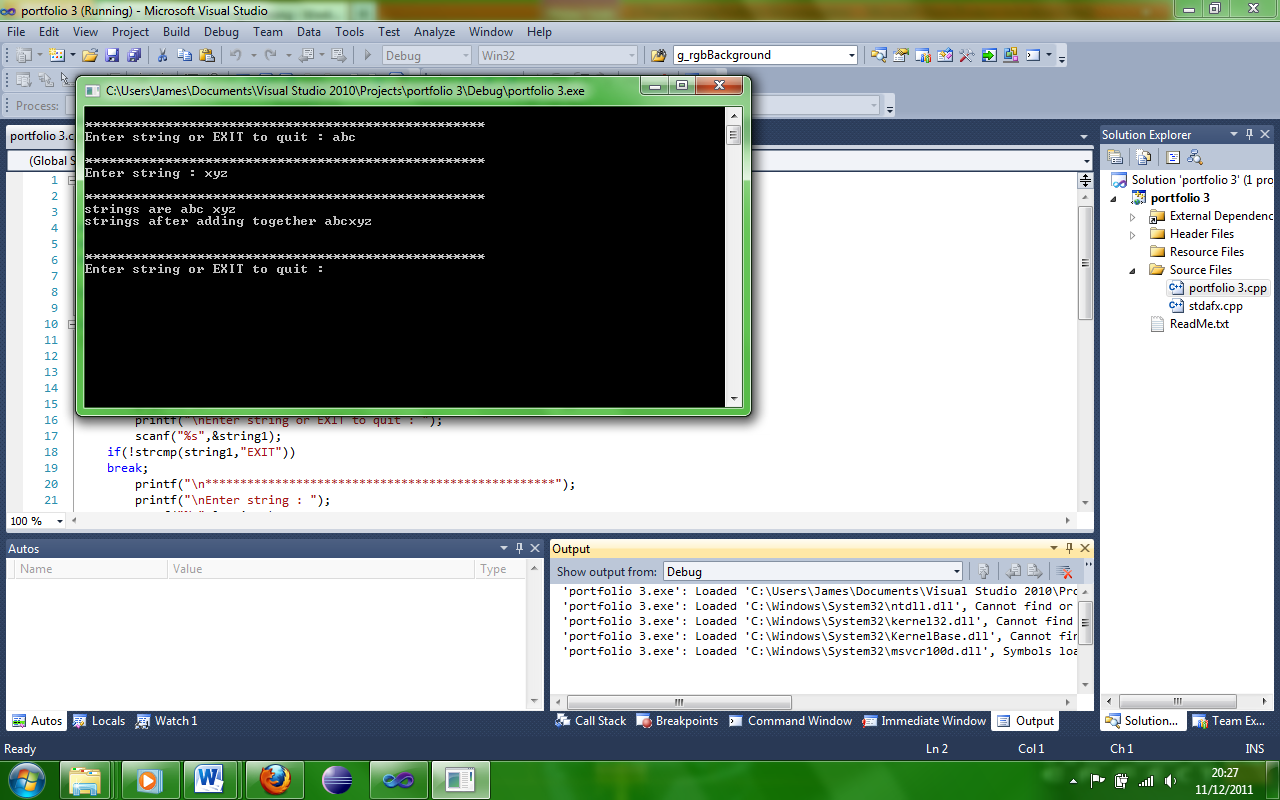
Here the program sets the two strings to a buffer size of 150.

The program prints out instructions for the user and scans the users input.

The program prints out the strings separately and then together.

When run the program asks for a string and then a second string, it displays these separate and then together.





**PORTFOLIO QUESTION 4 of 6. SIGN OFF WEEK 8. (30 marks)**

**This question is to be signed off at the start of your normal workshop class only. The program should be included in the portfolio due in during week 13. For each portfolio question you need to include, question, design (limited), code listing, screen shots to show testing, brief comments.**

*If you fail to demonstrate or have failed to meet the minimum pass grade you have the opportunity re-demonstrate your work again the following week* ***(Note this MUST be the following week).*** *If a pass grade is achieved a capped D grade will be awarded.*

You are required to write a program to convert Text into Morse code. The Letters A-Z can be represented in Morse code. Each Letter is represented by a combination of up to four dots “.” and/or dashes “-“.

For example

A = “.- “, B = “-...”,etc through to Z=”--..”

In each group of characters spaces have deliberately been introduced to fill the code up to the maximum of four characters. However, the space does not represent part of the code itself. Hence in this case each text letter is represented by four Morse code characters.

**The codes for the Letters A – Z *MUST* be held in a string array which *MUST* be DYNAMICALLY ALLOCATED to memory :**

You should use the string below for the characters a-Z

“.- -...-.-.-.. . ..-.--. ...... .----.- .-..-- -. --- .--.--.-.-. ... - ..- ...-.-- -..--.----..”

**Improved Solution**

Create a menu to read characters from the keyboard or a full message from a text file.

**Advanced work.**

Convert from Morse code to characters.

You may wish to consider including Error checking, error logs, etc.

**Marking Criteria:**

**Basic Solution:** (Total of 20 marks)

(consider these points)

Morse code string held in **Dynamic memory.**

**Message** from keyboard **or** held in string convert to Morse code.

**Function** to **return Morse code character(s) – pointers used.**

**Improved Solution:** (Total 25 marks)

(As above plus consider these points)

Accept message from a file. (*Test with file : Use “morse code – text format”)*

Menu option

**Advanced Solution:** (Total 30 marks)

(As above plus consider these points)

Converts Morse to a readable message.

*(Test with file : Use “morse code – morse format”)*

Modified menu option

Error checking – invalid characters should be detected

Error File.

**Code Listing:**

// portfolio4.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include <ctype.h>

#include <stdio.h>

#include <string.h>

#include <malloc.h>

void Myoptions(); // Prototype

void Keyboardinput();

void KeyboardMorse(char \*apple );

void MorseFile(char \* data);

void Fileinput();

int \_tmain(int argc, \_TCHAR\* argv[])

{

Myoptions(); // Function call

}

void Myoptions(){

// Creates the menu on screen when run

int optionsMenu;

do{

printf("\n");

printf("\n");

printf("\*\*\*\*\*\* Options Menu \*\*\*\*\*\*");

printf("\n");

printf("\n");

printf("1. Keyboard input");

printf("\n");

printf("\n");

printf("2. File input");

printf("\n");

printf("\n");

printf("0. Exit");

printf("\n");

printf("\n");

printf("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n");

printf("\n");

printf ("Enter your choice: ");

scanf("%d", &optionsMenu); // Asks for menu input from user

getchar();

switch (optionsMenu) {

case 0:

optionsMenu = 0;

break;

case 1:

Keyboardinput(); // Function call

break;

case 2:

Fileinput();

break;

default:

printf("Error, ivalide input, exiting \n");

break;

}}while(optionsMenu !=0);

}

void Keyboardinput(){

char apple[100];

printf("Enter text to be converted to morse code: ");

scanf("%s", &apple);

KeyboardMorse(apple);

printf("\n");

}

// Read from file and convert to Morse code

void Fileinput(){

char ch;

FILE \*pInputFile; // File handler

pInputFile = fopen("james.txt","r"); // Open file to read

if (pInputFile == NULL)

printf("Error loding file.");

else{

do{

ch = getc(pInputFile);

if(ch == EOF)

break;

ch = toupper(ch);

if((ch>=64) && (ch <=90))

MorseFile(&ch);

}while(ch != EOF);

}

fclose(pInputFile);

}

void KeyboardMorse(char \*apple ){

char \*ss= (char\*)malloc(110\* sizeof(char)); // Allocating dynamic memory

strcpy(ss, ".- -...-.-.-.. . ..-.--. ...... .----.- .-..-- -. --- .--.--.-.-. ... - ..- ...-.-- -..--.----..");

int i = 0,k=0;// Loop handler

int n = 0; // Handle morse code position

char ch = 65; // Handling character

char c;

while(ch!='\0'){

ch = apple[i];

if(ch == 32){

printf("%c",32);

continue;

}

i++;

ch = toupper(ch);

n = ch -65;

if(ch !='\0')

for(k =0; k<4;k++)

printf("%c",ss[n++]);

}

void free(void \*ss); // Free the allocated memory

}

// Take data from read file function and convert to morse code

void MorseFile(char \* data){

char \*ss= (char\*)malloc(110\* sizeof(char)); // Allocating dynamic memory

strcpy(ss, ".- -...-.-.-.. . ..-.--. ...... .----.- .-..-- -. --- .--.--.-.-. ... - ..- ...-.-- -..--.----..");

int k=0; // Loop handler

int n = 0; // handle morse code position

char ch = 65; // handling character

ch = \*data;

n = ch -65;

for(k =0; k<4;k++)

printf("%c",ss[n++]);

