Stolpersteine und Spezialitäten in C/C++

betreut von Aegidius Plüss, Gymnasium Bern-Neufeld

Heute beschreibe ich, wie man eine nicht ganz alltägliche Applikation entwickelt, die aber für die Schulumgebung eine wichtige Rolle spielen kann. Oft hatte ich die Aufgabe, für den Unterricht eine Simulation zu entwickeln, bei der des Benützerinterface eine wichtige Rolle spielt. Insbesondere sollte auf dem Bildschirm ein Hintergrundbild erzeugt werden, das so kompliziert ist, dass man es mit einem graphischen Editor erstellen muss, da es nicht zu Laufzeit mit den Graphikfunktionen von Champ aufgebaut werden konnte. Dazu muss man es möglich sein, Bilder in ein Champ-Fenster zu "laden". Die Champ-Bibliothek unterstützt dies für BMP-Bilddateien und für ICO-Ikonen. (Beides ist leider im Champ-Help noch nicht dokumentiert, funktioniert aber bestens.)

P55:

Wie schreibt man ein Programm, das eine vorgegebene Bilddatei während einer gewissen Zeit auf dem Bildschirm anzeigt und dann wieder verschwindet? Es soll auch möglich sein, mit einem Tastendruck das Bild vorzeitig zum Verschwinden zu bringen. (Ein mögliche Anwendung ist ein höflicher Willkommensgruss beim Einloggen auf einem Server.)

L55:

Man erzeugt das Bild mit irgend einer Graphik-Applikation, beispielsweise mit Adobe Photoshop. Dann speichert man dieses als BMP-Datei (in unserem Fall unter dem Namen LOGO.BMP).

Nun schreibt man eine Champ-Applikation (sie heisse LOGO.EXE) und gibt die BMP-Datei als Resource an. Dazu öffnet man mit einem Texteditor die Datei LOGO.RC. (Dies geht am besten, indem man mit der rechten Maustaste auf diese Datei im Projektfenster klickt und "*Anzeigen | Text bearbeiten*" auswählt.)

Nun fügt man die Zeile

```
Logo BITMAP "logo.bmp"
```

ein, wo Logo ein beliebiger Bezeichner ist, unter der man die Resource verwenden will. Man erzeugt im C++-Code mit

```
CPBitmap m;
```

ein Bitmap-Objekt, lädt dieses mit

```
m.loadBitmap( "Logo" );
```

und zeigt es (an der Pixelkoordinate 100/100) mit

```
m._put( wnd, 100, 100 );
```

an. Man beachte, dass wegen des **Underline** _ Pixelkoordinaten verwendet werden, wobei sich der Nullpunkt in der **oberen linken Ecke** befindet.

Der gesamte Code lautet:

```
// LOGO.CPP

#include <champ.h>
#include "cpbitmap.h"

void gmain ()
{
   CPWindow wnd( "", CP::WinMax );

   CPBitmap m;
   m.loadBitmap( "Logo" );
   m._put( wnd, 100, 100 );

   wnd._text( 50, 300 ) << "Willkommen auf dem Computernetz";

   CPAlarmTimer timer( 5000, true );
   while ( timer.isRunning() )
   {
      if ( kbhit() )
         timer.stop();
   }
}
```

Man beachte, dass es nötig ist, **cpbitmap.h** zu includen. Weitere Methoden zu Bitmap- und Ikon-Objekten entnimmt man dieser Datei.

P56:

Man möchte eine Simulation zur (nicht ausgeglichenen) Wheatstoneschen Brücke schreiben, wo die Brücke auf dem Bildschirm sichtbar ist und gewisse Teile davon zu Laufzeit verändert werden, etwa der eine der 4 Brückenwiderstände.

L56:

Da wir zur Laufzeit mit Champ-Graphik-Funktionen in das aus einer Bitmap stammende Bild schreiben wollen, müssen wir die Koordinaten möglichst exakt kennen. Am besten geht man wie folgt vor:

