Introduction to C++ vector & string

Vectors

- Instead of built-in arrays, in real applications you should use the *vector container classes,* which is part of the C++ *Standard Template Library. Why?*
  - You cannot copy built in arrays
  - There is no boundary checking for built in arrays
  - You cannot resize them if they are created on the stack
  - You must know the array size at compile time (not run time)
  - It is less convenient when passing built-in arrays to functions
- To use a vector, you require to include header file:

```
#include <vector>
```

Vectors – Declaration

- The syntax to declare a vector objects is:

```
vector<data type> arrayName;
```

  - Means you can have a vector of any type

- For example:

```
vector<double> a; // creates an empty vector

vector<double> a(0);// also is an empty vector

vector<double> a(5);// has 5 elements all with 0
```

Vectors – Declaration & Initialization

- All elements of the vector can be initialized to the same non-zero start value

```
vector<double> a(3, 300); // has three elements
                          // each containing 300.0
```

- If no value is provided, the elements are initialized to zero
- You can copy vectors:

```
vector<int> v(2);
v[0] = 15;          // More on accessing elements soon!
v[1] = 31;
vector<int> copy(a); // elements of "copy" are same as
                     // the elements of "v"
```

Vectors – Accessing

- As with built-in arrays the elements of a vector are ordered – they form an ordered list
  - Each element of the list has an index that always starts from zero and increases by one to the size of the list minus one
- Similarly, you can also access each element of the vector by using square brackets, `[]`, and the appropriate index
- For example

```
vector<double> grades(5);
grades[0] = 5.0;
grades[4] = 8.0;
grades[2] = 3;
```

Vectors – Example 1

- Write code to read in 10 numbers and print them in reverse order.
  - Notice how member functions size and at are used:

```
#include <vector>
int main()
{
   vector< int > numbers(10);

   for( int i = 0 ; i < number.size() ; ++i )
   {
     cout << "Please enter an integer number for:" << endl;
     cin >> numbers [ i ];
   }

   for( int i = number.size()-1 ; i >= 0 ; --i )
      cout << numbers.at(i) << endl;
   return 0;
}
```

Vectors – Resizing

- The **`resize()`** member function is used to change the size of a vector

```
vector<int> a(originalSize);   // size = originalSize
a.resize(newSize);             // size = newSize
```

- Here's a short example of reading in an unknown number of values

```
int counter = 0;
char answer;
vector <double> dArray;          // zero size
do
{
   dArray.resize( ++counter );  // increase the size
   cout << "Enter a value: \n";
   cin >> dArray[counter - 1];  // why subtract 1?

   cout << "Do you want to enter more (y/n)?" << endl;
   cin >> answer;
} while( (answer == 'y') || (answer == 'Y') );

cout << "You entered " << dArray.size() << " values\n";
```

Vectors – Other Member Functions

- Vectors have a number of member functions and operators
  - You will only be responsible for knowing the ones covered in class

- Examples:

```
vector <double> v(3, 6.5);
v.empty();                  // returns true if vector is empty
v.push_back( value );       // inserts value to the end of vector
v.pop_back();               // returns and removes the last element
v.at( index );              // same as v[index]
v.assign( n, value );       // assigns value to the first n elements
v.clear();                  // deletes all elements
```

Passing vectors to the functions

- As with other classes, you can pass one or more vectors as arguments into functions

- As discussed previously, it is recommended (although not strictly required) that vectors *always* be passed by reference
  - If necessary, to protect the argument from being changed, we use the `const` keyword – this approximates the behaviour of pass-by-value

Vectors – Passing to a Function

- For example

```
double average( const vector<int>& data ){
    double sum = 0;
    for( int i = 0 ; i < data.size() ; ++i ){
            sum += data[i];
    }
    return sum / data.size();
}
```

- You can also return vectors from functions. For example, you could have a function that returns a vector containing only positive values of vector v:

```
vector<int> all_positives(const vector<int> & v );
```

Vector of Vectors

- You can use **`typedef`** keyword to create a new data type *name* from existing data types. The syntax is:

```
typedef <existing_type_name> <new_type_name>;
```

- You can define a vector of vector to use it as matrix:

```
typedef vector<int> row;
typedef vector<row> matrix;
```

- Or , "**`matrix`**" can be declared as follow:

```
typedef vector< vector<int> > matrix;
```

- Now we can declare a matrix variable **`m`** as follows

```
matrix m;
```

- **`m`** has no elements (zero rows, zero columns).