void free(void \*ss); // Free the allocated memory

}

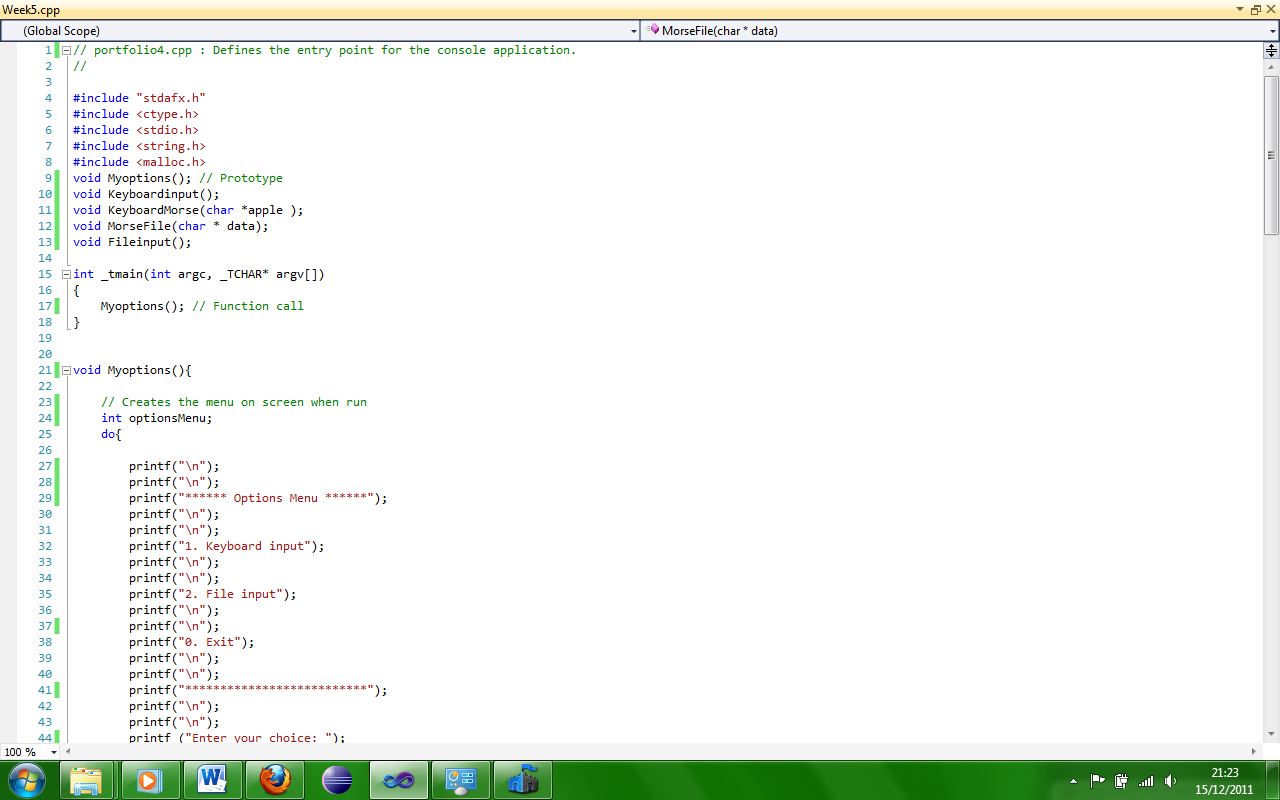
**Screen shots:**

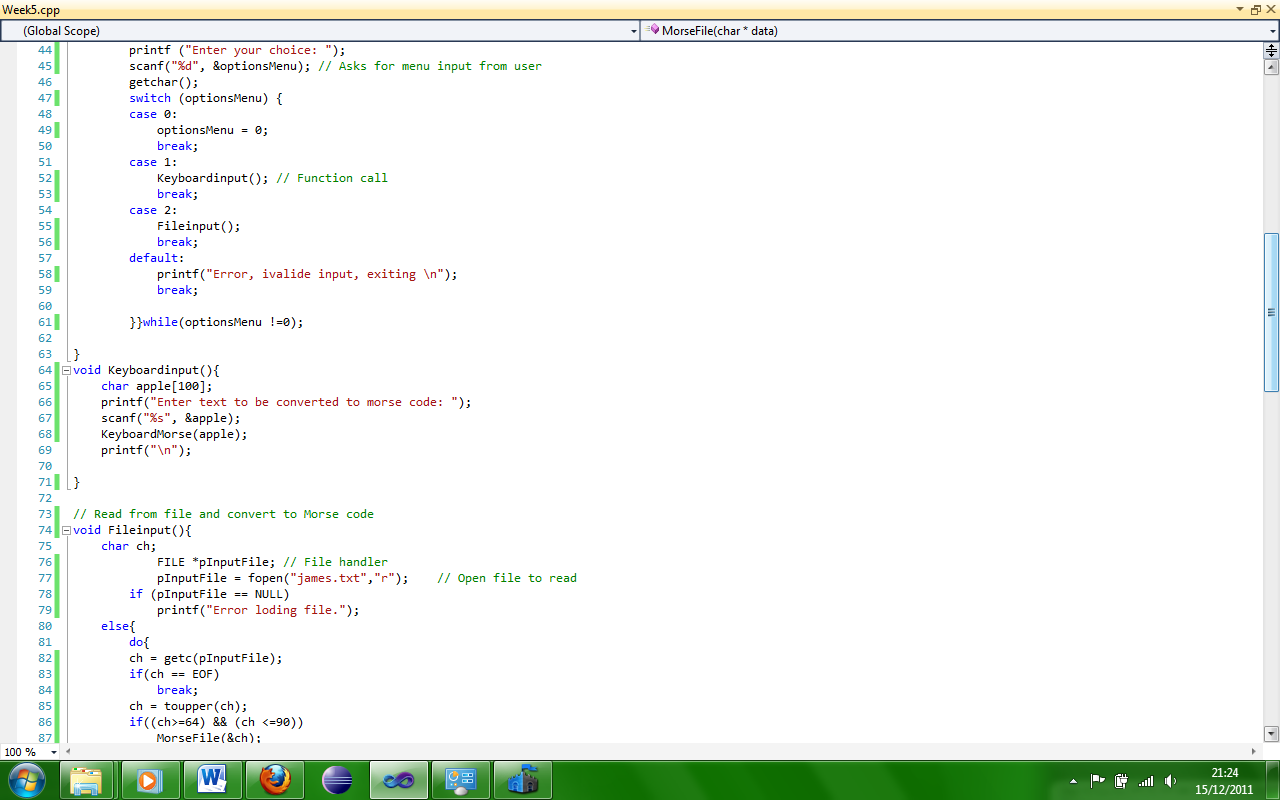
This program coverts a string into Morse code, it gets the string by either keyboard input or by file input.

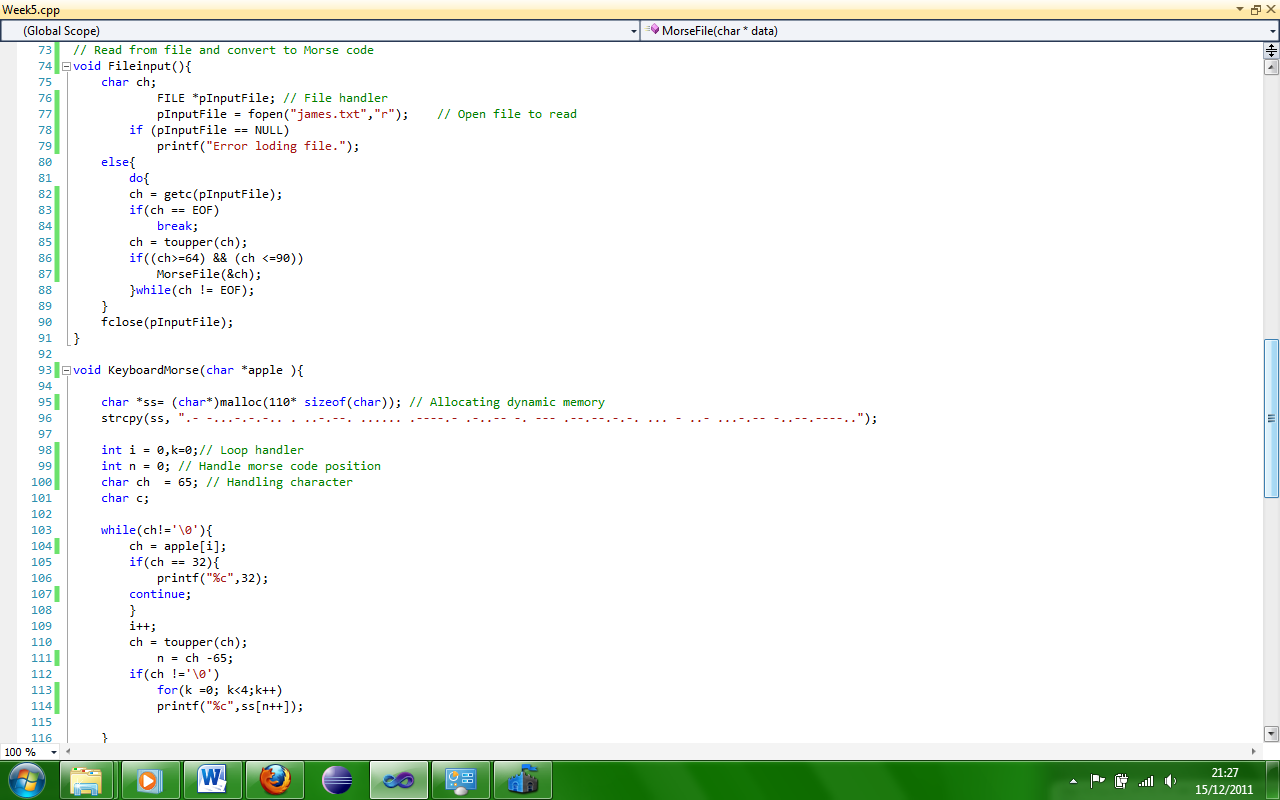
This part of the program prints out a menu on the screen when run which helps the user choose an option.

Here the program asks for input from the user and whatever the user inputs will determine which option is chosen.

Here is the code for the keyboard input option it sets the buffer to 100 characters, prints a message to the screen and scans the users input.



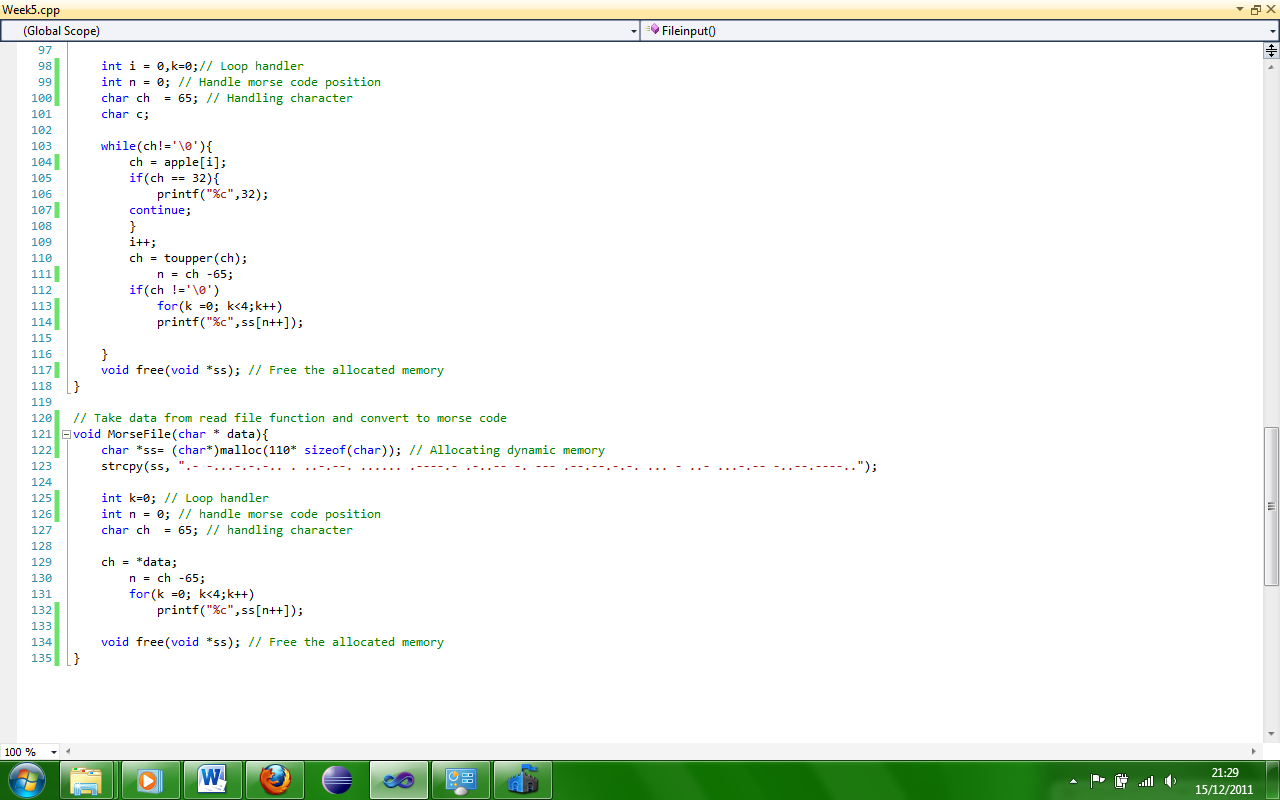


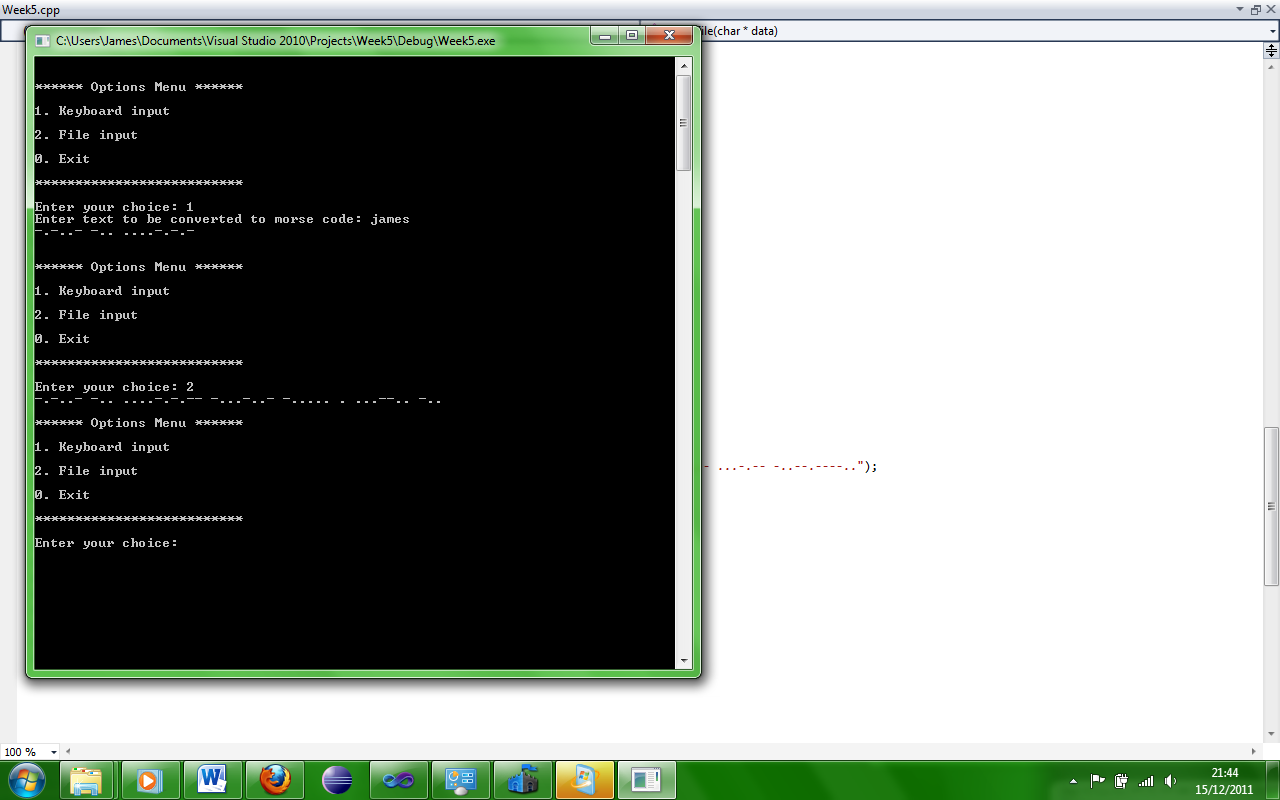


Here is the code for the file input option which opens the file and checks it for characters if it is not empty it converts these to Morse code.

