

Ph 265. Arrays, file I/O

1. Arrays

Array can be thought of as a collection of the numbered data of the same type. The individual piece of data is called element, while the "order number" of that piece is called index.

For example, an array can represent a series of grades for the courses (the index in this case will be the course number, and the grade itself will be a value of the element). Another example – is annual temperature "graph" for a particular city (the day of a year serves as an index).

Functions of a real variable can be also stored in an array, but as we already know, only with finite step. To store the function of a real variable in an array we will need two arrays. The first one will store the value of the "independent variable" as a function of a point number. The second array will store the value of a function at that point number.

However, having two separate arrays for this particular point may be not really convenient, since the two arrays may seem like two independent instances. Java and other programming languages give us an instrument of multidimensional array to deal with this trouble. The multidimensional array is an array, every element of which is in turn an array. Note: the dimensions of all the "1-st level elements" must be the same since the array is a structure which has elements of the same type!

2. Operating Arrays

In order to use the arrays in Java, you should first declare and create the array. As usual, by declaration we just tell the compiler that the variable will be an array variable, and by creating we mean "memory reservation" for all the array elements.

After creation we can address any array element by referring to the array (variable) name, followed by the index number in square brackets. We can also pass the arrays to the functions. And, in contrast to "conventional" (int, real, double) variables, the function will modify its array and object arguments. The reason is simple: the arrays and objects may take up a large amount of memory, and since passing the copy to the function requires creation of that copy we could run out of memory even before we start the crucial computations. To avoid this complication, Java passes the "address" of an array and of any object argument to the function.

Listing 18.1 OneDArray.java

```

1 // OneDArray.java: using arrays as vectors
2 public class OneDArray
3 { public static void main(String [] argv)           // main method
4 { double A[] = new double [10000];               // Initialize arrays
5   double B[] = new double [10000];
6   double C[] = new double [10000];
7   int j, k;
8   double startTime, endTime;
9   startTime = System.currentTimeMillis();          //Present t
10  for (int i=0; i< 10000; i++)
11  { A[i] = Math.sqrt(i);                          //Assign array elements
12    B[i] = i *i;
13    System.out.println( "i, A[i], B[i] = "
14                        + i + ", "+ A[i]+ ", " +B[i] ); }
15  System.out.println(" ");
16  for (int i=0; i< 10000; i++)
17  { j = (i + 1);
18    if ( j > 9999 ) j = j - 9999;
19    k = (i + 2);
20    if ( k > 9999 ) k = k - 9999;
21    C[i] = A[j]*B[k] - A[k]*B[j];
22    System.out.println("i, j, k, C[i] = "
23                      + i+ ", " + j+ ", " + k+ ", " + C[i]); }
24  endTime = System.currentTimeMillis();
25  System.out.println("time (s) =" + (endTime -startTime)/1000);
26 }}

```

3. File I/O; Streams

Here we assume that we are dealing with applications (not applets). Due to security reasons (applets run on end-user computer) applets are rarely allowed to communicate with file system and therefore file I/O from applets (although similar to one in standalone applications) is not considered here.

The simplest way to redirect all output of the program to the file is just using the ">" command of the operating system:

```
java YOURCLASS.CLASS>OUTPUT_FILE_NAME
```

In this case everything you would print via standard "System.out.println" would go directly to the file you have specified. Similarly, you can use the operator "<" to redirect all input from "System.in" from the default (keyboard). However, sometimes we would rather have access to the screen and keyboard in the runtime of our program...

In general, everything our Java program prints, draws, or inputs comes from (or to) some point. This flow of information is called "stream". Note that in analogy with "natural" streams, Java streams have definite direction – the information either flows out of the program (output stream), or into the program (input stream). In order to write to the file

or read from it, we have to first create the stream, and associate this stream with the file, and then we can either write to or read from the stream (see lines 10, 29-33).

Listing 20.1 Laplace.java

```

1 //Laplace.java: Solves Laplace eqn with finite difference method
2 Output data saved in 3D grid format of gnuplot
3 import java.io.*;
4 public class Laplace
5 {   static int Nmax = 100;           //Size of box
6     public static void main(String[] argv) throws IOException,
7         FileNotFoundException {
8         double U[][] = new double[Nmax][Nmax];
9         int i, j, iter;
10        //Save data in file Laplace.dat
11        PrintWriter w = new PrintWriter(new FileOutputStream("Laplace.
12            dat"), true);
13        for (i=0; i< Nmax; i++)           //Inititize array
14        { for (j=0; j< Nmax; j++) { U[i][j] = 0.0; } }
15        for (i = 25; i<= 75; i++)       //Boundary conditions+--+*/
16        { U[i][37] = -100.0; /*++--//parallel plates' potential
17            U[i][63] = 100.0; }
18        for (i = 0; i< Nmax; i++)       //box potential
19        { U[0][i] = 0.0;
20            U[Nmax-1][i] = 0.0;
21            U[i][0] = 0.0;
22            U[i][Nmax-1] = 0.0;
23        }
24        for (iter=0; iter< 1000; iter++) //iterate 1000 times
25        { for (i=1; i<=Nmax-2; i++) { //x-direction
26            for(j=1; j<= Nmax-2; j++) { //y-direction
27                U[i][j] = 0.25*(U[i+1][j] + U[i-1][j] + U[i][j+1] + U[
28                    i][j-1]);
29                if (j==37 && i>=25 && i<=75) U[i][j] = 100.0;
30                    //Fixed V on plates
31                if (j==63 && i>=25 && i<=75) U[i][j] = -100.0 ;
32            }
33        }
34        for (i=0; i<Nmax ; i=i+2) { //Write data in gnuplot 3D format
35            for (j=0; j<Nmax; j=j+2) {
36                w.println(""+U[i][j]+"");
37                w.println(""); //Blank line separates rows for gnuplot
38            }
39        }
40        System.out.println("data stored in Laplace.dat");
41    } //end main
42 } // end class

```

Some general remarks on the problem itself: we are solving Laplace equation:

$$\frac{\partial^2 V(x, y)}{\partial x^2} + \frac{\partial^2 V(x, y)}{\partial y^2} = 0$$

with some predefined boundary conditions.

To do this, we first define the coordinate mesh. Now our $U[i][j]$ gives us the value of the potential (V) at the point (x_{ij}) . In our calculations we use the following relation to find the values of the derivatives:

$$\frac{\partial^2 V(x, y)}{\partial x^2} \simeq \frac{V(x + \Delta, y) + V(x - \Delta, y) - 2V(x, y)}{\Delta^2}$$

$$\frac{\partial^2 V(x, y)}{\partial y^2} \simeq \frac{V(x, y + \Delta) + V(x, y - \Delta) - 2V(x, y)}{\Delta^2}$$

From where we derive:

$$V(x, y) \simeq \frac{1}{4} [V(x + \Delta, y) + V(x - \Delta, y) + V(x, y + \Delta) + V(x, y - \Delta)]$$

The program above implements the following algorithm to calculate the potential distribution:

```
1   set potential = 0 on surrounding around box
2   set potential = 100, -100 on upper, plates
3   repeat 1000 times
4       for Nmax x space steps
5           for Nmax y space steps
6               V = average of 4 nearest neighbors
7               keep potential on box \& plates fixed
8   print out V
```