**Lab 78**

**Two-Dimensional Arrays**

Remember that an array is a contiguous block of memory.

The array name is an alias for the address of the first byte in the contiguous set of bytes.

C++ can handle multi-dimensional arrays, of any number of rows or columns.

Remember, the entire array occupies memory.

Remember, every cell of the array is of the same type.

**Declaration Syntax:**

1-Dim array: anArray[num]

2-Dim array: anArray[num1][num2] // Most commonly represent Tables

3-Dim array: anArray[num1][num2][num3] // Represent a Cube

Two dimensional arrays are like a 2-dimensional table.

Example of a 3 by 4 array

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  |  |  |

**Array indexes**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *0* | *1* | *2* | *3* |
| *0* | [0][0] | [0][1] | [0][2] | [0][3] |
| *1* | [1][0] | [1][1] | [1][2] | [1][3] |
| *2* | [2][0] | [2][1] | [2][2] | [2][3] |

In a one dimensional array, you can directly access an element by using a single index for that element.

Example: anArray[3] would access the 4th element

In a two dimensional array, you can directly access an element by using two indexes for that element.

Example: anArray[2][3] would access the 3 row and 4 column.(bottom right corner)

**Array Initialization with loop**

for( i = 0; i < Rows; i++){

for( j = 0; j < columns; j++)

{ anArray[i][j] = 0; }

}

**Array Initialization with list**

int grades[NUM\_STUDENTS][NUM\_TESTS] = { {51, 30, 50, 21},

{91, 31, 98, 51},

{42, 21, 87, 11}

};

Program Sample:

#include <iostream.h>  
const int ROW=4;  
const int COLUMN =3;  
void main()  
{  
   int i,j;  
   int anArray[ROW][COLUMN];  
   for(i=0;i<ROWS;i++)  
   for(j=0;j<COLUMN;J++)  
   {  
   cout << "Enter value of Row "<<i+1;  
   cout<<",Column "<<j+1<<":";  
   cin>> anArray [i][j];  
   }  
   cout<<"\n\n\n";  
   cout<< " COLUMN\n";  
   cout<< " 1 2 3";

for(i=0;i<ROW;i++)  
   {  
   cout<<"\nROW "<<i+1;  
   for(j=0;j<COLUMN;J++){  
   cout<< anArray [i][j];

}   
 }

}

**Passing a 2-dimensional array in a function.**

Remember, the array is passes by reference.

returnType FunctionName( datatype anArray[ ][num2], int sizeOfFirstIndex )

Note the first index is left blank, and the second index number is specified.

Use: FunctionName( theArray , FirstIndexSize)

**OFFSET** - Remember the index is really an offset from the first address, of contiguous group of bytes.

This table shows the offset from the first byte.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | *0* | *1* | *2* | *3* |
| *0* | **0** | **1** | **2** | **3** |
| *1* | **4** | **5** | **6** | **7** |
| *2* | **8** | **9** | **10** | **11** |

**Exercise 1:**

Create a 10 by 10 integer array. Initialize the table so it contains the **Multiplication table**.

Write 3 functions:

1. Initialize the multiplication table
2. Print the complete table
3. Ask and lookup answer for a product of two numbers. Print answer.

To Test: Initialize, print whole table, lookup and print answer

**Exercise 2:**

**Tic Tack Toe game**

Create a 3 by 3 char array.

Write 3 functions

1. Initialize each element to a space.
2. Have the computer start by asking player x to choose *available* move.
   1. Mark it with an x
   2. Check for x winning –if yes, say x has won.
3. Have the computer start by asking player o to choose available move.
   1. Mark it with a o
   2. Check for o winning –if yes, say o has won.
4. If all nine cells are full,
   1. say “Cats game, no winner”.
5. Loop to play again, reinitialize board to spaces
   1. Y/N answer. Yes start play, No end game.

**STRUCTURES**

In C++ you may want to combine or group several variables together.

You can place them in a group or bucket if will called a structure.

Structure Syntax.

struct structName {

type var1;

type var2;

type var 3;

or more

}

This is unusual because what you are doing is actually creating/defining a new type. In the past you have used types in Declarations. Now you are creating types, then you write a declaration

Example:

**1st** Create a Structure (struct) that contain several data types.

(**Define** a new type)  
  
struct aStruct {  
 int intVar; // single integer variable  
 char charVar; // single char varaible  
 double doubleVar;// single double variable  
}  
  
**2nd** **Declare** an instance of the new type  
Lets declare an instance of the new type.

aStruct S;

**3rd** Use the array.

(**Use** the structure)

Since there are many variables inside a structure,

you have to use the DOT notation to select which member to use.

S.intVar = 3;

S.charVar = ‘A’;

S.doubleVar = 2.17;

**Arrays of structures**

Declaration:  
aStruct structArray[10];

Use:  
structArray[2] = { 5, 'A', 3.5 } ; // fill by list  
structArray[4] = [ 2, 'x', 12.2 } ; // file by list  
  
structArray[2].intVar = 2 ; // fill by variable  
structArray[2].charVar = 'A' ;// fill by variable  
structArray[4].doubleVar = 3.9; // fill by varible  
  
**Exercise 3 -**

**Card Deck.**

Create a structure for a playing card with face and suit variable.

Create a 4 by 13 array – of the appropriate ‘structure’. 4 –suites, 13 – cards each suite

Write a function initialize each card in the array, it will write the suite and card into each array element.

Write a function that will print out the deck.

Write a function that will randomize the deck.

To test: initialize, print, randomize, and print again

(Pay attention in class for help with exercises)