PIC 10A

Lecture 20: Arrays

What are arrays?

Arrays are much like vectors in the sense that they hold data of a same type stored sequentially.

Like vectors elements of the arrays are accessed using an index.

Arrays are a more primitive C++ construct than vectors. We don't need to include any libraries to use arrays.

Arrays are more efficient and often result in faster programs.

Arrays don't have any member functions :(

How do I declare an array?

Syntax:

```
type array_name[size];
```

eg.

```
int numbers[10];
double decimals[20],
Product inventory[1000];
```

We could also declare and fill an array at the same time:

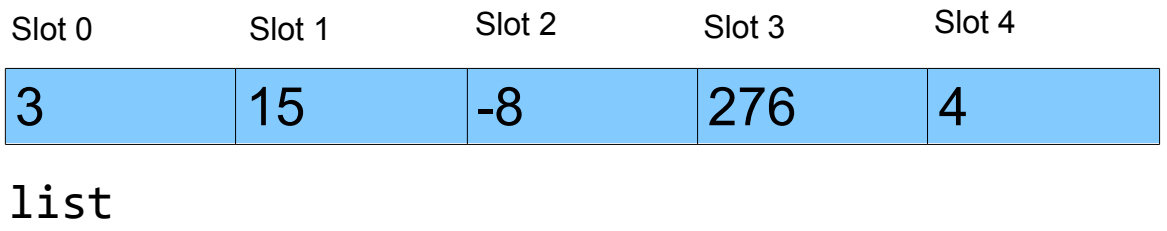
```
int list[5]={3,15,-8,276,4};

int list[]={3,15,-8,276,4};
```

Basics on arrays

We think of an array much the same way as a vector

Graphically a array might look like:



Idea is to access the data in the list array via the slot number.

The first slot is 0 and the last slot is size-1.

We use each slot of the array as its own variable.

```
cout << list[i];
```

Common mistake

Arrays run from index 0 to index size-1

Consider the following code:

Visual Studio will not give you an error if you run past the length of the array. But you'll get very strange results.

```
int values[3] = {10, 20, 30};
for (int i=0; i<=3; i++)
   cout << values[i] << " ";
```

OUTPUT
10 20 30 -858993460

Array's size is not dynamic!

Arrays size must be known at compilation time!

```
int size=5;
int numbers[size]; // Horrible, horrible error!
```

You cannot use a variable to set array size. Array size is always fixed.

```
const int size=5;
int numbers[size]; // Now it works.
```

Common mistake:

```
int size;
cin >> size;
int list[size];
```

Operations on arrays are done element-wise

say you have:

```
int a[3]={1,2,3};
int b[3]={2,3,4};
```

You can do:

```
a[0]=b[0];
```

You cannot do

```
a=b;   //WRONG
```

For same reason you can not do:

```
int ar[3];
ar={1,2,3};
```

Note: `int ar[3]={1,2,3};` is ok because we are declaring the variable.

Two things everyone must know

1. How do I fill an array?

```
int numbers[5];
 for (int i=0; i<5; ++i)
   cin >> numbers[i];
```

2. How do I output contents of the array?

```
for (int i=0; i<5;++i)
   cout << numbers[i];
```

Array size

You can declare your array to be as big as you want.

You do not need to fill the entire array.

Usually you need a variable to track how much of the array you are using!

You should make your array big enough so you will never need to go over its capacity.

You should make your array small enough so you don't waste resources.

Passing arrays to functions

Important point #1: ARRAYS ARE NOT PASSED BY VALUE! They are automatically passed by reference. You don't have to put an & in front of an array.

Important point #2: When you want to pass an array to a function you must also pass the length of the array in a separate variable. The function will not know how big your array is.

Example:

```
double mean(int ar[], int n)
{
int sum=0;
    for(int i=0; i<n; i++)
        sum += ar[i];
    return 1.0*sum/n;
}
```

Passing arrays to functions

When we call functions we do not put any types:

Example:

```
int double ave = mean(numbers,100);
```

or

```
int double ave = mean(ar, size);
```

Be aware that unlike vectors, arrays cannot be returned by functions.