Here the program handles the output for the keyboard Morse option; it allocates dynamic memory as well.

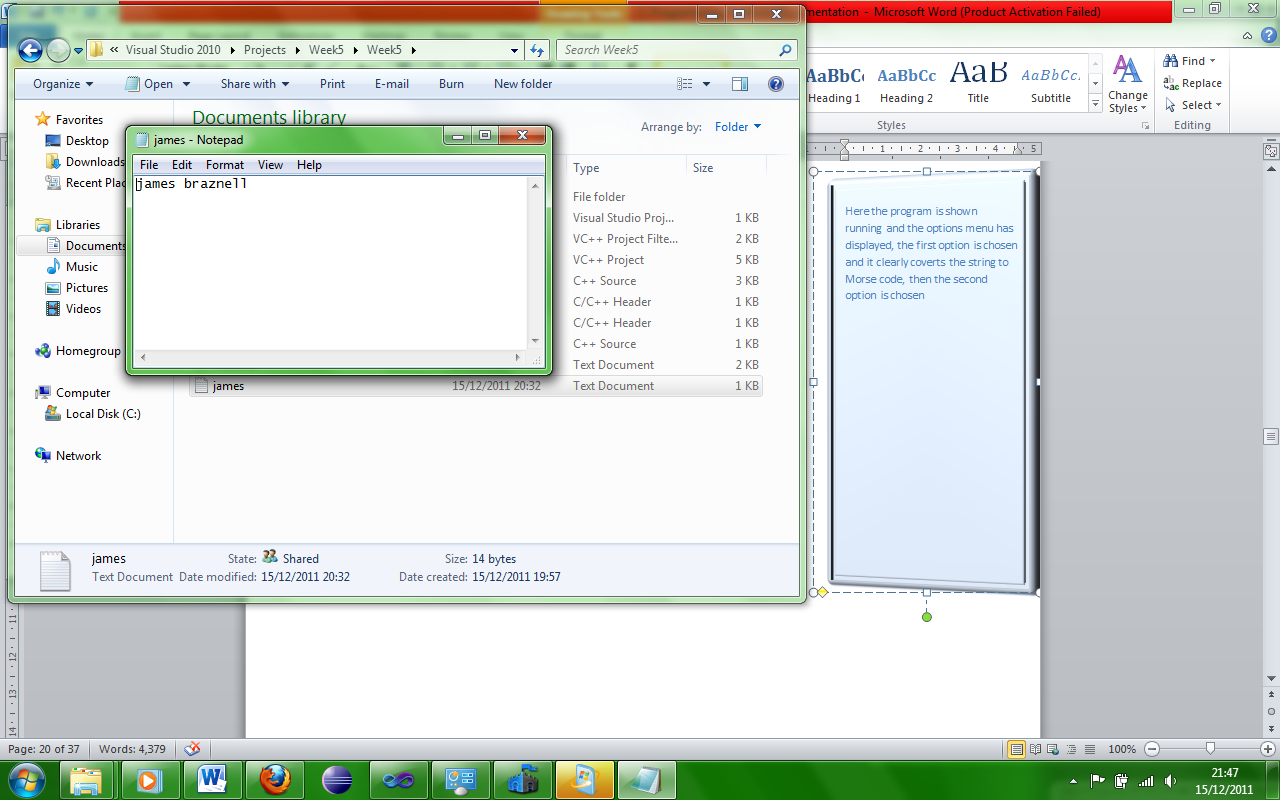
Here the program handles the output for the Morse file option; it allocates dynamic memory as well.





Here the program is shown running and the options menu has displayed, the first option is chosen and it clearly coverts the string to Morse code, then the second option is chosen and the program clearly converts that to Morse code as well.

Here is the text file that is used for the file input in this program.



**PORTFOLIO QUESTION 5 of 6. SIGN OFF WEEK 10. 20 marks**

**This question is to be signed off at the start of your normal workshop class only. The program should be included in the portfolio due in during week 13. For each portfolio question you need to include, question, design (limited), code listing, screen shots to show testing, brief comments.**

*If you fail to demonstrate or have failed to meet the minimum pass grade you have the opportunity re-demonstrate your work again the following week* ***(Note this MUST be the following week).*** *If a pass grade is achieved a capped D grade will be awarded.*

Part 1. (12 marks)

Create a Win32 application, using the default application as a base and display a simple message on the main window. Add a menu item (“COLOUR”) with four sub menus ‘RED’, ‘BLUE’, ‘GREEN and ‘CHOOSE’. When selected the text should be displayed in the appropriate colour. When the CHOOSE sub menu is selected the CHOOSECOLOR DIALOG should be displayed offering the user the option to select a colour for the text.

Part 2. (8 marks)

(a) Modify the application to include another menu item called “FONT”. The sub menus should be “COURIER”, “COMIC” .

When selected the text should be displayed in the appropriate font.

(b) Add an additional sub menu “CHOOSE”, when the CHOOSE sub menu is selected the CHOOSEFONT DIALOG should be displayed offering the user the option to select a font for the text.

**Note. Part 2(a) and in particular part(b) are to be considered as a research and implementation section. The information is not provided in the lecture notes. You are expected to research and investigate methods of implementing this part of the portfolio question.**

**Criteria :**

Part 1.

1. Menu “COLOUR” implemented with sub menu items.

No need to have any functionality but must run without errors. (2 marks)

2. A Text message displayed on the main window. (2 marks)

3. Change the colour of the Text via the menu items. (4 marks)

4. Change the colour of the Text using the CHOOSECOLOR Dialog. (4 marks)

Part 2.

1. Menu “FONT” implemented with sub menu items.

No need to have any functionality but must run without errors. (1 marks)

2. Change the font to “COURIER” or “COMIC”. (3 marks)

3. Change the font and colour using the CHOOSEFONT Dialog. (4 marks)

**Code Listing:**

// portfolio5.cpp : Defines the entry point for the application.

//

#include "stdafx.h"

#include "portfolio5.h"

#include "commdlg.h" // we need this for the Colour chooser dialog box

#define MAX\_LOADSTRING 100

// Global Variables:

HINSTANCE hInst; // current instance

TCHAR szTitle[MAX\_LOADSTRING]; // The title bar text

TCHAR szWindowClass[MAX\_LOADSTRING]; // the main window class name

COLORREF g\_rgbText = RGB(0, 0, 0);

COLORREF g\_rgbBackground = RGB(255, 255, 255);

COLORREF g\_rgbCustom[16] = {0};

HGDIOBJ g\_hfFont = GetStockObject(DEFAULT\_GUI\_FONT);

HGDIOBJ hf;

HDC hdc;

void Font(HWND hwnd){

long lfHeight;

hdc = GetDC(NULL);

lfHeight = -MulDiv(12, GetDeviceCaps(hdc,

LOGPIXELSY), 72);

ReleaseDC(NULL, hdc);

hf = CreateFont(lfHeight,0,0,0,0,TRUE,0,0,0,0,0,0,0,"Arial");

if(hf) {

DeleteObject(g\_hfFont);

g\_hfFont = hf;

}else{

MessageBox(hwnd, "Font creation failed!","Error", MB\_OK | MB\_ICONEXCLAMATION);

}

}

/\* This is the method to display the colour chooser dialog

It should show the current colour and 'remember custom colours

\*/

void DoSelectColour(HWND hwnd) {

/\* CHOOSECOLOR is a struct that controls the appearance

and behaviour of the ChooseColor dialog

\*/

CHOOSECOLOR cc = {sizeof(CHOOSECOLOR)};

cc.Flags = CC\_RGBINIT | CC\_FULLOPEN | CC\_ANYCOLOR;

cc.hwndOwner = hwnd;

cc.rgbResult = g\_rgbText; // this shows the current colour

cc.lpCustColors = g\_rgbCustom; // the array of 'custom' colours

if(ChooseColor(&cc)) // displays the dialog - returns

// true on OK, false on Cancel

{

g\_rgbText = cc.rgbResult;

}

}

void DoSelectFont(HWND hwnd)

{

CHOOSEFONT cf = {sizeof(CHOOSEFONT)};

LOGFONT lf;

GetObject(g\_hfFont, sizeof(LOGFONT), &lf);

cf.Flags = CF\_EFFECTS | CF\_INITTOLOGFONTSTRUCT | CF\_SCREENFONTS;

cf.hwndOwner = hwnd;

cf.lpLogFont = &lf;

cf.rgbColors = g\_rgbText;

if(ChooseFont(&cf))

{

HFONT hf = CreateFontIndirect(&lf);

if(hf)

{

g\_hfFont = hf;

}else{

MessageBox(hwnd, "Font creation failed!", "Error", MB\_OK | MB\_ICONEXCLAMATION);

}

g\_rgbText = cf.rgbColors;

}

}

//This is the method to draw the text in the window

void DrawMessage(HDC hdc, RECT\* prc)

{

SetBkColor(hdc, g\_rgbBackground);

SetTextColor(hdc, g\_rgbText);

char szMsg[] = "J.T.Braznell";

{

// Set object to the new font selected

HGDIOBJ hfOld = SelectObject(hdc,g\_hfFont);

/\* we want to change the 'top' of the rectangle so that it

is half the original height.

\*/

prc->top = (prc->bottom - prc->top) /2;

/\* DrawText takes a number of parameters:

a handle to the display area to draw on

a pointer to the string to draw

the number of characters to draw (-1 if all)

a RECT struct to format the text into

format control

\*/

DrawText(hdc, szMsg, -1, prc, DT\_WORDBREAK | DT\_CENTER);

/\* We could arrange the vertical centering of the text

by using DT\_SINGLELINE | DT\_CENTER | DT\_VCENTER

but this does not allow the word-wrap feature

\*/

SelectObject(hdc, hfOld);

DeleteObject(hf);//Delete the font - we have finished with it.

ReleaseDC(NULL, hdc);

}

}

// Forward declarations of functions included in this code module:

ATOM MyRegisterClass(HINSTANCE hInstance);

BOOL InitInstance(HINSTANCE, int);

LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);

INT\_PTR CALLBACK About(HWND, UINT, WPARAM, LPARAM);

int APIENTRY \_tWinMain(HINSTANCE hInstance,

HINSTANCE hPrevInstance,

LPTSTR lpCmdLine,

int nCmdShow)

{

UNREFERENCED\_PARAMETER(hPrevInstance);

UNREFERENCED\_PARAMETER(lpCmdLine);

// TODO: Place code here.

