Homework 7
CS 101—Spring 1998

Assigned in laboratory 11, due at start of laboratory 13.

Perform the following activities in groups of size two or three. Although you are allowed to talk to people outside your group regarding assignment requirements and debugging, the work of each group must be its own—code must not be shared among groups.

Objective

Use two-dimensional arrays to model the physical world in a engineering problem. Develop skills designing and implementing objects. Use the display facilities of the PC and Windows to illustrate the operation of the algorithm. This is called *scientific visualization*.

Problem Description

A square pipe with a square hole is immersed halfway in an ice-water bath. A liquid of known temperature, `InnerTemp`, is flowing through the pipe. The temperature of the interior surface of the pipe (the hole) is equal to `InnerTemp`. The temperature of the bottom half of the exterior surface is kept at 0° centigrade by the ice-water bath. The temperature of the top exterior surface is maintained at a known temperature, `TopTemp`. The temperature of the top half of the exterior sides of the pipe increases linearly from 0° Centigrade at the edge of the water level to `TopTemp` at the top corners. This is shown schematically in Figure 1. The problem is to compute the steady

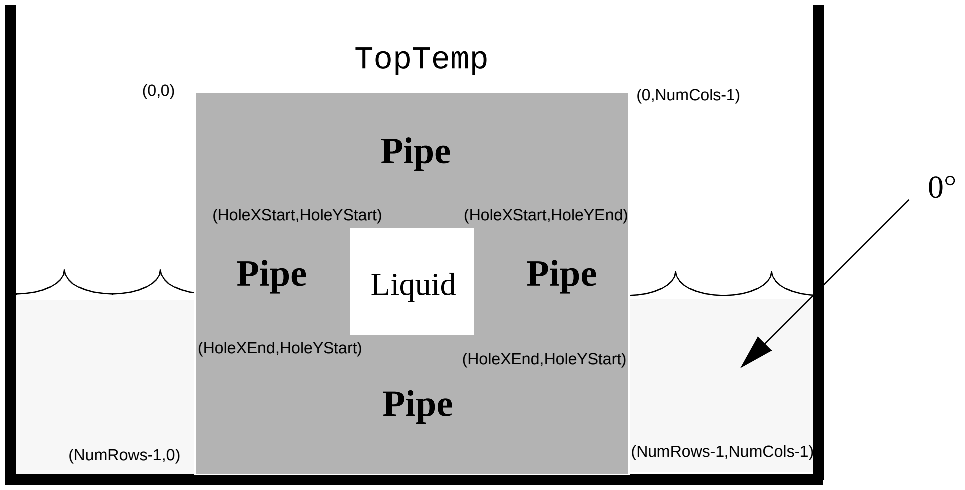

Figure 1.

state temperature distribution inside the walls of the pipe. (We use a square pipe to simplify the calculations.)

Algorithm

The temperature at point (x,y) in the pipe wall, $U(x, y)$, satisfies LaPlace’s equation:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

One way of solving this equation begins by dividing up the region into squares. Consider the variation in U along line A in Figure 2.

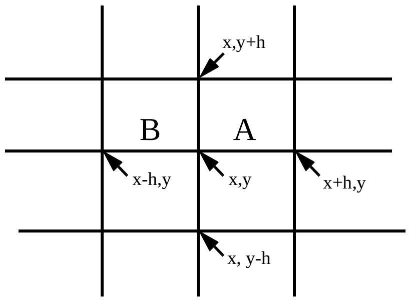

Figure 2.

We see that:

$$\left(\frac{\partial U}{\partial x}\right)_A \approx \frac{U(x+h, y) - U(x, y)}{h} = u_x$$

Similarly, along line B:

$$\left(\frac{\partial U}{\partial x}\right)_B \approx \frac{U(x, y) - U(x-h, y)}{h} = \bar{u}_x$$

Thus,

$$\left(\frac{\partial^2 U}{\partial x^2}\right)_{x, y} = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x}\right)_{x, y} \approx \frac{u_x - \bar{u}_x}{h}$$

Simalarly, along the y-axis,

$$\left(\frac{\partial^2 U}{\partial y^2}\right)_{x,y} \approx \frac{u_y - \bar{u}_y}{h}$$

where $u_y = \frac{U(x, y+h) - U(x, y)}{h}$ and $\bar{u}_y = \frac{U(x, y) - U(x, y-h)}{h}$

Substituting these into LaPlace’s equation gives:

$$U(x, y) \approx \frac{1}{4}\{U(x+h, y) + U(x-h, y) + U(x, y+h) + U(x, y-h)\}$$

which says that the approximate solution at point (x,y) is the average of the values at its four neighbors.