Important example

We want to search through an array a of integers and find the largest value occuring in the array.

```
int max(int a[], int size)
{
   int m = a[0];
   for(int j=1; j<size; j++)
      if(m < a[j])
         m = a[j];
   return m;
}
```

It is important that we set m initially to some element in the array. By default we pick the first element. Why can't we just set m to 0 initially?

2D Arrays

By 2D array we mean an array with two indeces that is storing data of a matrix or a table.

1.2	5.6	-3.2	0.7
6.8	0	4.2	3.3
-7	2.1	3.9	2.2

Each element in this table is identified by a row number and a column number.

First row is row 0 and first column is column 0.

The cell with 4.2 is in cell identified by row 1 and column 2.

2D arrays cont.

Declaring 2D arrays is much easier than declaring 2d vectors.

To declare the 2D array from previous slide we write:

```
double table[3][4];
```

Then we can start filling this table;

```
table[0][0] = 1.2;
table[0][1] = 5.6;
table[0][2] = -3.2;
table[0][3] = 0.7; // This finishes first row

table[1][0] = 6.8;
table[1][1] = 0;

etc.
```

2D Arrays cont.

In general when we have data structures indexed by two numbers we use nested for loops to fill them:

```
int value;

for (int i=0; i < 3; i++)
{
   for (int j=0; j < 4; j++)
   {
      cout << “Enter a value: ”;
      cin >> value;
      table[i][j] = value;
   }
}
```

Passing a 2D array to a function

When you pass a 2D array to a function you have to specify the size of the 2D array.

Recall with normal arrays you do not put the size.

```
void my_function (double table [10][20], int m, int n)
{
// stuff
}
```

Actually, you are only required to give the number of the columns. But specifying the number of rows too doesn't hurt.

```
void my_function (double table [][20], int m, int n)
{
// stuff
}
```

Passing a 2D array to a function

To pass these numbers, probably best to declare the array size as global constants. At the top of your program, right under the #includes state the size of your array.

```
const int num_rows = 10;
const int num_cols = 20;

void my_function (double table [num_rows][num_cols])
{
// stuff
}
```

2D array example

```
const int POWERS_ROWS = 11;
const int POWERS_COLS = 6;


int main()
{
   double powers[POWERS_ROWS][POWERS_COLS];
   for (int i = 0; i < POWERS_ROWS; i++)
      for (int j = 0; j < POWERS_COLS; j++)
         powers[i][j] = pow(i, j);

   print_table(powers, POWERS_ROWS, POWERS_COLS);

   return 0;
}
```

2D array example

```
void print_table(const double table[][POWERS_COLS],
   int table_rows, int table_cols)
{
   const int WIDTH = 10;
   cout << fixed << setprecision(0);
   for (int i = 0; i < table_rows; i++)
   {
      for (int j = 0; j < table_cols; j++)
         cout << setw(WIDTH) << table[i][j];
      cout << "\n";
   }
}
```

2D array example: Matrix transpose

A matrix is just a table of values:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

A transpose of a matrix A written as A^T we mean a matrix which we obtain from A by changing every row into a column

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

Matrix transpose example cont.

We want to write a program that computes a matrix transpose.

NOTE: `A[i][j] = A`^{T}`[j][i]`.

We will write a function that takes as argument a 2D array that contains the data for the matrix A.

We will also supply as an argument another 2D array which by the end will store the matrix transpose of A.

Why do we need to pass this other 2D array as an argument? Couldn't we just have the function return it?

Passing a 2D array to a function

```
const int A_rows = 2; //Or whatever sizes you want.
const int A_cols = 3;

void transpose (int A[A_rows][A_cols],
   int Atrans [A_cols][A_rows])
{
      for (int i = 0; i < A_rows; i++)
         for (int j=0; j < A_cols; j++)
            Atrans [j][i] = A[i][j];
      return;
}
```

Vector or an Array?

1) Size can change while the program runs.

2) You will need a variable to keep track of how much of it is currently filled.

3) Has member functions that help in manipulating it.

4) Is automatically passed by reference.

5) It is a valid return type for functions.

6) Can be used in assignment.

7) Better suited for 2D structures.

8) Is more cpu efficient.

9) Is a primitive data type