MSG msg;

HACCEL hAccelTable;

// Initialize global strings

LoadString(hInstance, IDS\_APP\_TITLE, szTitle, MAX\_LOADSTRING);

LoadString(hInstance, IDC\_PORTFOLIO5, szWindowClass, MAX\_LOADSTRING);

MyRegisterClass(hInstance);

// Perform application initialization:

if (!InitInstance (hInstance, nCmdShow))

{

return FALSE;

}

hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC\_PORTFOLIO5));

// Main message loop:

while (GetMessage(&msg, NULL, 0, 0))

{

if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))

{

TranslateMessage(&msg);

DispatchMessage(&msg);

}

}

return (int) msg.wParam;

}

//

// FUNCTION: MyRegisterClass()

//

// PURPOSE: Registers the window class.

//

// COMMENTS:

//

// This function and its usage are only necessary if you want this code

// to be compatible with Win32 systems prior to the 'RegisterClassEx'

// function that was added to Windows 95. It is important to call this function

// so that the application will get 'well formed' small icons associated

// with it.

//

ATOM MyRegisterClass(HINSTANCE hInstance)

{

WNDCLASSEX wcex;

wcex.cbSize = sizeof(WNDCLASSEX);

wcex.style = CS\_HREDRAW | CS\_VREDRAW;

wcex.lpfnWndProc = WndProc;

wcex.cbClsExtra = 0;

wcex.cbWndExtra = 0;

wcex.hInstance = hInstance;

wcex.hIcon = LoadIcon(hInstance, MAKEINTRESOURCE(IDI\_PORTFOLIO5));

wcex.hCursor = LoadCursor(NULL, IDC\_ARROW);

wcex.hbrBackground = (HBRUSH)(COLOR\_WINDOW+1);

wcex.lpszMenuName = MAKEINTRESOURCE(IDC\_PORTFOLIO5);

wcex.lpszClassName = szWindowClass;

wcex.hIconSm = LoadIcon(wcex.hInstance, MAKEINTRESOURCE(IDI\_SMALL));

return RegisterClassEx(&wcex);

}

//

// FUNCTION: InitInstance(HINSTANCE, int)

//

// PURPOSE: Saves instance handle and creates main window

//

// COMMENTS:

//

// In this function, we save the instance handle in a global variable and

// create and display the main program window.

//

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)

{

HWND hWnd;

hInst = hInstance; // Store instance handle in our global variable

hWnd = CreateWindow(szWindowClass, szTitle, WS\_OVERLAPPEDWINDOW,

CW\_USEDEFAULT, 0, CW\_USEDEFAULT, 0, NULL, NULL, hInstance, NULL);

if (!hWnd)

{

return FALSE;

}

ShowWindow(hWnd, nCmdShow);

UpdateWindow(hWnd);

return TRUE;

}

//

// FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)

//

// PURPOSE: Processes messages for the main window.

//

// WM\_COMMAND - process the application menu

// WM\_PAINT - Paint the main window

// WM\_DESTROY - post a quit message and return

//

//

LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)

{

int wmId, wmEvent;

PAINTSTRUCT ps;

HDC hdc;

switch (message)

{

case WM\_COMMAND:

wmId = LOWORD(wParam);

wmEvent = HIWORD(wParam);

// Parse the menu selections:

switch (wmId)

{

case IDM\_ABOUT:

DialogBox(hInst, MAKEINTRESOURCE(IDD\_ABOUTBOX), hWnd, About);

break;

case IDM\_EXIT:

DestroyWindow(hWnd);

break;

case ID\_COLOUR\_RED: // Colour Menu - Red

g\_rgbText = RGB(255, 0, 0); // set the current colour

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_COLOUR\_GREEN: // Colour Menu - Green

g\_rgbText = RGB(0, 255, 0); // set the current colour

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_COLOUR\_BLUE: // Colour Menu - Blue

g\_rgbText = RGB(0, 0, 255); // set the current colour

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_COLOUR\_CHOOSE: // Colour Menu - Select...

DoSelectColour(hWnd); // call our method

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_FONT\_COURIER: // Font Menu - Courier

hdc = GetDC(NULL);

g\_hfFont = CreateFont((-MulDiv(12, GetDeviceCaps(hdc,LOGPIXELSY),72)),0,0,0,0,FALSE,0,0,0,0,0,0,0,\_T("Corier New"));

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_FONT\_COMIC: // Font Menu - Courier

hdc = GetDC(NULL);

g\_hfFont = CreateFont((-MulDiv(12, GetDeviceCaps(hdc,LOGPIXELSY),72)),0,0,0,0,FALSE,0,0,0,0,0,0,0,\_T("Comic Sans MS"));

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

case ID\_FONT\_CHOOSE: // Font Menu - Select...

DoSelectFont(hWnd); // call our method

InvalidateRect(hWnd, NULL, TRUE); // Force a Repaint of the window

UpdateWindow(hWnd);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

break;

case WM\_PAINT:

hdc = BeginPaint(hWnd, &ps);

// This is our code to call our DrawMessage method

RECT rcClient; // the DrawText function needs a client rectangle

GetClientRect(hWnd, &rcClient);

DrawMessage(hdc, &rcClient);

EndPaint(hWnd, &ps);

break;

case WM\_DESTROY:

PostQuitMessage(0);

break;

default:

return DefWindowProc(hWnd, message, wParam, lParam);

}

return 0;

}

// Message handler for about box.

INT\_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)

{

UNREFERENCED\_PARAMETER(lParam);

switch (message)

{

case WM\_INITDIALOG:

return (INT\_PTR)TRUE;

case WM\_COMMAND:

if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)

{

EndDialog(hDlg, LOWORD(wParam));

return (INT\_PTR)TRUE;

}

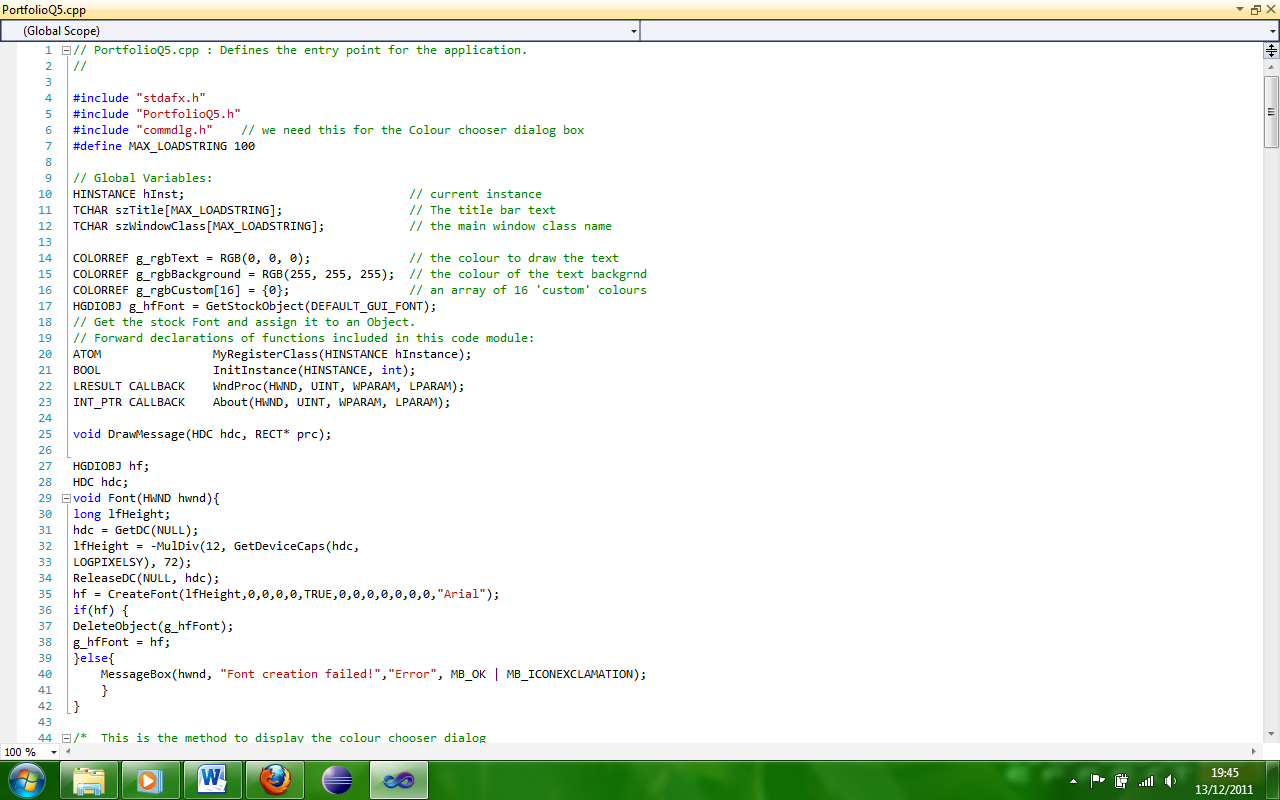
break;

}

return (INT\_PTR)FALSE;

