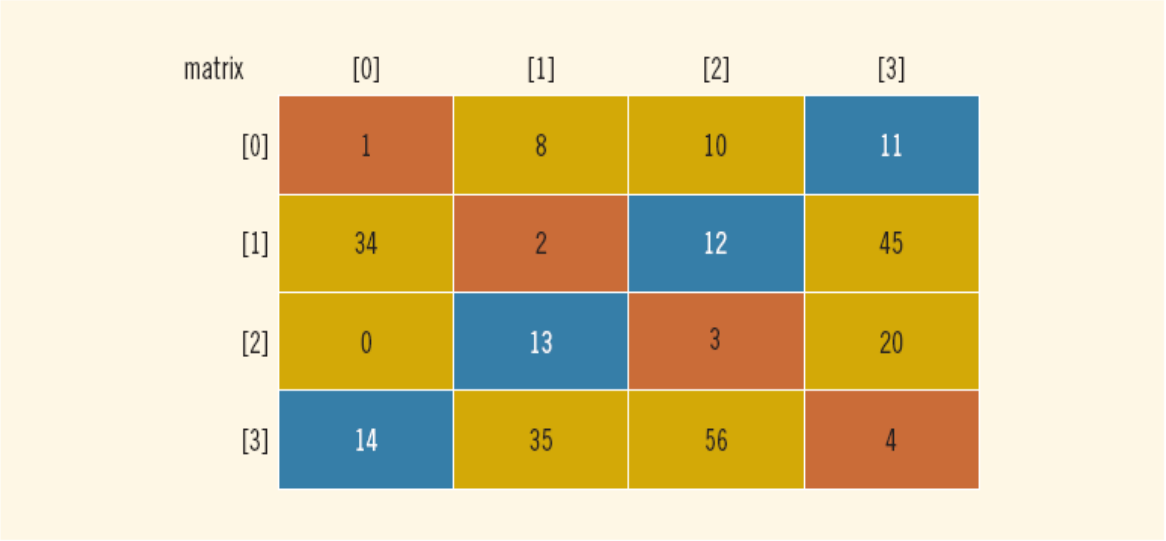
Namespaces & String Type

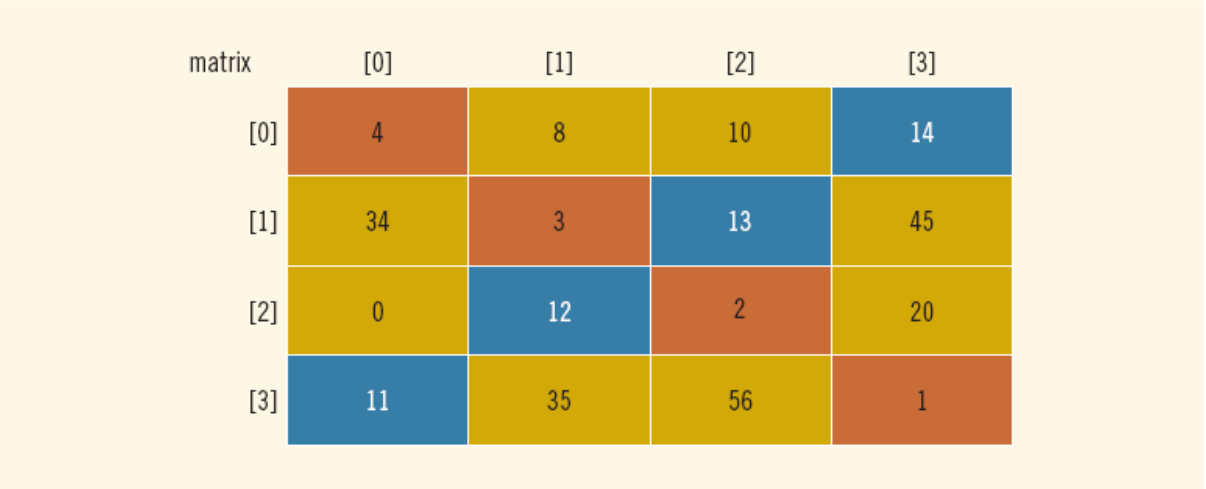
Dr. Mohammed M Abozahhad

Reversing Diagonal (Home Work)