Vector of Vectors: Allocating Memory

- Setting rows and columns:

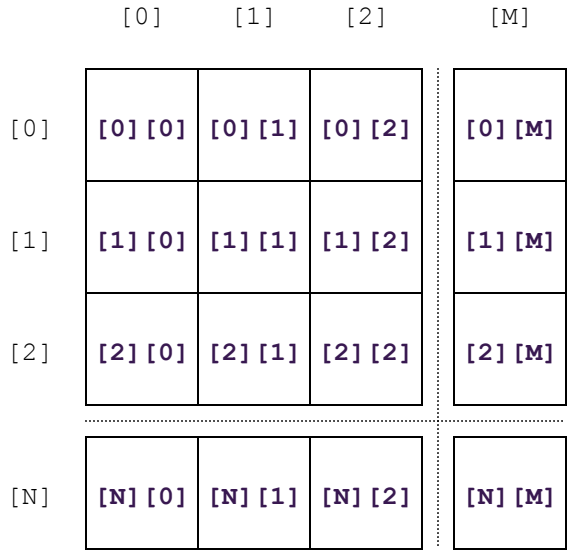
```
const int numRows = 3; // number of rows
const int numCols = 5; // number of columns

m.resize( numRows );   // set number of rows

for( int j = 0 ; j < numRows ; j++ )
   // ...and set the number of columns for each row
   m.at(j).resize( numCols );
```

- You can also write:

```
vector <vector <int> > m (3, vector<int> (5));
```

Vector of Vectors: Accessing Elements

- When accessing data from a matrix, you must specify a row index and a column index;
- Both row and column indices start at zero and increment by one



- Example:

```
for( int i = 0 ; i < m.size() ; i++ )
   for( int j = 0 ; j < m.at(i).size() ; j++ )
         cout << m.at(i).at(j) << endl;
```

Vector of Vectors: Resizing

- You can resize the matrices as follows:
- For example, to resize the matrix to have `numRows` rows and `numCols` columns respectively, we can use the following code

```
int numRows = 20;
int numCols = 7;
m.resize( numRows );  // resize rows

for( int j = 0 ; j < numRows ; j++ )
   m.at(j).resize( numCols ); // resize columns
```

Vector of Vectors: passing to the functions

- If a `matrix` is to be used as a type for function parameters, it must be defined *prior* to the prototype of the function

```
#include <vector>
#include <iostream>
using namespace std;

typedef vector<int> row;
typedef vector<row> matrix;

void display( const matrix& m );

int main()
{
  // call display function
}

void display( const matrix& m )
{
  // display rows and columns of matrix

}
```

Vector of Class Objects:

- You can have vector of C++ struct of class objects:

```
class Point{
    private:
      double x;
      double y;
    public:
      Point(double a, double b): x(0), y(0) {}
      double getX()const;
      double getY()const;
      void setX(double v) { x = v; }
      void setY(double v) { y = v; }
};
int main(){
   vector <Point> p (3);
   p.at(0).setX(20);
   p.at(0).setY(30)
   cout << "\n x coordinate of 1st element is: " << p.at(0).getX();
   ...
   return 0;
}
```

Introduction to C++ string class

Strings – Declaration

- There are several different ways to declare a string object but in all cases, you must have `#include <string>` at the top of your program

- To create an empty string, use the following syntax

```
string stringName;
```

- To create a string and initialize it, use the following syntax

```
string stringName( "Initialization String Here" );
```

  - Example: `string course( "ENCM 339" );`

- It should be noted that spaces are allowed.
  - In the above example, the string has 8 characters

Strings – Declaration

- To create a string and initialize it to a character, use the following syntax

```
string stringName( n, 'character' ); // where n = 1
```

  - Example: `string firstInitial( 1,'J' );`

- If n is larger than 1, you will create a string containing n of the character specified
  - This is similar to initializing a vector

- To create a string as a copy of another string, use the following:

```
string stringName( otherStringName );
```

  - Example:
```
string student1( "John Smith" );
string student2( student1 );
```

- The contents of the second string is also "John Smith"

Strings – Accessing and Changing Elements

- Elements of a string are accessed in much the same way you access elements of a vector
  - Square bracket operator, `[]`
  - The `.at()` member function

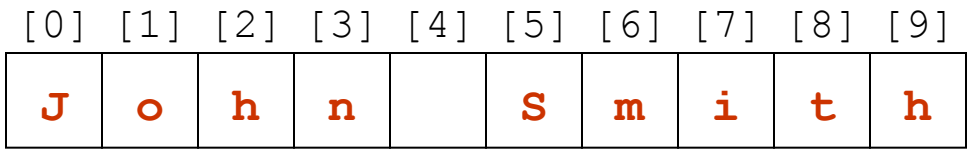
- The index into the string should be in the range of 0 to N-1 where N is the length of the string
  - More on determining the length of a string shortly

- Each element of a string is a **`char`**
  - This element can be used just like any other char
    - Assign it a value
    - Assign its value to another variable
    - Output it to file (more on this later)
    - etc.