}

**Screen shots:**

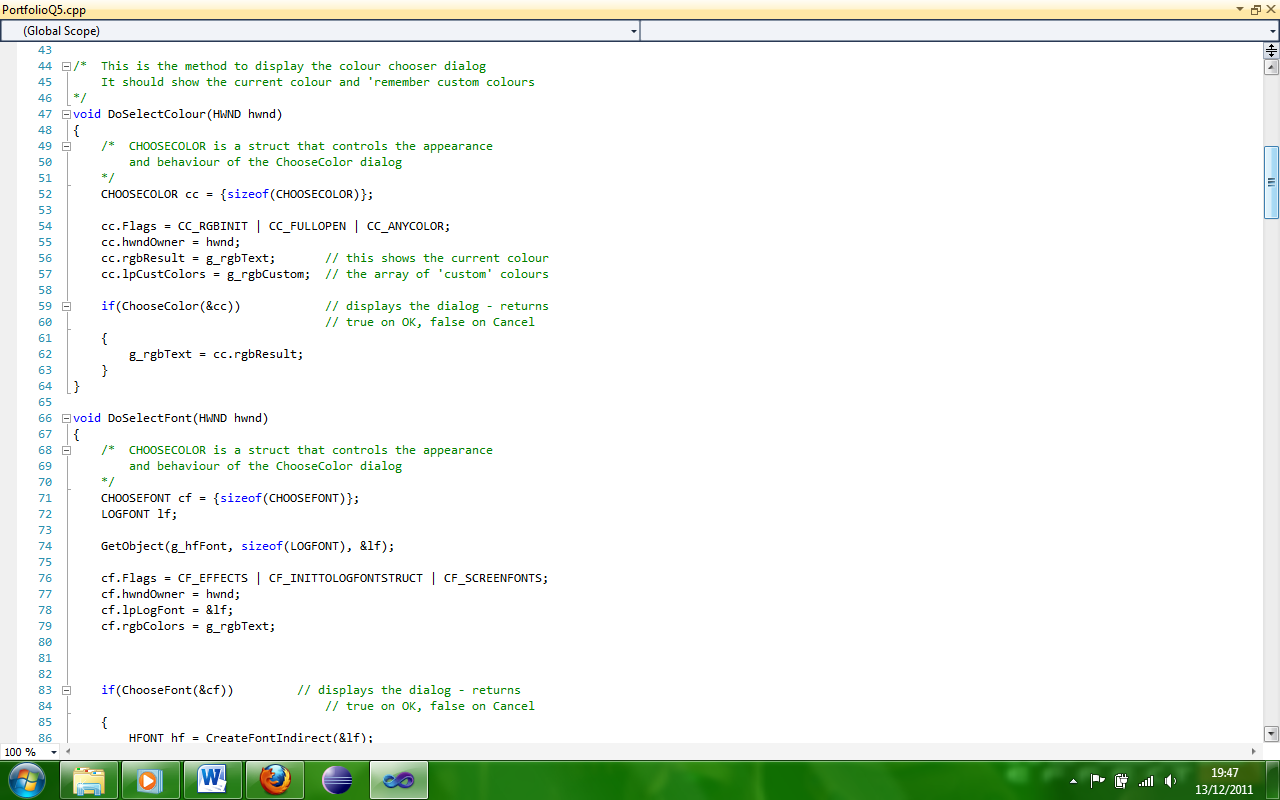


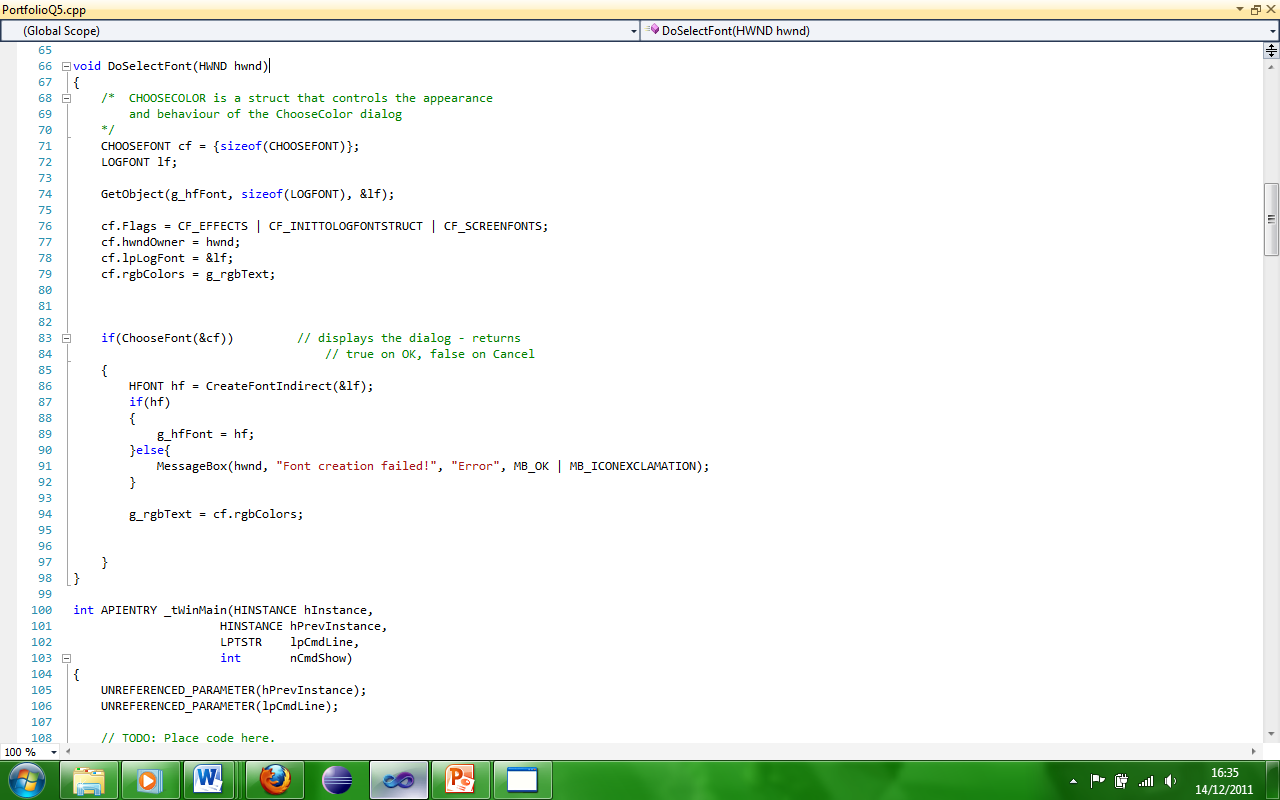
This win 32 program displays a message in the main window and it can be altered by the sub menus which allow for modifying the colour and font of the text respectively.

Here the program sets global variables as well as an array of colours; it also sets the text to a default font.

Here the program draws a message on the screen and sets its base attributes.

This part of the program is for when the user selects the choose colour sub-menu, it shows the current colour, an array of colours which can be customized and then saved.

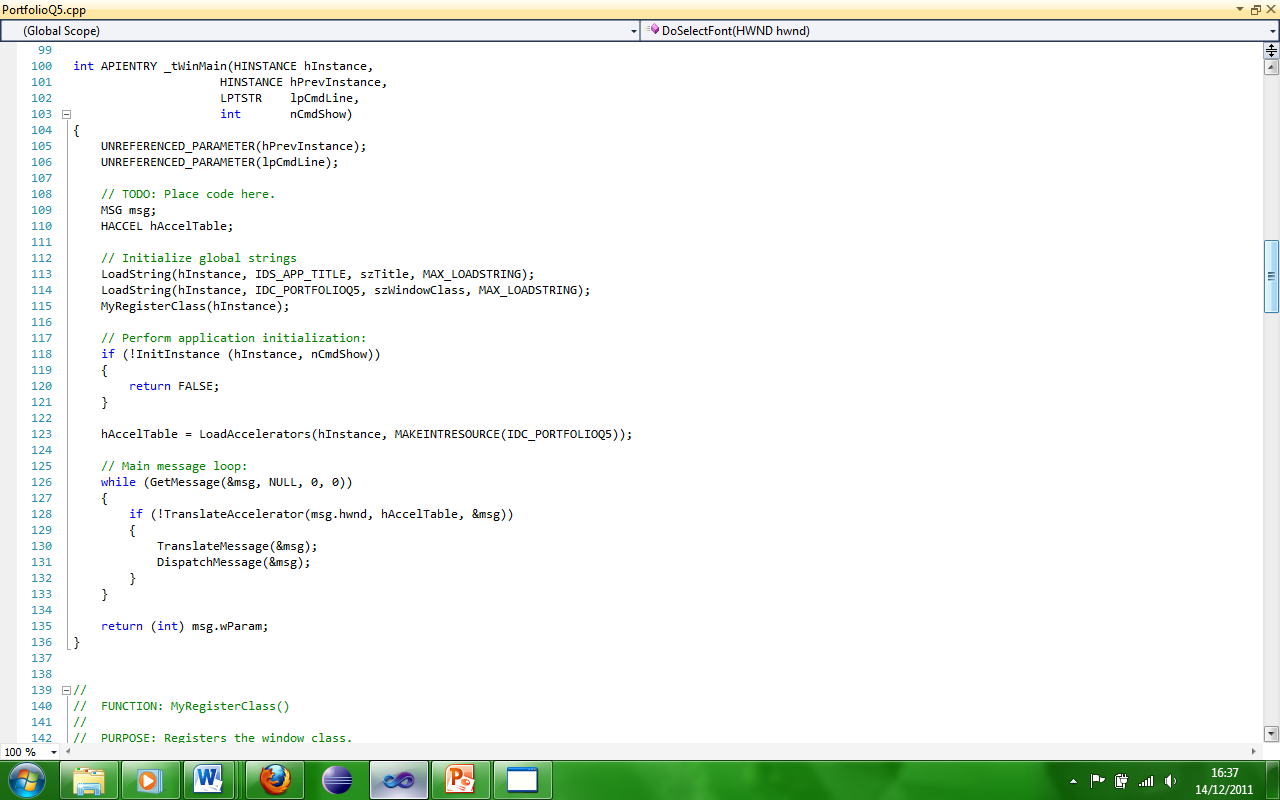


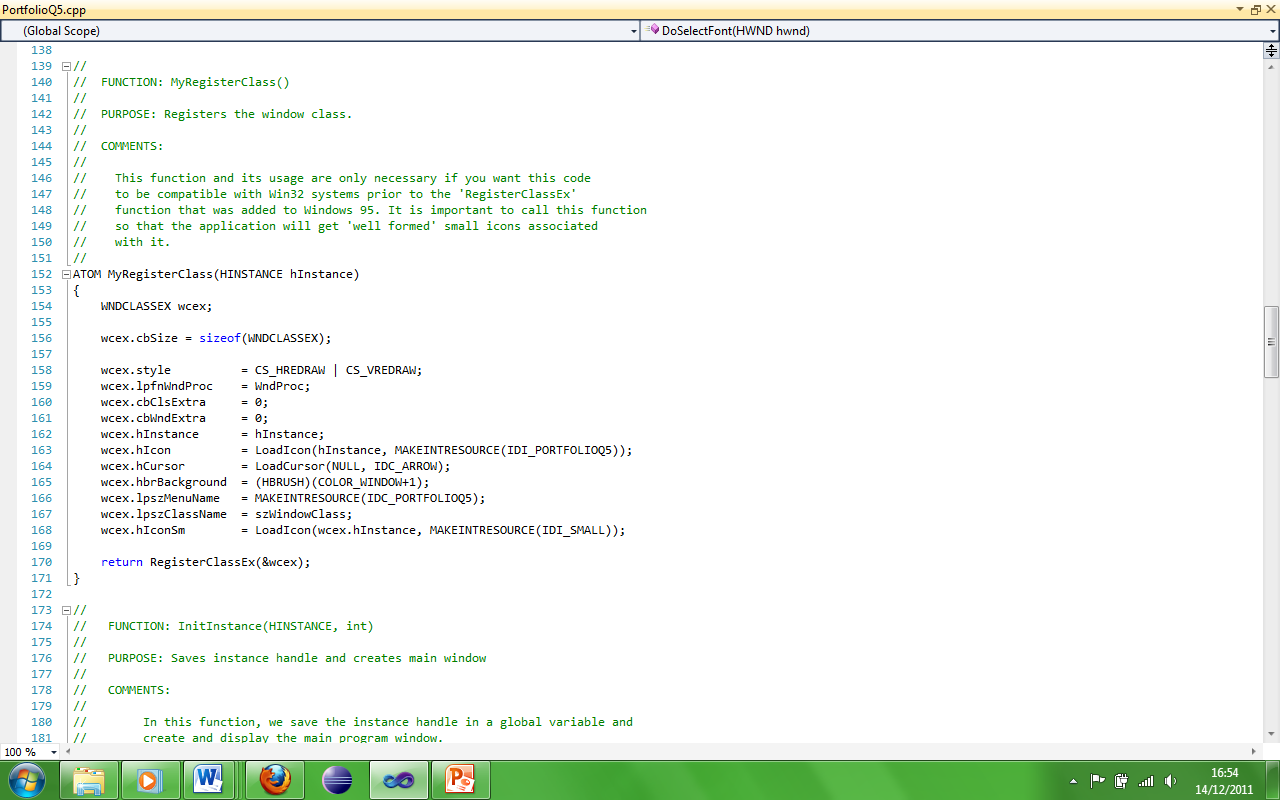


This part of the program sets the appearance of the choose colour dialog.

The if statement will display the dialog and will return true on OK and false on Cancel.

