```
/*
 *  main.c
 *  Assignment One
 *
 *  Created by Derek Williams on 10-01-30.
 *  Copyright 2010 __MyCompanyName__. All rights reserved.
 *
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <OpenGL/OpenGL.h>
#include <GLUT/GLUT.h>

#include "main.h"

int activeScheme = DEFAULT_SCHEME;

int xbounds[2]      = {BOUNDS_LEFT, BOUNDS_RIGHT};
double tbounds[2]   = {TIME_START, TIME_END};

int lastFrameTime = 0;
int blowup = 0;

double dx = 0.1;
double dt = 0.0;

int Xm = 0;
int Tn = 0;

double lambda = 0.8;

float time = 0;

double **data = 0;

typedef double scheme[2][3];

scheme schemes[NUM_SCHEMES];

char *schemeNames[NUM_SCHEMES] = {
    FORWARD_SPACE_NAME,
    BACKWARD_SPACE_NAME,
    CENTRAL_SPACE_NAME,
    LAX_FREDRICHS_NAME,
    LEAPFROG_NAME,
    EQUILIBRIUM_NAME
};

void initializeSchemes(void)
{
    double lambda2 = lambda*lambda;
```

```
    schemes[FORWARD_SPACE][1][1] = 1 + lambda;
    schemes[FORWARD_SPACE][1][2] = -lambda;

    schemes[BACKWARD_SPACE][1][0] = lambda;
    schemes[BACKWARD_SPACE][1][1] = 1 - lambda;

    schemes[CENTRAL_SPACE][1][0] = 0.5 * lambda;
    schemes[CENTRAL_SPACE][1][1] = 1;
    schemes[CENTRAL_SPACE][1][2] = -0.5 * lambda;

    schemes[LAX_FREDRICHS][0][1] = 1;
    schemes[LAX_FREDRICHS][1][0] = lambda;
    schemes[LAX_FREDRICHS][1][2] = -lambda;

    schemes[LEAPFROG][1][0] = 0.5 * (1 + lambda);
    schemes[LEAPFROG][1][2] = 0.5 * (1 - lambda);

    schemes[EQUILIBRIUM][1][0] = 0.5 * (lambda + lambda2);
    schemes[EQUILIBRIUM][1][1] = 1 - lambda2;
    schemes[EQUILIBRIUM][1][2] = 0.5 * (-lambda + lambda2);
}

int initialize (void)
{
    int i;

    time = 0;
    blowup = 0;

    dt = dx/lambda;

    Xm = (xbounds[1] - xbounds[0]) / dx;
    Tn = (tbounds[1] - tbounds[0]) / dt;

    if (data)
        free(data);

    data = malloc(Tn * sizeof(double *));
    if (data == NULL)
        perror("Error allocating memory.\n");

    for (i = 0; i < Tn; i++) {
        data[i] = malloc(Xm * sizeof (double));
        if (data[i] == NULL)
            perror("Error allocating memory.\n");
    }

    initializeSchemes();
}

double initialValue (double t, double x)
{
```

```
    if ( fabs(x) <= 0.5)
        return pow(cos(M_PI * x), 2);
    return 0.0;
}

double applyScheme(int scheme, int t, int x)
{
    int n, m;
    double coef;
    double result = 0.0;

    for (n = -2; n <= -1; n++) {
        for (m = -1; m <= 1; m++) {
            if (coef = schemes[scheme][n+2][m+1])
                result += coef * data[t+n][x+m];
        }
    }

    if (abs(result) > 5 && !blowup)
        blowup = t;

    return result;
}

void finiteDifference (void)
{
    int x, t, m, n, l;

    /* apply initial condition */
    for (x = 0; x < Xm; x++)
        data[0][x] = initialValue(0, xbounds[0] + x*dx);

    l = 1;
    if (activeScheme == LAX_FREDRICHS)
    {
        l = 2;
        data[1][0] = 0.0;               // boundary condition

        for (x = 1; x < Xm-1; x++)
            data[1][x] = applyScheme(CENTRAL_SPACE, 1, x);
        data[1][x] = data[1][x-1];
    }

    for (t = l; t < Tn; t++) {
        data[t][0] = 0.0;               // boundary condition

        for (x = 1; x < Xm-1; x++)
            data[t][x] = applyScheme(activeScheme, t, x);

        if (activeScheme == BACKWARD_SPACE)
            data[t][x] = applyScheme(activeScheme, t, x);
        else
            data[t][x] = data[t][x-1];  // boundary condition
```

```
    }
}

void display(void)
{
    int x, t;
    char c[100], *pc;
    int width, wwidth;
    wwidth = (xbounds[1] - xbounds[0]);

    if (lastFrameTime == 0)
    {
        lastFrameTime = glutGet(GLUT_ELAPSED_TIME);
    }

    int now = glutGet(GLUT_ELAPSED_TIME);
    int elapsedMilliseconds = now - lastFrameTime;
    float elapsedTime = elapsedMilliseconds / 1000.0f/dt;
    lastFrameTime = now;

    time += elapsedTime;
