DATA STRUCTURES

2D Array and Pointers

CS Dept. – Qassim University

OUTLINE

1. Implementation of matrix operations
2. Example of operations on matrix class
3. Pointers Revision
4. Exercises on Pointers.

COMMON MATRIX OPERATIONS

- Common matrix operations are addition, multiplication and transposition.

IMPLEMENTATION OF MATRIX OPERATIONS

```
#include <iostream>
using namespace std;
const int MAX = 3;
class matrix
    {
       private:
            int mat[MAX][MAX];
       public:
            matrix(); void create();
            void display();
            void matadd (matrix &m1, matrix &m2);
            void matmul (matrix &m1, matrix &m2);
            void transpose (matrix &m1);
    };
```

INITIALIZING MATRIX WITH 0

```
matrix:: matrix()
{
	for (int i = 0; i < MAX; i++)
	{
		for (int j = 0; j < MAX; j++)
			mat[i][j] = 0;
	}
}
```

CREATING A MATRIX

```
void matrix:: create()
{
      int n;
      for (int i = 0; i < MAX; i++)
      {
            for (int j = 0; j < MAX; j++)
            {
                  cout << endl << "Enter the element:";
                  cin >> n;
                  mat[i][j] = n;
            }
      }
}
```

The **create()** function creates a 2-D array containing 3 rows and 3 columns

DISPLAYING THE CONTENTS OF MATRIX

```
void matrix:: display()
{
      for (int i = 0; i < MAX; i++)
      {
            for (int j = 0; j < MAX; j++)
                  cout << mat[i][j] << “”;
            cout << endl;
      }
}
```

The **display()** function displays the elements of the matrix.

ADDING TWO MATRICES

```
void matrix:: matadd (matrix &m1, matrix &m2)
{
      for (int i = 0; i < MAX; i++)
      {
            for (int j = 0; j < MAX; j++)
                  mat[i][j] = m1.mat[i][j] + m2.mat[i][j];
      }
}
```

The function **matadd()** adds the elements of two matrices **m1** and **m2**. The result is stored in matrix **mat** of the object that has called the **matadd()** function.

MULTIPLYING TWO MATRICES

```
void matrix:: matmul (matrix &m1, matrix &m2)
{
    for (int k = 0; k < MAX; k++)
    {
        for (int i = 0; i < MAX; i++)
        {
            for (int j = 0; j < MAX; j++)
                mat[k][i] += m1.mat[k][j] * m2.mat[j][i];
        }
    }
}
```

The function **matmul()** multiplies the elements of matrix **mat1.mat** with the elements of matrix **mat2.mat** and stores the result into the matrix **mat** of the object **mat4** that called the function.

OBTAINING TRANSPOSE OF MATRIX

```
void matrix:: transpose (matrix &m1)
{
    for (int i = 0; i < MAX; i++)
    {
        for (int j = 0; j < MAX; j++)
            mat[i][j] = m1.mat[j][i];
    }
}
```

The function transposes a matrix.
The transpose of a matrix is obtained by interchanging the rows with corresponding columns of a given matrix.
The transposed matrix is stored in matrix **mat** of the object

EXAMPLES (TRY THIS CODE)

```
matrix mat1;
cout << endl << “Enter elements for first matrix:” << endl;
mat1.create();
matrix mat2;
cout << endl << “Enter elements for second matrix:” << endl;
mat2.create();
cout << endl << “First matrix:” << endl;
mat1.display();
cout << endl << “Second matrix:” << endl;
mat2.display();
matrix mat3;
```

EXAMPLES (TRY THIS CODE)

```
mat3.matadd (mat1, mat2);
cout << endl << “After Addition:” << endl;
mat3.display();
matrix mat4;
mat4.matmul (mat1, mat2);
cout << endl << “After Multiplication:” << endl;
mat4.display();
matrix mat5;
mat5.transpose (mat1);
cout << endl << “Transpose of first matrix:” << endl;
mat5.display();
```

POINTER REVIEW!

- When a variable is declared, the memory needed to store its value is assigned a specific location in memory (**its memory address**).
- The address of a variable can be obtained by preceding the name of a variable with an ampersand sign (&), known as address-of operator.

POINTER REVIEW!

For example:

```
foo = &var;
```

- let's assume, in order to help clarify some concepts, that var is placed during runtime in the memory address 1776.
- In this case, consider the following code statements, what would be the results?

```
var = 25;

foo = &var;

bar = var;
```

POINTER REVIEW!

- The variable that stores the address of another variable is what in C++ is called a pointer.
- They can be used to access the variable they point to directly.
- This is done by preceding the pointer name with the dereference operator (*)

baz = *foo

- The declaration of pointers follows this syntax:

type * name;

TRY THIS CODE

```
int firstvalue, secondvalue;
  int * mypointer;

  mypointer = &firstvalue;
  *mypointer = 10;
  mypointer = &secondvalue;
  *mypointer = 20;
  cout << "firstvalue is " << firstvalue << '\n';
  cout << "secondvalue is " << secondvalue << '\n'
```

TRY THIS CODE

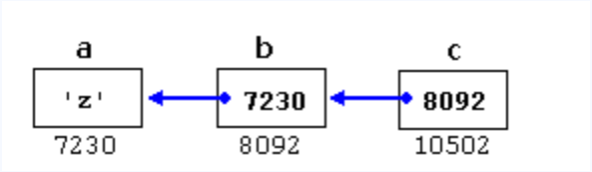
```
int firstvalue = 5, secondvalue = 15;
  int * p1, * p2;
  p1 = &firstvalue;                  // p1 = address of firstvalue
  p2 = &secondvalue;                 // p2 = address of secondvalue
  *p1 = 10;                          // value pointed to by p1 = 10
  *p2 = *p1;                         // value pointed to by p2 = value pointed to by p1
  p1 = p2;                           // p1 = p2 (value of pointer is copied)
  *p1 = 20;                          // value pointed to by p1 = 20
```

TRY THIS CODE

```
int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << ", ";
```

POINTERS TO POINTERS

```
char a;
char * b;
char ** c;
a = 'z';
b = &a;
c = &b;
```




```
c is of type char** and a value of
*c is of type char* and a value of
**c is of type char and a value of
```

NULL POINTERS

- Besides a memory address, there is one additional value that a pointer can hold: **a null value**.
- **A null value** (often shortened to null) is a special value that means something has no value.
- When a pointer is holding a null value, it means the pointer is not pointing at anything.

```
int * p = 0;
int * q = nullptr;
int * r = NULL;
if (p == nullptr) // explicit test for equivalence
        cout << "p is null \n";
    else
        cout << "p is non-null \n";
```