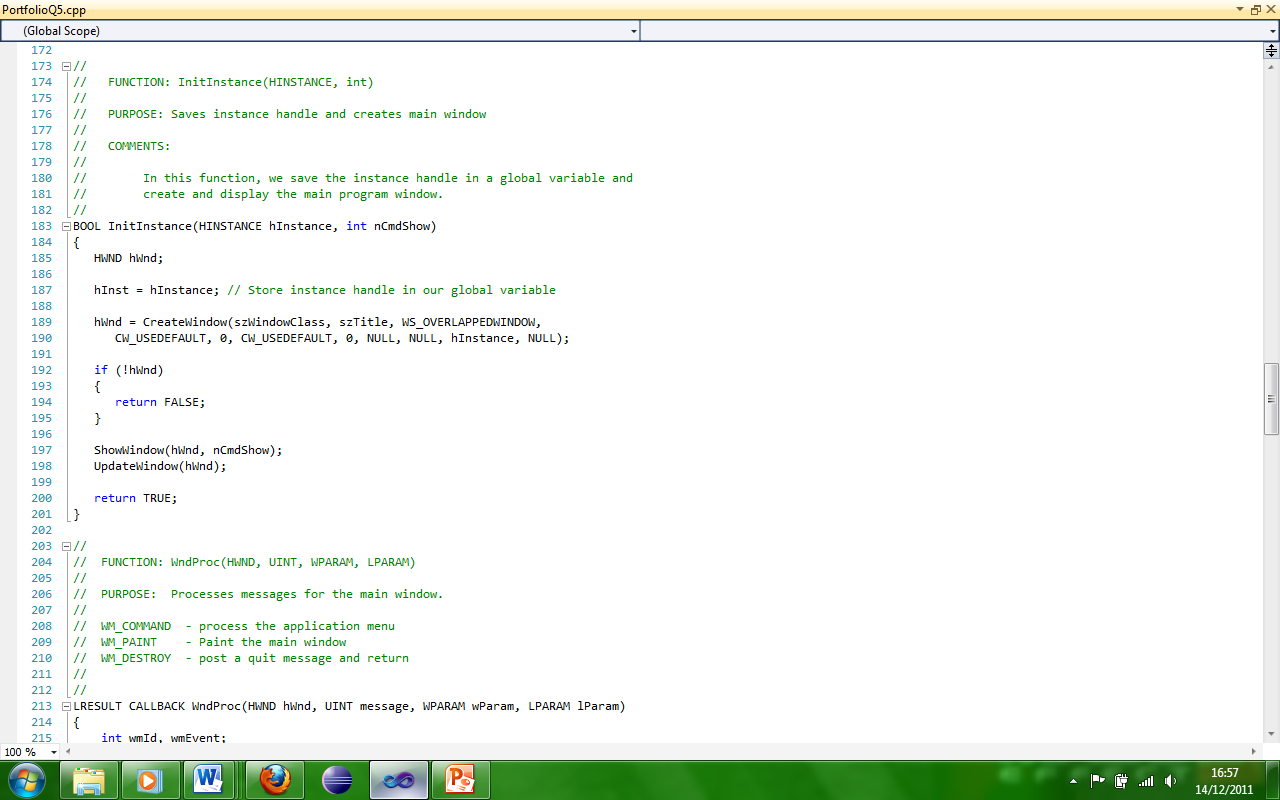
Initializes global strings, performs the application initialization and the main message loop.

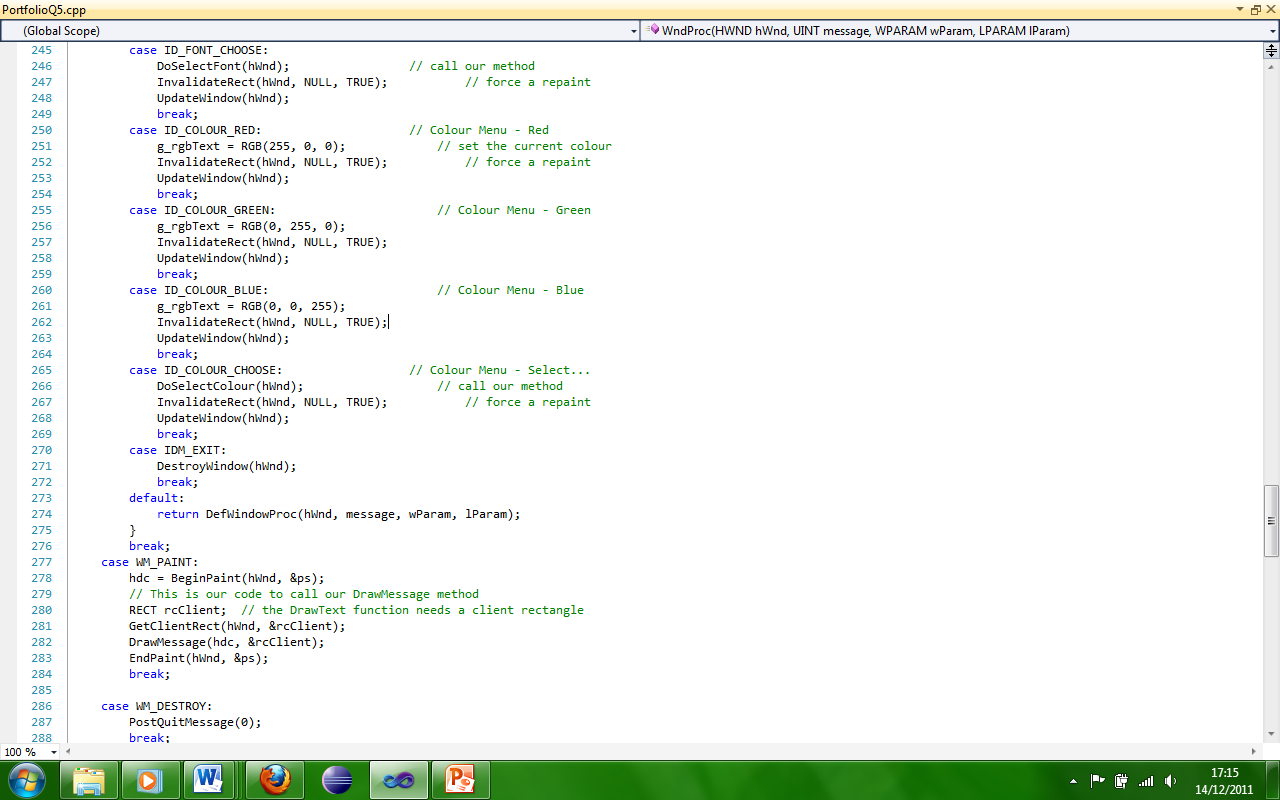
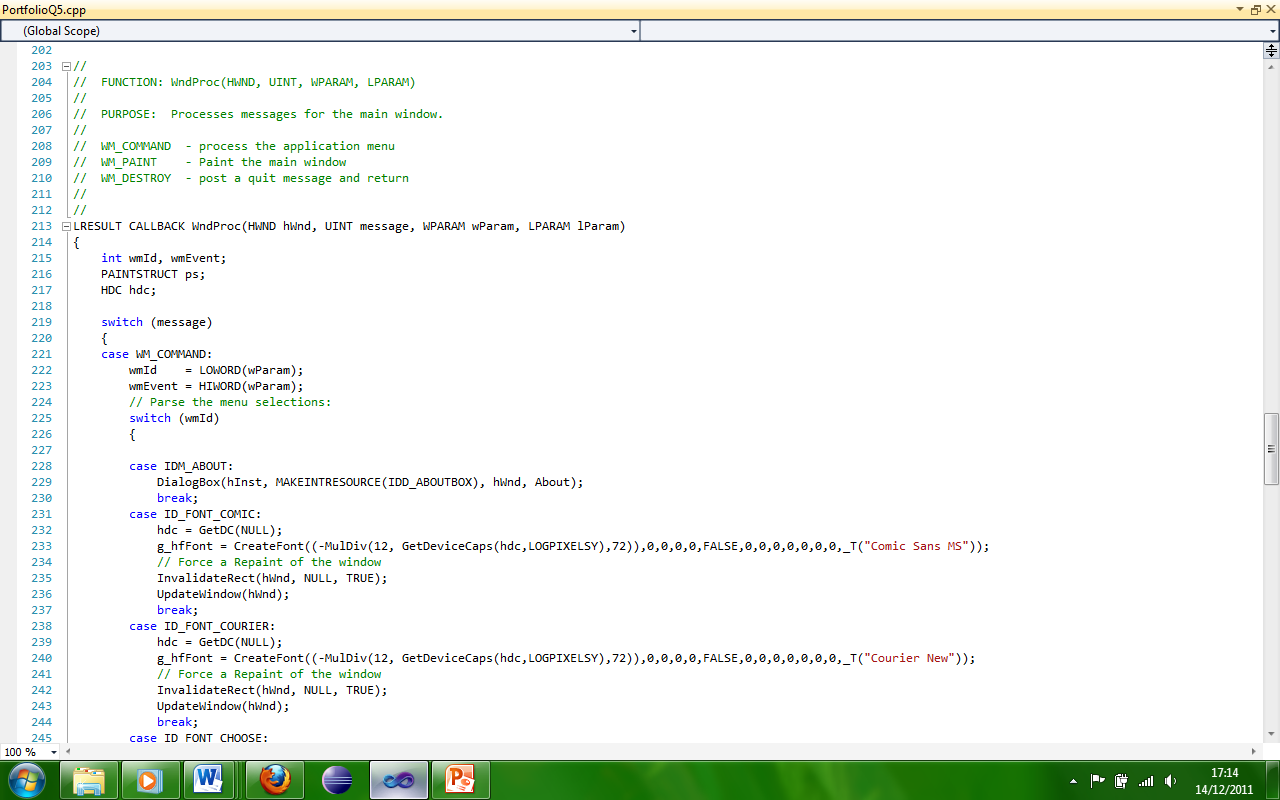




This part of the program registers the window class and is necessary for this code to work on systems prior to Windows 95.

This is the InitInstance function. It saves the instance handle and creates the main window.



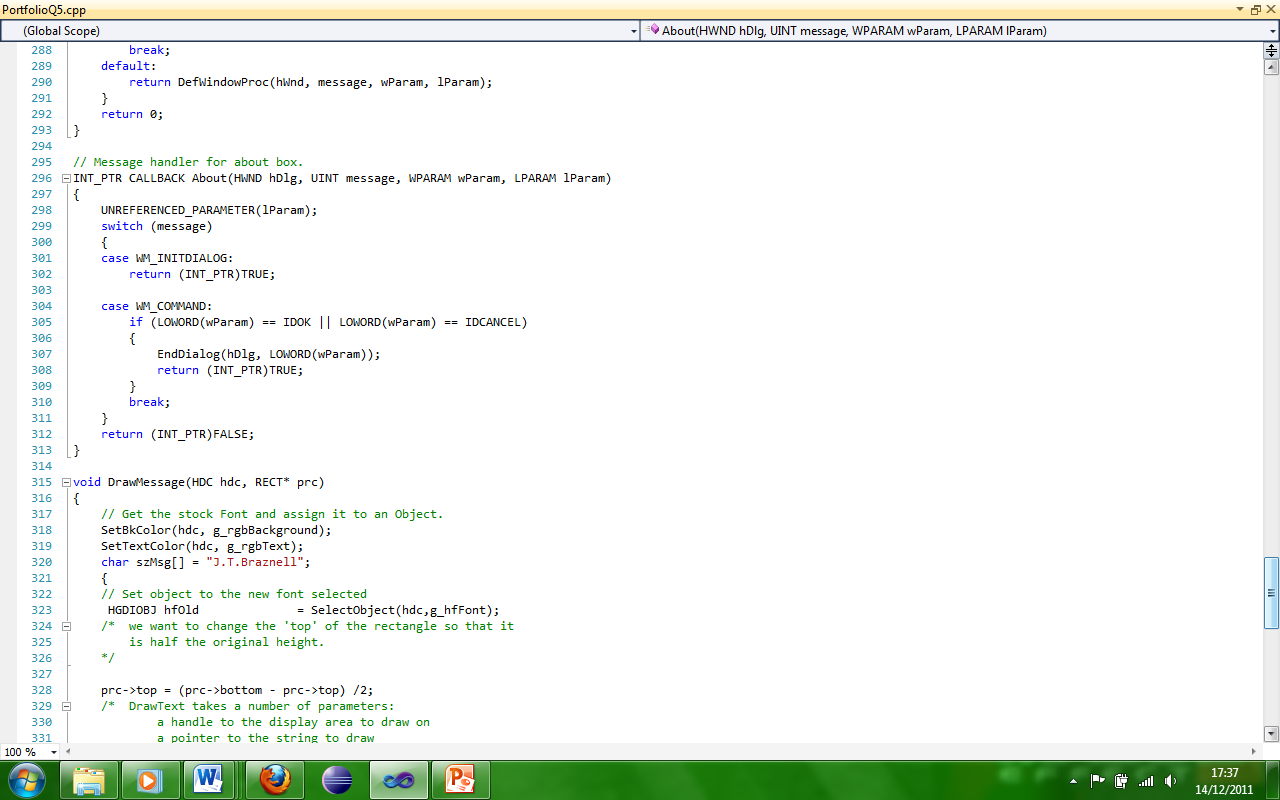


This part of the program processes messages for the main window.