1. Man erstellt das Bild, beispielsweise mit Designer, in einer Grösse, die der gewünschten Bildschirmdimension entspricht. Man merkt sich anhand des eingeblendeten cm-Lineals Breite und Höhe. Jetzt wählt man alle Elemente aus und kopiert das Bild in die Zwischenablage.
2. Man startet eine gutes Bildbearbeitungsprogramm, beispielsweise Adobe Photoshop, und wählt eine neues Bild mit dem Bilddimensionen, die man sich gemerkt hat. Nun holt man das Bild aus der Zwischenablage. Um Bildpositionen zu bestimmen, wählt man einen Lineal in Pixel-Format (in Photoshop unter *Datei* | *Voreinstellungen* | *Masseinheiten* | *Lineale*).
3. Nun speichert man das Bild mit *Datei* | *Kopie speichern unter* als BMP-Datei, die man in der Resource-Datei des C++-Projekts, wie oben beschrieben, angibt. (Dies etwas umständliche Verfahren ist nötig, da die von Designer exportierten BMP-Dateien schlechte Auflösung aufweisen.)
4. Im Code wählt man zuerst eine passende Fenstergrösse, etwa mit
   ```
   #define WINWIDTH 700
   #define WINHEIGHT 600
   ```

5. Man wählt eventuell auch einen Offset von der linken oberen Ecke, etwa mit
   ```
   #define XOFFSET 10
   #define YOFFSET 20
   ```
6. Es ist nun einfach, einen exakten Ort in der Graphik festzulegen. Will man beispielsweise an einer bestimmten Stelle einen Float auschreiben, so bestimmt man im Photoshop mit dem Lineal die Pixel-Koordinaten x und y und kann diese Stelle im Code mit x + XOFFSET bzw. y + YOFFSET angeben. Man beachte, dass sich der Koordinatenursprung für die Pixelkoordinaten (Methoden, die mit _ beginnen) oben links befindet. Das Code-Gerüst ist das folgende:

```
CPWindow wnd( "Wheatstonesche Brücke",
              CPSize( WINWIDTH, WINHEIGHT ) );
CPBitmap m;
m.loadBitmap( "Wheat" );
m._put( wnd, XOFFSET, YOFFSET );

// Pixel coordinates determined with graphics editor
int x = 95;
int y = 75;
float value = 10.2;  // Just an example
wnd._textFont( "Times New Roman", -20 );
wnd._text( x + XOFFSET, y + YOFFSET ) << value;
```

P57:

Bei der Eingabe durch den Benützer gibt es böse Laufzeitfehler, falls dieser eine unerlaubte Eingabe (etwa Buchstaben statt Zahlen für einen float) macht. Wie kann man die Eingabe in C++ am einfachsten validieren?

L57:

Für die Validierung von Eingaben eignet sich hervorragend die Klasse istrstream. Man liest also, wie in alten Zeiten, die Eingabe immer als String ein und versucht den Eingabewert mit istrstream in das gewünschte Format zu konvertieren. Falls die Eingabe eine unerlaubtes Format besitzt, liefert die Methode fail() den Wert true zurück. Der Code ist im nachfolgenden Beispiel der Wheatstoneschen Brücke in der Funktion *validate* ersichtlich.

P58:

Im Zusammenhang mit mathematischen Problemen wäre eine C++-Bibliothek für Matrizenrechnung sehr praktisch. Insbesondere sollte man lineare Gleichungssystem lösen können. Wo findet man eine solche Bibliothek?

L58:

Auf dem Markt sind mehrere Bibliotheken für Matrizenoperationen erhältlich. In C++ lässt sich diese besonders schön schreiben, falls man dazu Templates einsetzt und die Operationen überladet. Damit erhält man eine Syntax, die fast vollständig mit der mathematischen Notation übereinstimmt! Der Einsatz von Templates erlaubt es, Matrizen für verschiedene Datentypen (int, doubles, usw.) gleichartig zu verwenden. Die hier vorgestellte, kostenfreie