To solve this problem, you will develop one class (`Pipe`) and use another class that we provide (`DisplayPipe`). The class `Pipe` models the square pipe. The class `DisplayPipe` models a graphical display to observe the heat flow in the pipe. The class `Pipe` should contain an array to hold the temperatures in the walls of the pipe and the fluid flowing through the square hole. This array should be called `PipeTemperature`. It should hold double precision floating-point values. The maximum size of a pipe can be 100 by 100.

Each element of `PipeTemperature` that corresponds to a point on an interior or exterior surface of the pipe is initialized to the appropriate temperature, `TopTemp`, `InnerTemp`, 0, etc. All other elements are initialized to 0. Row 0 is the top surface of the pipe. `HoleXStart` is the top surface of the hole through the pipe. Row `HoleXEnd` is the bottom surface of the hole. Row `NumRows-1` is the bottom surface of the pipe. Column 0 is the left surface of the pipe. Column `HoleYStart` is the left surface of the hole. Column `HoleYEnd` is the right surface of the hole. Column `NumCols-1` is the right surface of the pipe. See Figure 1.

Now a temperature must be computed for each interior point of the pipe wall. This temperature is the average of the temperatures of the four surrounding points. Computing the temperature for each point in the grid constitutes one *iteration* of the algorithm. Each time a new temperature is computed for a point, the absolute difference between this new temperature and the old temperature at that point is also computed. This value is called the *residual* at that point. This process is repeated until the largest residual found anywhere in a complete iteration is less than a convergence factor, `epsilon`. At this point the process terminates.

Implementation

Your solution should consist of a module named `pipe.cpp` (and `pipe.h`). This module will contain the implementation and interface to the `Pipe` class. The constructor for `Pipe` is responsible for initializing the temperatures in the pipe. Additionally, the class `Pipe` should have a member function for computing the new set of temperatures in the pipe. After each iteration, the temperature of the pipe will be displayed so that we can see how the temperatures are converging to their final values. The temperature information will be color coded so that each grid element will have a color indicating its temperature. We are providing the class `DisplayPipe` for you to use.

To illustrate the mimimum functionality that your implementation should provide, we are providing a test harness module called `simulator.cpp`. The code for the test harness is shown below.

```
#include <assert.h>
#include <fstream.h>
#include "ezwin.h"
#include "display.h"
#include "pipe.h"

const float WindowWidth  = 15.0;
const float WindowHeight = 15.0;
const float WindowX = 1.0;
const float WindowY = 1.0;

// Create the pipe. The parameters are as follows (in order).
// The pipe is a 40 x 40 array (rows x cols).  The hole top is at row 16,
// the hole bottom is row 24. The left surface of the hole is at column 16
// and the right surface is at row 24.
// The temperature of the fluid (InnerTemp) is 200 and the
// temperature of the top surface (TopTemp) is 349.
Pipe P(40, 40, 16, 24, 16, 24, 200, 349);


// Set Epsilon and initialize Residual
const double Epsilon  = 0.5;
double Residual = 2.0;

int ApiMain() {

   // Instantiate a display for pipe P
   PipeDisplay Display("Pipe Cross-Section", P, WindowWidth, WindowHeight,
    Position(WindowX, WindowY));
   // Open a output stream to write the starting and ending configurations
   ofstream DataOut("pipe.dat");
   DataOut << "Initial Configuration Data" << endl;
   DataOut << P;

   // Begin the computation
   while (Residual > Epsilon) {
      Residual = P.Iterate();
      Display.PlotPipe(P);
   }

   // Tell the user the program is done and write the final data to the file
   DataOut << "Computation complete" << endl;
   DataOut << P;

   cout << "Computation Complete" << endl;
   cout << "Enter a character to remove the display " << endl;
   char x;
   cin >> x;
   return 0;
}

// User is shutting down the program
int ApiEnd() {
   return 0;
}
```

You should use this program to test your implementation of pipe.cpp. You may not make any changes to this program. That is, your implementation of the class pipe must provide the services this program needs.

To create an executable, you will need to create a project that includes the following files and libraries: `sim.cpp`, `pipe.cpp`, `display.cpp`, and `ezwin.lib.`

Notes

So you can see how the program is supposed to work, we have provided an executable called `demo.exe`.

The code we are providing and the demo can be downloaded from the CS101 webpage. Download file `hw7.exe`. There is a handout on the class website for setting up a project file.

Submit your code pipe.cpp electronically to your TA in advance of the lab in which it is due. Turn in a paper copy of your code in lab.