- Before:



- After:



Reversing Diagonal (continued)

- To reverse both the diagonals:

```
  //Reverse the main diagonal
for (row = 0; row < NUMBER_OF_ROWS / 2; row++)
{
    temp = matrix[row][row];
    matrix[row][row] =
      matrix[NUMBER_OF_ROWS - 1 - row][NUMBER_OF_ROWS - 1 - row];
    matrix[NUMBER_OF_ROWS - 1 - row][NUMBER_OF_ROWS - 1 - row]
          = temp;
}

  //Reverse the opposite diagonal
for (row = 0; row < NUMBER_OF_ROWS / 2; row++)
{
    temp = matrix[row][NUMBER_OF_ROWS - 1 - row];
    matrix[row][NUMBER_OF_ROWS - 1 - row] =
                    matrix[NUMBER_OF_ROWS - 1 - row][row];
    matrix[NUMBER_OF_ROWS - 1 - row][row] = temp;
}
```

Objectives

- Learn about the `namespace` mechanism
- Explore the `string` data type, and learn how to use the various `string` functions to manipulate strings

Namespaces

- ANSI/ISO standard C++ was officially approved in July 1998
- Most of the recent compilers are also compatible with ANSI/ISO standard C++
- For the most part, standard C++ and ANSI/ISO standard C++ are the same
  - However, ANSI/ISO Standard C++ has some features not available in Standard C++

Namespaces (continued)

- Global identifiers in a header file used in a program become global in the program
  - Syntax error occurs if an identifier in a program has the same name as a global identifier in the header file
- Same problem can occur with third-party libraries
  - Common solution: third-party vendors begin their global identifiers with _ (underscore)
    - Do not begin identifiers in your program with _

Namespaces (continued)

- ANSI/ISO Standard C++ attempts to solve this problem with the namespace mechanism
- Syntax:

```
namespace namespace_name
{
    members
}
```

where a member is usually a variable declaration, a named constant, a function, or another namespace

Namespaces (continued)

EXAMPLE 8-8

The statement:

```
namespace globalType
{
    const int N = 10;
    const double RATE = 7.50;
    int count = 0;
    void printResult();
}
```

defines `globalType` to be a `namespace` with four members: named constants `N` and `RATE`, the variable `count`, and the function `printResult`.

Namespaces (continued)

- The scope of a `namespace` member is local to the namespace
- Ways a `namespace` member can be accessed outside the `namespace`:

```
namespace_name::identifier
```

```
using namespace namespace_name;
```

```
using namespace_name::identifier;
```

Accessing a `namespace` Member

- Examples:

```
globalType::RATE
globalType::printResult();
```

- After the `using` statement, it is not necessary to precede the `namespace_name::` before the namespace member
  - Unless a `namespace` member and a global identifier or a block identifier have same name

Accessing a `namespace` Member

- Here we can see that more than one variables are being used without reporting any error because they are declared in the different namespaces and scopes.
- #include <iostream>
- using namespace std;
- namespace first {
- int val = 500; // Variable created inside namespace
- }
- int val = 100; // Global variable
- int main() {
- int val = 200; // Local variable
- // These variables can be accessed from outside the namespace using the scope operator ::
- cout << first::val << '\n'; // 500
- cout << ::val << '\n'; // 100
- cout << val << '\n'; // 200
- return 0; }

`string` Type

- To use the data type `string`, the program must include the header file `string`
- The statement:

```
string name = "William Jacob";
```

  declares name to be a string variable and also initializes name to `"William Jacob"`
  - The first character, `'W'`, is in position 0
  - The second character, `'i'`, is in position 1
  - `name` is capable of storing any size string