Matrixbibliothek wird über eine Include-Datei matrix.h eingebunden. Typische Operationen sind aus folgender Demonstration ersichtlich:

```
// LOGO.CPP

#include <champ.h>
#include "matrix.h"

typedef matrix<double> Matrix;


ginit ()
{
   cinit( "Matrix Demo" );

   cout << "Creating two random number matrices M1 and M2:\n";
   Matrix m1;
   int i,j;

   srand( (unsigned)time( NULL ) );
   for ( i = 0; i < 4; i++ )
      for ( j = 0; j < 4; j++ )
         m1(i, j) = 12 / ((rand()%25)+1.0);
   m1.SetSize(4, 4);

   Matrix m2(4, 4);

   for ( i = 0; i < 4; i++ )
       for ( j = 0; j < 4; j++ )
          m2(i, j) = 12 / ((rand()%25)+1.0);

   cout << "\nThe matrix M1 is:\n" << m1 << endl;
   cout << "\nPress any key...\n";
   getch();

   cout << "\nThe matrix M2 is:\n" << m2 << endl;
   cout << "\nPress any key...\n";
   getch();

   Matrix m3 = m1 + m2;
   cout << "\nMatrix addition:\nM1 + M2 = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = m1 - m2;
   cout << "\nMatrix subtraction:\nM1 - M2 = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = m1 * m2;
   cout << "\nMatrix multiplication:\nM1 * M2 = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = !m1;
   cout << "\nMatrix inversion:\nInv M1 = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = m1 ^ 4;
   cout << "\nMatrix power:\nM ^ 4 = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = m1.Adj();
   cout << "\nAdjoint of M1:\nAdj M1 = \n" <<  m3 << endl;
   cout << "\nPress any key...\n";
   getch();

   cout << "\nCofactor of M1(0,0) = " << m1.Cofact(0,0) << endl;
```

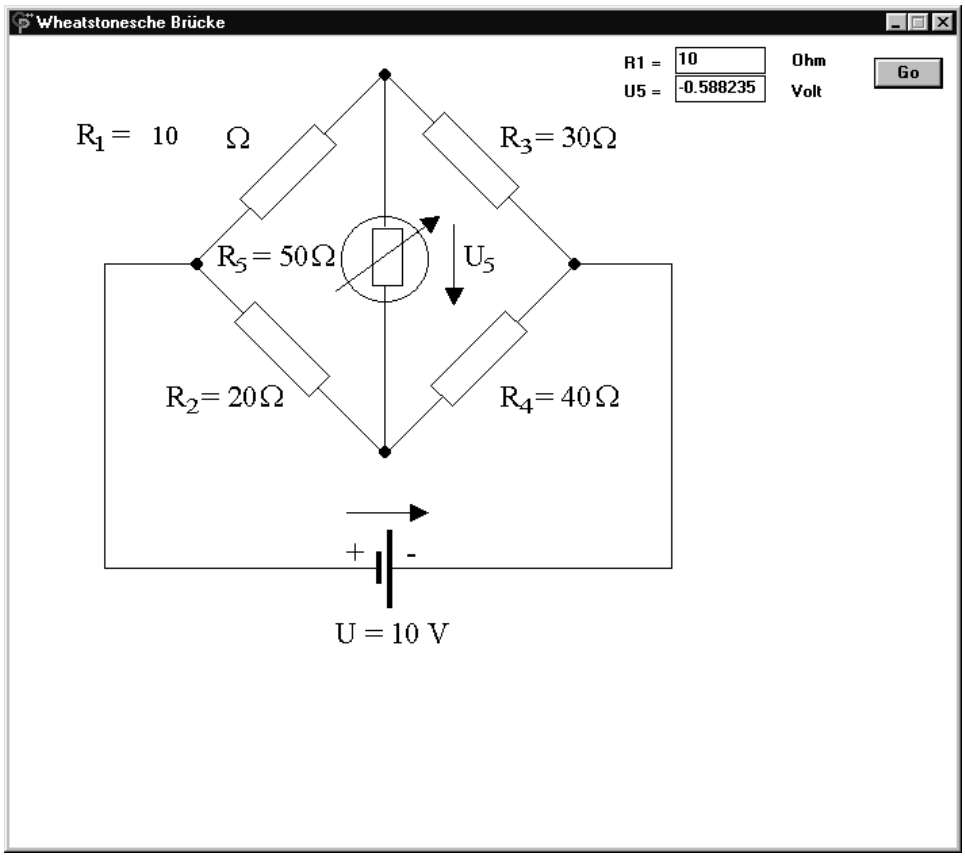
```
   cout << "Determinant of M1 = " << m1.Det() << endl;
   cout << "Condition No. of M1 = " << m1.Cond() << endl;
   cout << "Norm of M1 = " << m1.Norm() << endl << endl;
   cout << "\nPress any key...\n";
   getch();

   m3 = (m1 * 2.0) + (m2 ^ 3) - (2 * m1 /3) * (1/m2);
   cout << "\nMatrix equation:\n" ;
   cout << "(M1 * 2) + (M2 ^ 3) - (2 * M1 /3) * (1/M2) = \n" << m3 << endl;
   cout << "\nPress any key...\n";
   getch();
   cend();
}
```