Homework 7
CS 101—Spring 1998

Assigned in laboratory 11, due at start of laboratory 13.

Perform the following activities in groups of size two or three. Although you are allowed to talk to people outside your group regarding assignment requirements and debugging, the work of each group must be its own—code must not be shared among groups.

Objective

Use two-dimensional arrays to model the physical world in a engineering problem. Develop skills designing and implementing objects. Use the display facilities of the PC and Windows to illustrate the operation of the algorithm. This is called *scientific visualization*.

Problem Description

A square pipe with a square hole is immersed halfway in an ice-water bath. A liquid of known temperature, `InnerTemp`, is flowing through the pipe. The temperature of the interior surface of the pipe (the hole) is equal to `InnerTemp`. The temperature of the bottom half of the exterior surface is kept at 0° centigrade by the ice-water bath. The temperature of the top exterior surface is maintained at a known temperature, `TopTemp`. The temperature of the top half of the exterior sides of the pipe increases linearly from 0° Centigrade at the edge of the water level to `TopTemp` at the top corners. This is shown schematically in Figure 1. The problem is to compute the steady

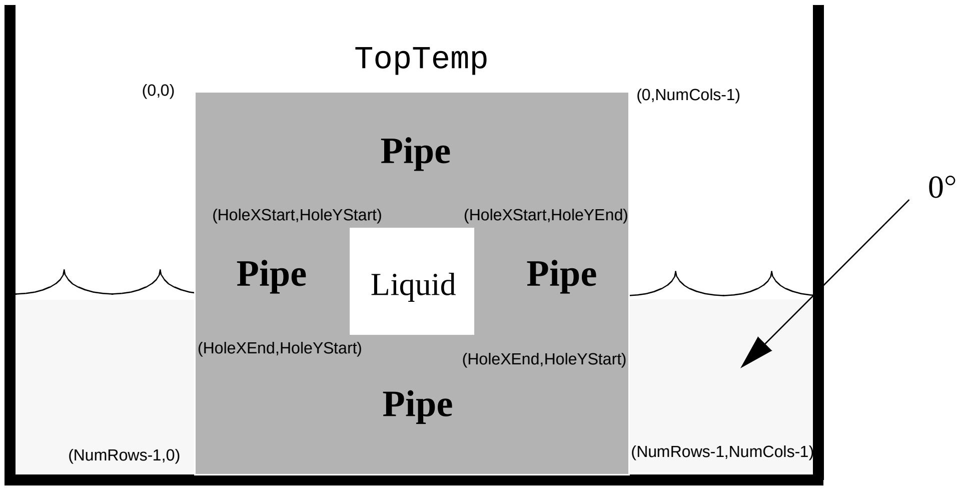

Figure 1.

state temperature distribution inside the walls of the pipe. (We use a square pipe to simplify the calculations.)

Algorithm

The temperature at point (x,y) in the pipe wall, $U(x, y)$, satisfies LaPlace's equation:

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = 0$$

One way of solving this equation begins by dividing up the region into squares. Consider the variation in U along line A in Figure 2.

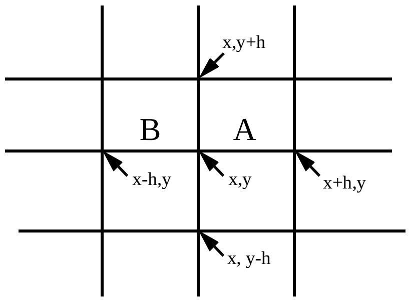

Figure 2.

We see that:

$$\left(\frac{\partial U}{\partial x}\right)_A \approx \frac{U(x+h, y) - U(x, y)}{h} = u_x$$

Similarly, along line B:

$$\left(\frac{\partial U}{\partial x}\right)_B \approx \frac{U(x, y) - U(x-h, y)}{h} = \bar{u}_x$$

Thus,

$$\left(\frac{\partial^2 U}{\partial x^2}\right)_{x, y} = \frac{\partial}{\partial x}\left(\frac{\partial U}{\partial x}\right)_{x, y} \approx \frac{u_x - \bar{u}_x}{h}$$

Simalarly, along the y-axis,

$$\left(\frac{\partial^2 U}{\partial y^2}\right)_{x,\,y} \approx \frac{u_y - \bar{u}_y}{h}$$

where $u_y = \frac{U(x, y+h) - U(x, y)}{h}$ and $\bar{u}_y = \frac{U(x, y) - U(x, y-h)}{h}$

Substituting these into LaPlace’s equation gives:

$$U(x, y) \approx \frac{1}{4}\{U(x+h, y) + U(x-h, y) + U(x, y+h) + U(x, y-h)\}$$

which says that the approximate solution at point (x,y) is the average of the values at its four neighbors.

