PFI lecture ARRAY

- What is an Array?
  - An array may be thought as a new data type, it has a type and a value
  - An array has many elements, each array element may be thought as a variable by itself, which has a type and a value.
  - All array elements have the same type.
  - Array elements are stored in memory one after another consecutively. The first element is indexed at 0, the second element indexed at 1, and so on.

- Why do we need arrays in programming?
  - To have an expression that may refer to many variables.
  - To have a collection of variables, where the size can be easily specified and each variable in the collection may be easily identified.

Examples where arrays are needed

- Storing all course grades of a student in our program. For each course, we have a grade and a credit hour. Here we must have two variables for each course. The more courses we have, the more variables are needed.

- Storing all user ID and password in order to validate the user credential. Each online account has a user ID and a password. So we need two variables for that. The more users we have the more variables we must have.

- Storing and manipulating a two dimensional digital picture. Each pixel has a x and y coordinates and a color. So three variables are needed for each pixel. A picture may easily have thousands of pixels. Hence that many variables are needed.

Syntax for declaring an array

TYPE array_name[size_of_array_literal];

- Declaring an array of int with 10 elements
  - **int** hours[10];
- Declaring an array of char with 15 elements
  - **char** grade[15];
- Declaring an array of strings with 7 elements
  - **string** name[7];
- Declaring an array of double with 1000 elements
  - **double** balance[1000];

Visualization of arrays

TYPE array_name[size_of_array_literal];

- **int** hours[10];

hours[0]	hours[1]	hours[2]	hours[3]	hours[4]	hours[5]	hours[6]	hours[7]	hours[8]	hours[9]

- **char** grade[15];

grade[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	grade[14]

- **string** name[7];

name[0]	name[1]	name[2]	name[3]	name[4]	name[5]	name[6]

- **double** balance[1000];
  - Too big to diagram it! But you get the idea.

Referring to an element

- int hours[10];

hours[0]	hours[1]	hours[2]	hours[3]	hours[4]	hours[5]	hours[6]	hours[7]	hours[8]	hours[9]

- Using a literal as an index
  - hours[3]=hours[2] – hours[7];
- Using a variable as an index (n is a variable of type int)
  - houns[n]=hours[n] + hours[5];
  - We should make sure the value of n is between 0 and 9
- Using an expression as an index
  - hours[n+1]=hours[n] + hours[n-1];

Initialization in declaration

Typically used for small sized arrays

An exact number of elements in { }

- int hours[10] ={10,20,30,40,50,60,70,80,90,100};

Fewer values in { }, the rest is set to zero

- int hours[10] ={10,20,30}; //hours[3] to hours[9] is 0

More Values in { }, compiler error!

- int hours[10] ={0,1,2,3,4,5,6,7,8,9,10};

- What do you think of the following? No value at all.
- int hours[10] ={ };

Initialization using loop

```
const int SIZE =40; // with const, cannot change value
 int fibo[SIZE];
 fibo[0]=1;
 fibo[1]=1;
 for (int i=2; i < SIZE; i=i+1)
      fibo[i]=fibo[i-1]+fibo[i-2];
 for (int i=0; i < SIZE; i=i+1)
      cout << fibo[i] << " ";
```

Syntax for declaring higher dimensional arrays

Two Dimensional Array

TYPE array_name[size_of_row][size_of_col];

- Declaring an array of int with 10 rows and 5 columns
  - **int** table[10][5];

Three Dimensional Array

TYPE array_name[size_of_d1][size_of_d2] [size_of_d3];

- Declaring an array of int with 10, 5, 20 as the size of each dimension.
  - **int** cube[10][5][20];

Initialization using loop

```
const int SIZE =4; // with const, cannot change value
 int board[SIZE][SIZE];
 for (int i=0; i < SIZE; i=i+1)
     for (int j=0; j < SIZE; j=j+1)
            board[i][j] = i + j;
for (int i=0; i < SIZE; i=i+1) {
     for (int j=0; j < SIZE; j=j+1)
            cout << setw(4) << board[i][j];
     cout << endl;
 }
```