`string` Type (continued)

- Binary operator `+` and the array subscript operator `[]`, have been defined for the data type `string`
  - `+` performs the string concatenation operation
- Example:

```
str1 = "Sunny";
str2 =  str1 + " Day";
```

stores `"Sunny Day"` into `str2`

- `[]` to get a specific character from the string

Additional `string` Operations

- `length`
- `size`
- `find`
- `substr`
- `swap`

`length` Function

- Returns the number of characters currently in the string
- Syntax:

```
strVar.length()
```

  where `strVar` is variable of the type `string`
- `length` returns an unsigned integer
- The value returned can be stored in an integer variable

```
string firstName;
string name;
string str;

firstName = "Elizabeth";
name = firstName + " Jones";
str = "It is sunny.";
```

Statement	Effect
`cout << firstName.length() << endl;`	Outputs 9
`cout << name.length() << endl;`	Outputs 15
`cout << str.length() << endl;`	Outputs 12

```
string::size_type len;
```

Statement	Effect
`len = firstName.length();`	The value of `len` is 9
`len = name.length();`	The value of `len` is 15
`len = str.length();`	The value of `len` is 12

`size` Function

- `size` is the same as the function `length`
  - Both functions return the same value
- Syntax:

```
strVar.size()
```

  where `strVar` is variable of the type string
- As in the case of the function `length`, the function `size` has no arguments

`find` Function

- Searches a string for the first occurrence of a particular substring
- Returns an unsigned integer value of type `string::size_type`
  - Or `string::npos` if unsuccessful
- Syntax:

```
strVar.find(strExp)
```

```
strVar.find(strExp, pos)
```

  - `strExp` can be a string or a character

`find` Function (continued)

```
string sentence;
string str;
string::size_type position;

sentence = "Outside it is cloudy and warm.";
str = "cloudy";
```

Statement	Effect
`cout << sentence.find("is") << endl;`	Outputs 11
`cout << sentence.find("and") << endl;`	Outputs 21
`cout << sentence.find('s') << endl;`	Outputs 3
`cout << sentence.find('o') << endl;`	Outputs 16
`cout << sentence.find(str) << endl;`	Outputs 14
`cout << sentence.find("the") << endl;`	Outputs the value of `string::npos`
`cout << sentence.find('i', 6) << endl;`	Outputs 8
`position = sentence.find("warm");`	Assigns 25 to `position`

`substr` Function

- Returns a particular substring of a string
- Syntax:

```
strVar.substr(expr1, expr2)
```

`expr1` and `expr2` are expressions evaluating to unsigned integers

  - `expr1` specifies a position within the string (starting position of the substring)
  - `expr2` specifies the length of the substring to be returned

`substr` Function (continued)

```
string sentence;
string str;

sentence = "It is cloudy and warm.";
```

Statement	Effect
`cout << sentence.substr(0, 5) << endl;`	Outputs: `It is`
`cout << sentence.substr(6, 6) << endl;`	Outputs: `cloudy`
`cout << sentence.substr(6, 16) << endl;`	Outputs: `cloudy and warm.`
`cout << sentence.substr(17, 10) << endl;`	Outputs: `warm.`
`cout << sentence.substr(3, 6) << endl;`	Outputs: `is clo`
`str = sentence.substr(0, 8);`	`str = "It is cl"`
`str = sentence.substr(2, 10);`	`str = " is cloudy"`

`swap` Function

- Interchanges contents of two string variables
- Syntax:

```
strVar1.swap(strVar2);
```

  where `strVar1` and `strVar2` are string variables

- Suppose you have the following statements:

```
string str1 = "Warm";
string str2 = "Cold";
```

- After `str1.swap(str2);` executes, the value of `str1` is `"Cold"` and the value of `str2` is `"War"`