Processes the application menu.

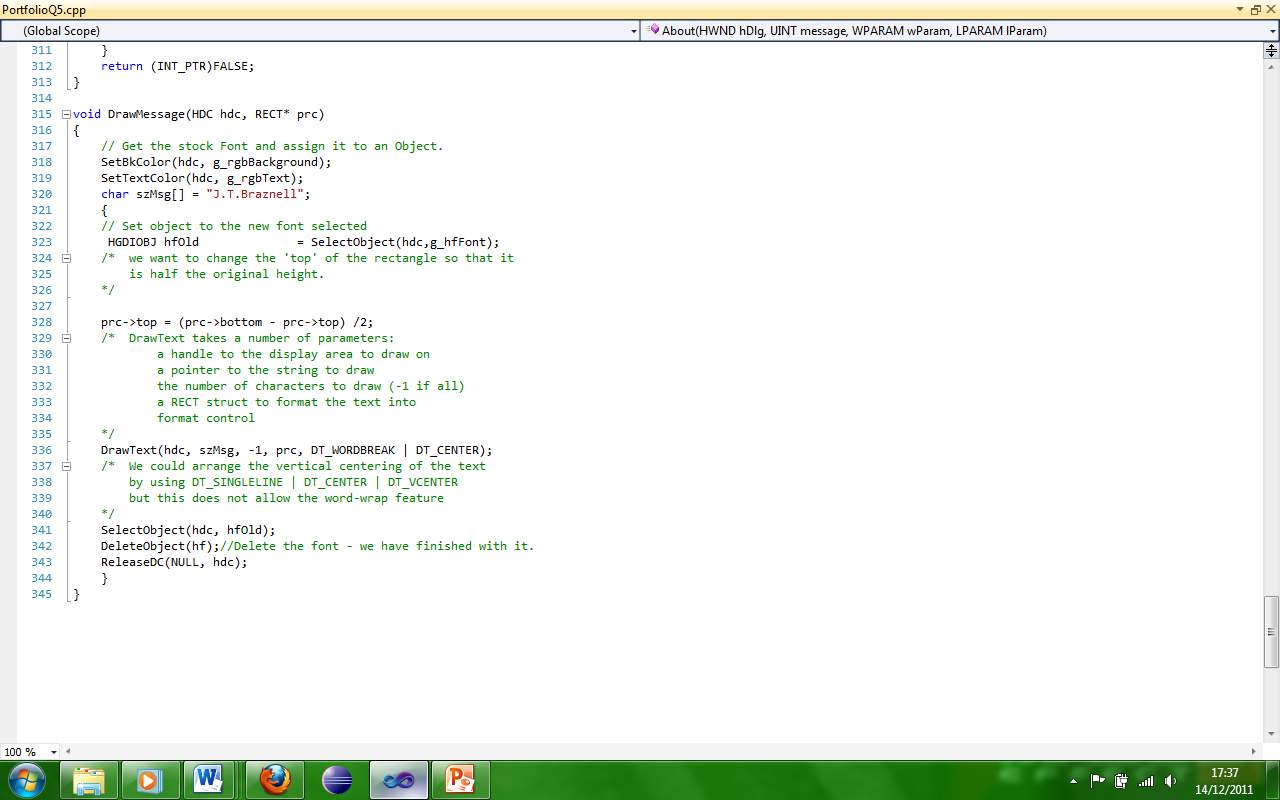
Paints the main window.

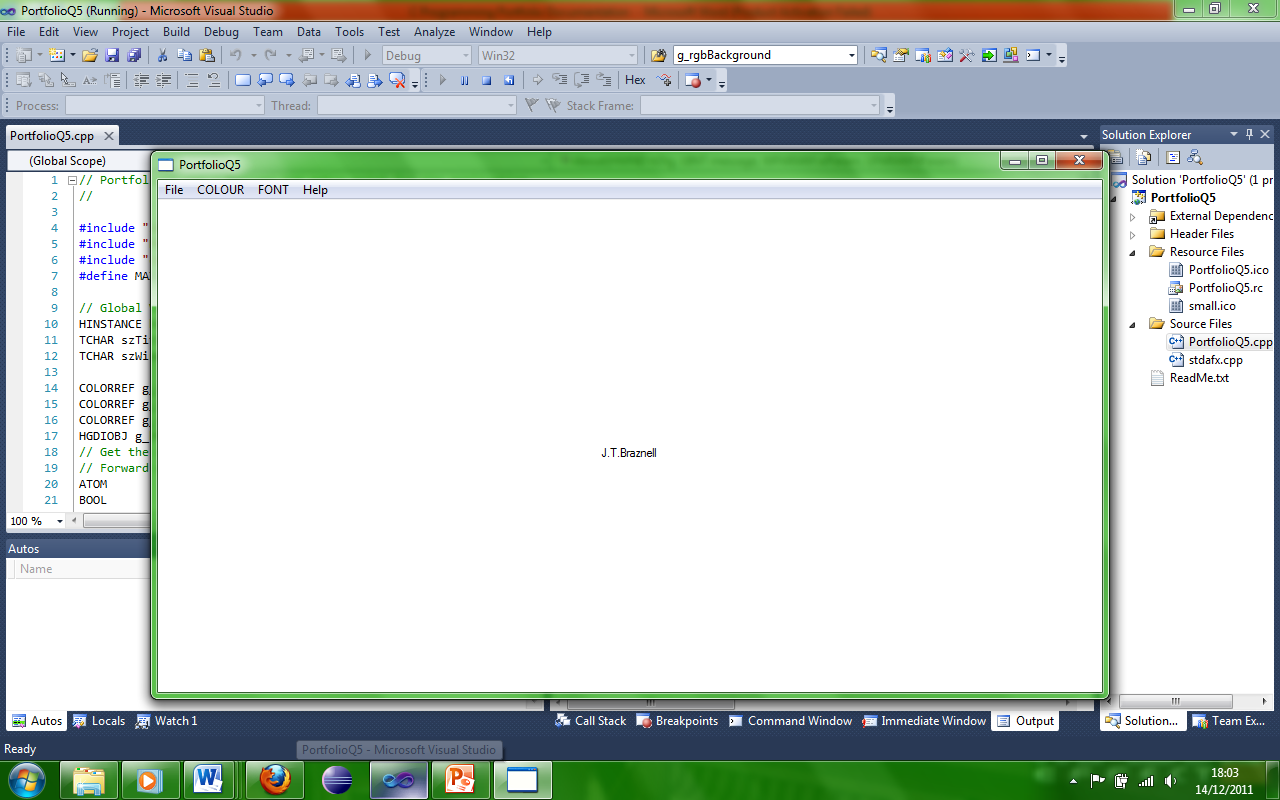
Post’s a quit message and return.



This code handles the call-back for the about box as well as the message handler.

Assigns the default font to the message displayed in the main window, also the message is written here.



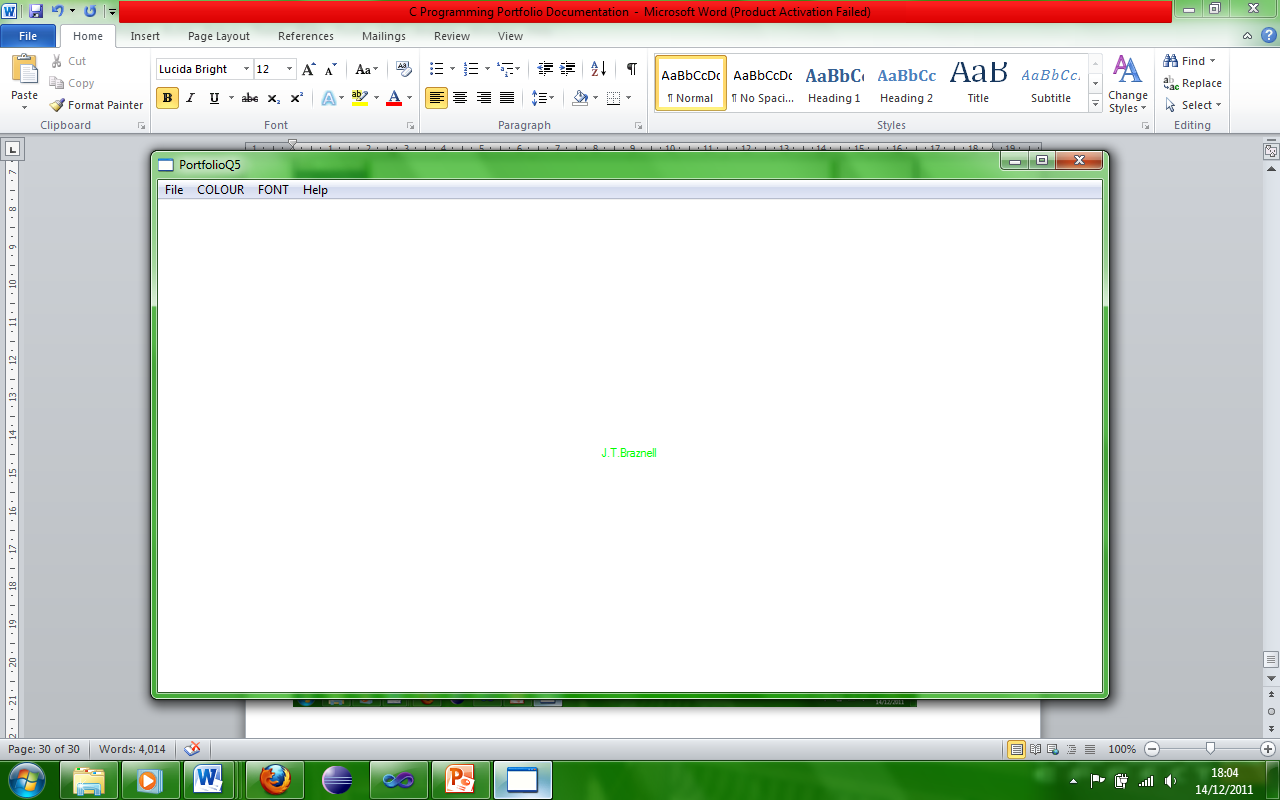


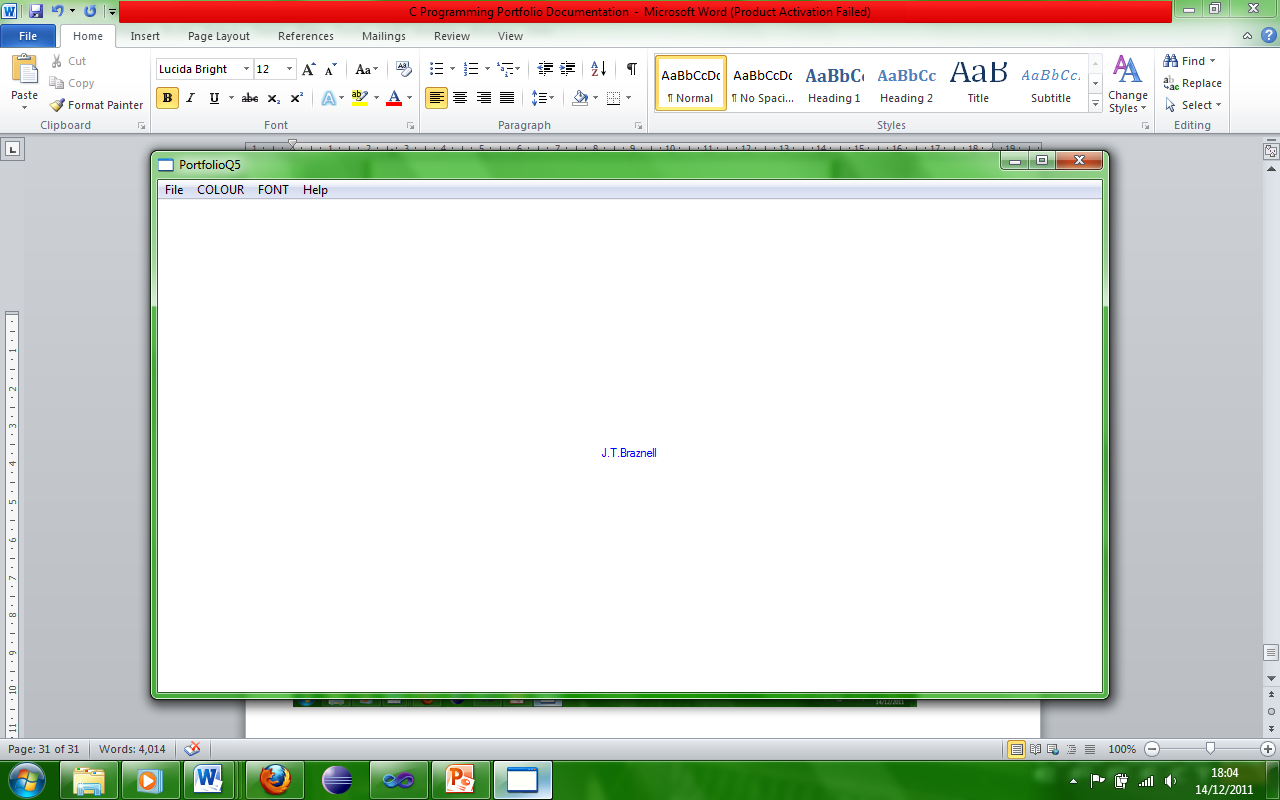
Here the program is working and the text is displayed in the main window with default font and colour.