Strings – Accessing and Changing Elements

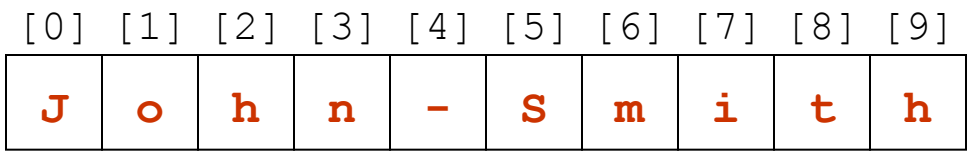
- Here is a simple example

```
string student_name("John Smith");
```



- You can also change a single element of the string:

```
student_name.at(4) = '-';
cout << student_name;        // Outputs: "John-Smith"
```



Strings – Determining Size

- You can use the `.size()` or `.length()` member function to determine the number of characters in a string, *excluding the terminating character*
  - The return value is of type `int`

- For example

```
string s1;
string s2(1,' ');
string s3("D'oh!");

int a = s1.size();          // a is 0
int b = s2.size();          // b is 1
int c = s3.size();          // c is 5
```

Strings – Assignment

- You can use the assignment operator with strings
  - The string on the left hand side of the operator will be automatically resized

```
string a( "First String" );
string b( "Second String" );

cout << a << endl;    // outputs: "First String"
cout << b << endl;    // outputs: "Second String"

a = "Third String";
cout << a << endl;    // outputs: "Third String"

b = 'z';
cout << b << endl;    // outputs: "z"

b = a;
cout << b << endl;    // outputs: "Third String"
```

Strings – Functions

- As with other user-defined objects, it is recommended that a string be passed by reference to a function, although this is not required
  - Use the `const` keyword if necessary to protect the argument from being changed

```
void get_filename( string &filename )
{
    cout << "Enter file name\n";
    cin >> filename;
    cout << "You entered " << filename << endl;
}

void main()
{
    string input_file_name;

    // get input file name
    get_filename( input_file_name );
    ...
}
```

Strings – Appending

- It is often required that you build a string incrementally. To do this, use the `.append()` member function or the `+=` operator
  - This function can take a built-in C++ string, a character, or another string as an input argument

```
string name( "main" );
string end( " is the filename" );
string program_name;

program_name.append( name );     // string
program_name.append( 1, '.' );   // char
program_name.append( "cpp" );    // built-in string
program_name += end;             // another string
```

Outputting C++ strings

- We've already seen that we can output a string by using the insertion operator (<<)

```
string message( "Hello" );
cout << message;              // outputs: "Hello"
```

Reading C++ strings

- As we have seen, you can use the input operator to read in string objects
  - However, this will only read until the first *white space character* is reached (space, tab, newline)
  - In other words, you can only read one word at a time!

```
string first_word, second_word;

cout << "Enter two words: \n";
cin >> first_word >> second_word;
```

- The results of the read operation will be the same if the input had either of the following forms:

```
Hello  class
```

OR

```
Hello
class
```

Strings – Inputting

- What if you want to read multiple words at once?
  - In this case, you can use the `getline()` function
  - This is **not** a member function, but it will read a line of text up to but not including the delimiter character (which defaults to the new line character – `'\n'`)

```
string line_of_text;
cout << "Enter a line of text: ";
getline( cin, line_of_text, '\n' );
```

- As you might expect, this will read input from the keyboard, but you can also read from file (more on files later in the course)

Strings – Removing and Inserting Characters

- To remove characters from a string, use the `.erase(k, n)` member function
  - This deletes n characters starting at index k

- To insert characters, use the `.insert(k, "string")` member function, which inserts `"string"` starting at index `k`

```
string course("ENCM339 class");
course.erase(4, 3);          // course is "ENCM class"
course.insert(4, " 369");    // course is "ENCM 369 class"
```

Strings – Comparisons

- Strings can be compared to each other in the same way that integers are compared, that is, using the comparison operators

  `<` `>` `<=` `>=` `!=` `==`

- For strings, the ordering is basically alphabetical (for a given 'case' – upper or lower)
  - Use the ASCII list to determine the correct order

- Here are a couple other useful member functions to manipulate C++ strings:

```
string str = "ABCD";
str.empty();                  // Returns true if the string is empty.
str.substr( exp1, exp2 ); // Returns the sub-string of str starting at
                              //  index exp1 of length exp2.
```