    t = time;

    if (t >= Tn)
    {
        t = Tn - 1;
        time = tbounds[1]/dt;
    }

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glColor3f(0.2f, 0.2f, 0.2f);
    glBegin(GL_LINE_STRIP);
    for (x = 0; x < Xm; x++) {
        glVertex2f(xbounds[0] + x*dx, data[0][x]);
    }
    glEnd();

    glColor3f(1.0f, 1.0f, 1.0f);
    glBegin(GL_LINE_STRIP);
    for (x = 0; x < Xm; x++) {
        glVertex2f(xbounds[0] + x*dx, data[t % Tn][x]);
    }
    glEnd();

    sprintf(c, "%s (Time %.2f)", schemeNames[activeScheme], tbounds[0] +
        time*dt);
    pc = c;
    glColor3f(0.0f, 1.0f, 0.0f);
    width = glutBitmapLength(GLUT_BITMAP_HELVETICA_18, (unsigned char *)pc)
        ;
    glRasterPos2f(xbounds[0] + 0.5*wwidth*(1 - (float)width/glutGet
        (GLUT_WINDOW_WIDTH)), 1.5);
```

```
        pc = c;
        while (*pc != '\0')
            glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *pc++);

        if (blowup)
        {
            sprintf(c, "Blow-up (Time %.2f)", tbounds[0] + blowup*dt);
            pc = c;
            glColor3f(1.0f, 0.0f, 0.0f);
            width = glutBitmapLength(GLUT_BITMAP_HELVETICA_18, (unsigned char *
                )pc);
            glRasterPos2f(xbounds[0] + 0.5*wwidth*(1 - (float)width/glutGet
                (GLUT_WINDOW_WIDTH)), 1.3);
            pc = c;
            while (*pc != '\0')
                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, *pc++);
        }

        glutSwapBuffers();

    }

    void reshape(int width, int height)
    {
        glViewport(0, 0, width, height);

        glEnable(GL_LINE_SMOOTH);

        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(xbounds[0], xbounds[1], -1, +2);
        glMatrixMode(GL_MODELVIEW);
    }

    void idle(void)
    {
        glutPostRedisplay();
    }

    void Menu (int value) {
        int dirty = 0;

        switch (value) {
            case FORWARD_SPACE:
            case BACKWARD_SPACE:
            case CENTRAL_SPACE:
            case LAX_FREDRICHS:
            case LEAPFROG:
            case EQUILIBRIUM:
                dirty = 1; //(activeScheme != value);
                activeScheme = value;
                break;
```

```
        case H_10:
        case H_20:
        case H_40:
            dirty = 1; //(dx != 1/(double)value);
            dx = 1/(double)value;
            break;

        case LAMBDA_8:
            dirty = 1; //(lambda != 0.8);
            lambda = 0.8;
            break;
        case LAMBDA_16:
            dirty = 1; //(lambda != 1.6);
            lambda = 1.6;
            break;

        default:
            break;
    }

    if (dirty) {
        initialize();
        finiteDifference();
    }
}

void buildMenu (void)
{
    glutCreateMenu((Menu));
    glutAddMenuEntry(FORWARD_SPACE_NAME, FORWARD_SPACE);
    glutAddMenuEntry(BACKWARD_SPACE_NAME, BACKWARD_SPACE);
    glutAddMenuEntry(CENTRAL_SPACE_NAME, CENTRAL_SPACE);
    glutAddMenuEntry(LAX_FREDRICHS_NAME, LAX_FREDRICHS);
    glutAddMenuEntry(LEAPFROG_NAME, LEAPFROG);
    glutAddMenuEntry(EQUILIBRIUM_NAME, EQUILIBRIUM);
    glutAddMenuEntry(SEPERATOR, -1);
    glutAddMenuEntry(H_10_NAME, H_10);
    glutAddMenuEntry(H_20_NAME, H_20);
    glutAddMenuEntry(H_40_NAME, H_40);
    glutAddMenuEntry(SEPERATOR, -1);
    glutAddMenuEntry(LAMBDA_8_NAME, LAMBDA_8);
    glutAddMenuEntry(LAMBDA_16_NAME, LAMBDA_16);
    glutAttachMenu(GLUT_LEFT_BUTTON);

}

int main(int argc, char** argv)
{
    int t, x;

    initialize();
    finiteDifference();
```

```
    glutInit(&argc, argv);

    glutInitDisplayMode(GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH);
    glutInitWindowSize(640, 480);

    glutCreateWindow("Assignment One");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);

    buildMenu();

    glutMainLoop();

    return EXIT_SUCCESS;
}
```