The text is displayed but the colour has been changed to red.

The text is displayed but the colour has been changed to green.



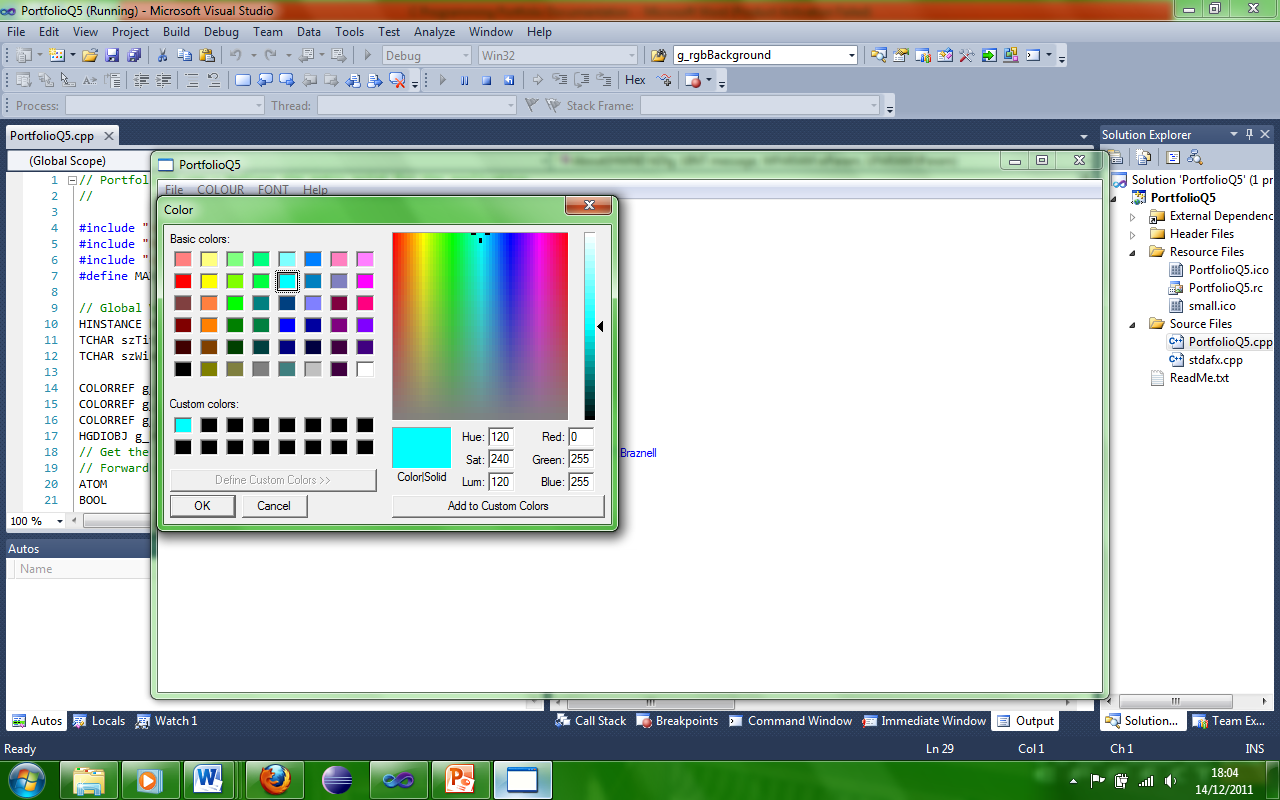




The text is displayed but the colour has been changed to blue.

Here the choose colour window shows a range of colours which can be modified to the users wishes and then added to custom colours which the program will remember until shutdown, when the user presses ok if they select a different colour the text will be changed accordingly.

The text is displayed but the colour has been changed to Aqua.

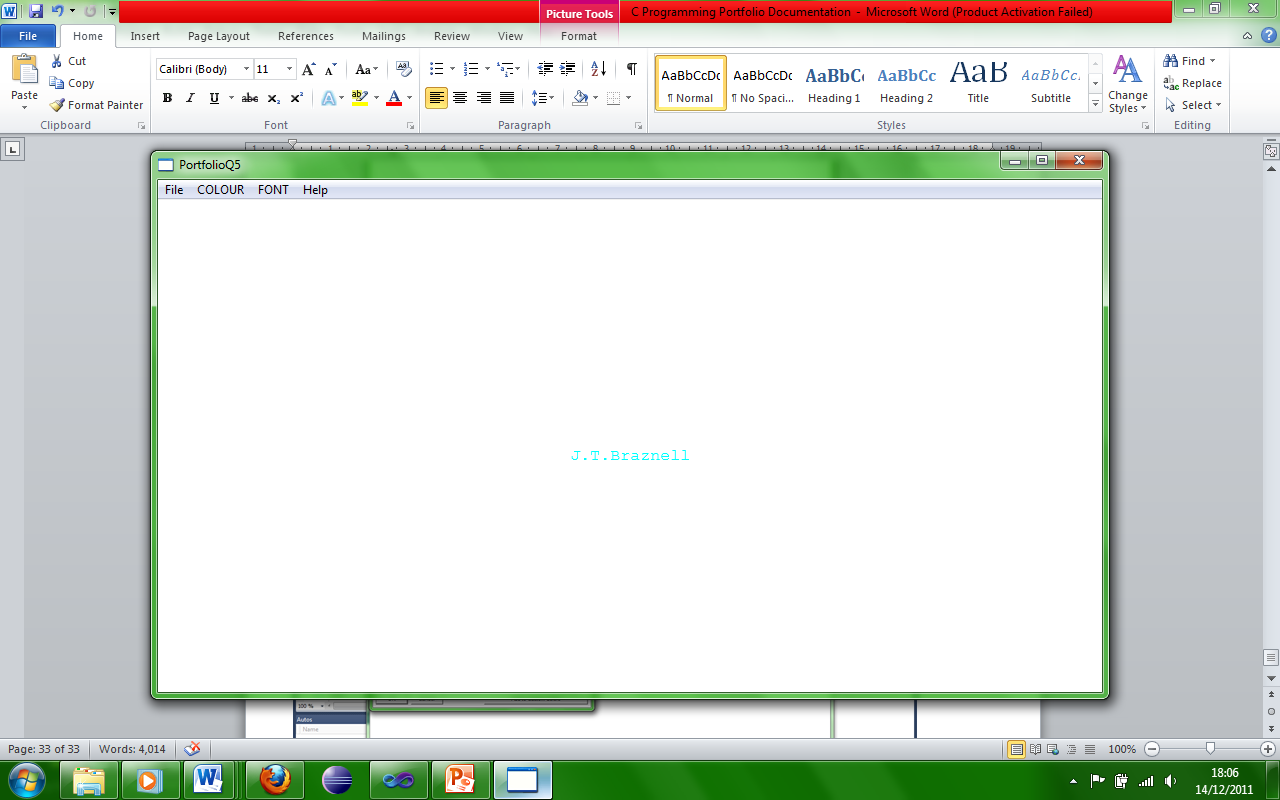


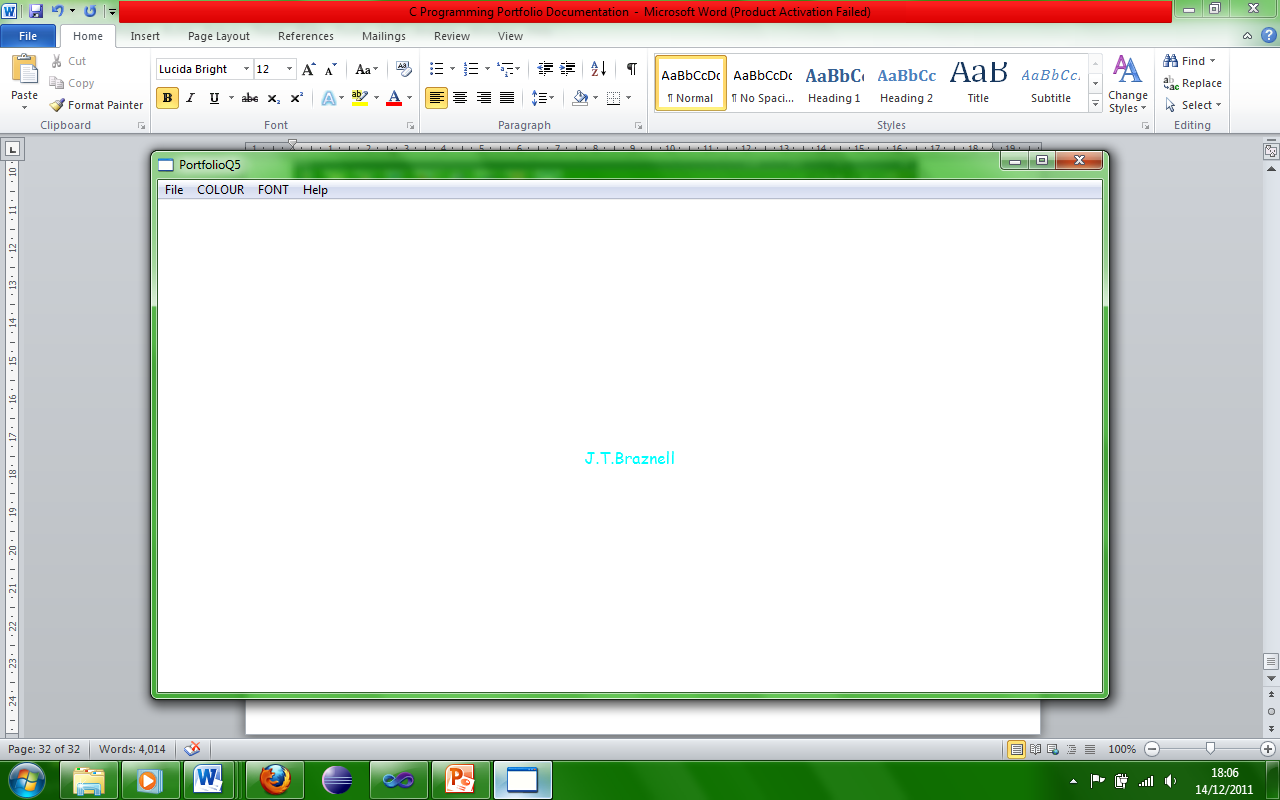


The text is displayed but the font has been changed to courier.

The text is displayed but the font has been changed to comic.

Here the choose font window shows a range of fonts, styles, sizes, effects, scripts as well as colours, when the user presses ok, any changes the user makes will be displayed in the main window.









Here the text is displayed and the font has clearly been changed.