Arrays

Chapter 8

page 477

What if you have a 1000 line file?

- Read in the following file and print out a population graph as shown below. The maximum value for a population is 5000. *Each input file will have exactly five lines of data*. Use one star to represent each 1000 people.

Input file

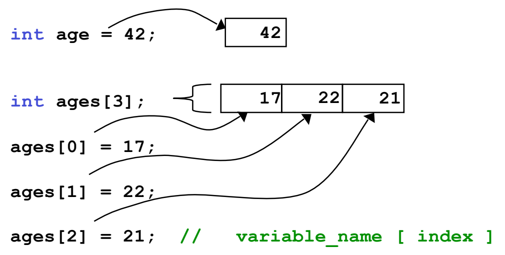
```
1970 1000
1980 1000
1990 2000
2000 4000
2010 5000
```

Output to Screen:

```
5000                         *
4000                    *    *
3000                    *    *
2000               *    *    *
1000   *    *      *    *    *
     1970 1980 1990 2000 2010
```

Arrays

- Provides an easy way to store and access multiple (related) values of the same data type



```
int age = 42;

int ages[3];

ages[0] = 17;

ages[1] = 22;

ages[2] = 21;  //   variable_name [ index ]
```

Declaring an Array

- The size of the array must be a *literal* or a `const int`.

```
int size = 99;
const int constSize = 1024;

string names[3];          // literal
double tempatures[size];      // illegal!
int tests[constSize];     // const int
```

- When the code is compiled, the exact size of the array must be known

Using arrays

- The first element in the array is the **0th** element!
- You can use a single element of an array just like any other variable
- The ***index*** is just an `int`
- Must use an index to access the array

Practice

- Write a snippet of code to print to the screen the sum and average of the values in this array:

```
int vals[4];
vals[0] = 1;
vals[1] = 2;
vals[2] = 4;
vals[3] = 8;
```

When would I use this?

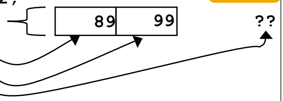
- Read in 5 test scores from the user. Calculate the average test score and print out the scores in reverse order.
- input: 100, 90, 84, 90, 89
- output:

```
Average: 90.6
89
90
84
90
100
```

You could do this with 5 integers. But what about 100 test scores? Or 1000?

Danger! Danger!

```
const int constSize = 2;
int tests[constSize];

tests[0] = 89;
tests[1] = 99;
tests[4] = 42; // what happens?
```



- C++ does *not* check to make sure the index falls within the array
  - no *bounds checking*
  - this will cause unpredictable results!
  - you may write over other, valid data
  - you may write over part of the program
    - common security problem (buffer overflow)

Initialization

- How do you set the initial values for the array elements?
- What is the equivalent of:

```
int value = 2;
int tests[2] =
string names[3] =
```

- Initialize just a few values:

```
int value[4] =
```

Implicit array sizing

- Set the size of the array by initializing it
- You *must* either specify a size or initialize the array

```
string names[] =



char letters[] =
```

Using Arrays

- Write code that will use arrays to store the names the months and the number of days in each month (assume no leap year!). Print the following to the screen:

```
January     31
February    28
March       31
April       30
May         31
June        30
July        31
August      31
September   30
October     31
November    30
December    31
```

Arrays as Function Arguments

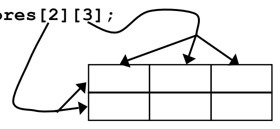
- You can pass an array as an argument to a function

```
void printIntArray(int arr[], int size);

int main()
{
 int values[] = { 5, 6, 0, 1, 2, 3, 4};
 const int SIZE = 7;

 printIntArray(values, SIZE);

 return 0;
}
```

Two dimensional arrays

- A grid of data!

```
int testScores[2][3];
```



```
testScores[0][0] = 99;
testScores[0][1] = 80;
testScores[0][2] = 88;
testScores[1][0] = 89;
testScores[1][1] = 77;
testScores[1][2] = 85;
```

Why use 2D arrays?

- Hold the scores for each student in one array.

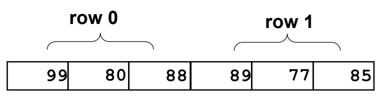
```
const int BOB = 0;
const int ALICE = 1;
const int MIDTERM1 = 0;
const int MIDTERM2 = 1;
const int FINAL = 2;
int testScores[2][3] = { {0, 0, 0},
                         {0, 0, 0} };
testScores[BOB][MIDTERM1] = 99;
testScores[ALICE][FINAL] = 85;
```

- Which values are we setting above?

A 2D array in memory

- The 2D array laid out by rows in memory

```
int testScores[2][3];
```



```
testScores[0][0] = 99;
testScores[0][1] = 80;
testScores[0][2] = 88;
testScores[1][0] = 89;
testScores[1][1] = 77;
testScores[1][2] = 85;
```

This is called row major order. Some languages use column major order.

Passing a 2D array to a function

- Must specify at least one of the sizes in the function’s parameter list so the compiler knows how to access the array

```
void printInt2DArray(int arr[][3], int size);

int main()
{
 int values[] = { {5, 6, 0},
                  {2, 2, 3} };
 printIntArray(values, 2);
 return 0;
}
```

5	6	0	2	2	3

Practice

- Write the function:

```
void printInt2DArray(int arr[][3], int size);
```

N-Dimensional Arrays

```
string fifthDimension[5];
fifthDimension[0][0][0][0][0] = “up”;
fifthDimension[0][1][0][0][0] = “up”;
fifthDimension[0][0][1][0][0] = “and”;
fifthDimension[0][0][0][1][0] = “away”;
```