To solve this problem, you will develop one class (`Pipe`) and use another class that we provide (`DisplayPipe`). The class `Pipe` models the square pipe. The class `DisplayPipe` models a graphical display to observe the heat flow in the pipe. The class `Pipe` should contain an array to hold the temperatures in the walls of the pipe and the fluid flowing through the square hole. This array should be called `PipeTemperature`. It should hold double precision floating-point values. The maximum size of a pipe can be 100 by 100.

Each element of `PipeTemperature` that corresponds to a point on an interior or exterior surface of the pipe is initialized to the appropriate temperature, `TopTemp`, `InnerTemp`, 0, etc. All other elements are initialized to 0. Row 0 is the top surface of the pipe. `HoleXStart` is the top surface of the hole through the pipe. Row `HoleXEnd` is the bottom surface of the hole. Row `NumRows-1` is the bottom surface of the pipe. Column 0 is the left surface of the pipe. Column `HoleYStart` is the left surface of the hole. Column `HoleYEnd` is the right surface of the hole. Column `Num-Cols-1` is the right surface of the pipe. See Figure 1.

Now a temperature must be computed for each interior point of the pipe wall. This temperature is the average of the temperatures of the four surrounding points. Computing the temperature for each point in the grid constitutes one *iteration* of the algorithm. Each time a new temperature is computed for a point, the absolute difference between this new temperature and the old temperature at that point is also computed. This value is called the *residual* at that point. This process is repeated until the largest residual found anywhere in a complete iteration is less than a convergence factor, `epsilon`. At this point the process terminates.

Implementation

Your solution should consist of a module named `pipe.cpp` (and `pipe.h`). This module will contain the implementation and interface to the `Pipe` class. The constructor for `Pipe` is responsible for initializing the temperatures in the pipe. Additionally, the class `Pipe` should have a member function for computing the new set of temperatures in the pipe. After each iteration, the temperature of the pipe will be displayed so that we can see how the temperatures are converging to their final values. The temperature information will be color coded so that each grid element will have a color indicating its temperature. We are providing the class `DisplayPipe` for you to use.

To illustrate the mimimum functionality that your implementation should provide, we are providing a test harness module called `simulator.cpp`. The code for the test harness is shown below.

```
#include <assert.h>
#include <fstream.h>
#include "ezwin.h"
#include "display.h"
#include "pipe.h"

const float WindowWidth  = 15.0;
const float WindowHeight = 15.0;
const float WindowX = 1.0;
const float WindowY = 1.0;

// Create the pipe. The parameters are as follows (in order).
// The pipe is a 40 x 40 array (rows x cols).  The hole top is at row 16,
// the hole bottom is row 24. The left surface of the hole is at column 16
// and the right surface is at row 24.
// The temperature of the fluid (InnerTemp) is 200 and the
// temperature of the top surface (TopTemp) is 349.
Pipe P(40, 40, 16, 24, 16, 24, 200, 349);


// Set Epsilon and initialize Residual
const double Epsilon  = 0.5;
double Residual = 2.0;

int ApiMain() {

   // Instantiate a display for pipe P
   PipeDisplay Display("Pipe Cross-Section", P, WindowWidth, WindowHeight,
    Position(WindowX, WindowY));
   // Open a output stream to write the starting and ending configurations
   ofstream DataOut("pipe.dat");
   DataOut << "Initial Configuration Data" << endl;
   DataOut << P;

   // Begin the computation
   while (Residual > Epsilon) {
      Residual = P.Iterate();
      Display.PlotPipe(P);
   }

   // Tell the user the program is done and write the final data to the file
   DataOut << "Computation complete" << endl;
   DataOut << P;

   cout << "Computation Complete" << endl;
   cout << "Enter a character to remove the display " << endl;
   char x;
   cin >> x;
   return 0;
}

// User is shutting down the program
int ApiEnd() {
   return 0;
}
```

You should use this program to test your implementation of pipe.cpp. You may not make any changes to this program. That is, your implementation of the class pipe must provide the services this program needs.

To create an executable, you will need to create a project that includes the following files and libraries: `sim.cpp`, `pipe.cpp`, `display.cpp`, and `ezwin.lib.`

Notes

So you can see how the program is supposed to work, we have provided an executable called `demo.exe`.

The code we are providing and the demo can be downloaded from the CS101 webpage. Download file `hw7.exe`. There is a handout on the class website for setting up a project file.

Submit your code pipe.cpp electronically to your TA in advance of the lab in which it is due. Turn in a paper copy of your code in lab.