Mit diesen Kenntnissen und Hilfsmittel wird es fast zum Kinderspiel, die Simulation der Wheatstoneschen Brücke zu schreiben.

```
// WHEAT.CPP

#include <champ.h>
#include "wheat.rh"
#include "cpbitmap.h"
#include "matrix.h"

typedef matrix<double> Matrix;

#define WINWIDTH 700
#define WINHEIGHT 600
#define XOFFSET 10
#define YOFFSET 20

double R1;
double R2 = 20; // Ohm
double R3 = 30;
double R4 = 40;
double R5 = 50;

double U0 = 10;  // Volt


void doOk ( CPDialog & dlg );
bool validate ( char * valueStr, double & value );
double calculate( double R1 );

CPWindow wnd( "Wheatstonesche Brücke", CPSize( WINWIDTH, WINHEIGHT ) );

void gmain ()
{
   CPBitmap m;
   m.loadBitmap( "Wheat" );
   m._put( wnd, XOFFSET, YOFFSET );

   CPModelessDialog dlg( "UserDialog" );
   dlg.showModeless( wnd, CP::upperRight );
   wnd._textFont( "Times New Roman", -20 );
   CPPushButton okButton( dlg, IDOK, doOk );

   while ( true )
      CP::yield();
}

void doOk ( CPDialog & dlg )
{
   int xR1 = 95;
   int yR1 = 60;
   char valueStr[20];
   double value;

   CPEdit input( dlg, IDC_INPUT );
```

```
   CPEdit output( dlg, IDC_OUTPUT );
   input.text() >> valueStr;
   if ( validate( valueStr, value ) )
   {
      wnd._text( xR1 + XOFFSET, yR1 + YOFFSET ) << value;
      output.text() << calculate( value ) << ends;
   }
   else
   {
      CP::msgBox( "Error" ) << "Illegal entry";
   }
}


bool validate( char * valueStr, double & value )
{
   istrstream is( valueStr );
   if ( is.eof() )
       return false; // Empty entry
   else
   {
      is >> value;    // Try to convert
      if ( is.fail() )         // Can't convert
         return false;
      else
      {
         // The following line will delete everything after the first space
         // It is unnecessary because is will be destroyed anywhy
         //         is.clear();
         return true;
      }
   }
}

double calculate ( double R1 )
{
   Matrix A( 6, 6 );
   Matrix xRow( 6, 1 );
   Matrix bRow( 6, 1 );

   A( 0, 0 ) =  0; A( 0, 1 ) = -1; A( 0, 2 ) =  0;
   A( 0, 3 ) =  1; A( 0, 4 ) =  0; A( 0, 5 ) = -1;

   A( 1, 0 ) = -1; A( 1, 1 ) =  1; A( 1, 2 ) =  1;
   A( 1, 3 ) =  0; A( 1, 4 ) =  0; A( 1, 5 ) =  0;

   A( 2, 0 ) =  1; A( 2, 1 ) =  0; A( 2, 2 ) =  0;
   A( 2, 3 ) = -1; A( 2, 4 ) = -1; A( 2, 5 ) =  0;

   A( 3, 0 ) =  0; A( 3, 1 ) =-R1; A( 3, 2 ) = R2;
   A( 3, 3 ) =  0; A( 3, 4 ) =  0; A( 3, 5 ) = R5;

   A( 4, 0 ) =  0; A( 4, 1 ) =  0; A( 4, 2 ) =  0;
   A( 4, 3 ) = R3; A( 4, 4 ) =-R4; A( 4, 5 ) = R5;

   A( 5, 0 ) =  0; A( 5, 1 ) =-R1; A( 5, 2 ) =  0;
   A( 5, 3 ) =-R3; A( 5, 4 ) =  0; A( 5, 5 ) =  0;

   bRow( 0, 0 ) = 0; bRow( 1, 0 ) = 0; bRow( 2, 0 ) =   0;
   bRow( 3, 0 ) = 0; bRow( 4, 0 ) = 0; bRow( 5, 0 ) = -U0;

   xRow = !A * bRow;
   return R5 * xRow( 5, 0 );
}
```

Der Artikel und der dazugehörige Source-Code ist erhältlich auf www.clab.unibe.ch/champ.


Wheatstonesche Brücke
R1 =
10
Ohm
U5 =
-0.588235
Volt
Go
R1 = 10 Ω
R3 = 30Ω
R5 = 50Ω
U5
R2 = 20Ω
R4 = 40Ω
+
-
U = 10 V