Vectors and Strings

Read more in cplusplus.com

Askar Khaimuldin

"Well, I'll eat it," said Alice, "and if it makes me grow larger, I can reach the key; and if it makes me grow smaller, I can creep under the door; so either way I'll get into the garden."

Lewis Carroll, Alice's Adventures in Wonderland

Strings

#include <string>

String objects has several build-in functions.

It can increase its length dynamically

```
string s;
s = s + 'c';
s = s + "sse";
cout << s << endl;
```

Strings

Moreover, it is possible to use a string object as an array of chars

```
string s;
cin >> s;
for (int i = 0; i < s.length(); i++){
    if (islower(s[i])){
        s[i] = toupper(s[i]);
    }
}
```

Nevertheless, it is prohibited to use cells of memory which are not meant to be used

```
string s;
s[3] = 'c'; // since you haven`t reserved this cell yet
```

Strings vs Char array

A character array is simply an array of characters can terminated by a null character.

A string is a class which defines objects that be represented as stream of characters.

It is the way more safe to use string rather than char array =)

Strings

1. getline() :- This function is used to store a stream of characters as entered by the user in the object memory.

2. push_back() :- This function is used to input a character at the end of the string.

3. pop_back() :- Introduced from C++11(for strings), this function is used to delete the last character from the string

Strings

4. find() :- Searches the string for the first occurrence of the sequence specified by its arguments.

5. find_first_of() :- Searches the string for the first character that matches any of the characters specified in its arguments.

6. insert() :- Inserts additional characters into the string right before the character indicated by pos (or p):

Iterators in C++ STL

Iterators are used to point at the memory addresses of STL containers.

They are primarily used in sequence of numbers, characters etc.

They reduce the complexity and execution time of program.

1. begin() :- This function is used to return the beginning position of the container.
2. end() :- This function is used to return the end position of the container.
3. advance() :- This function is used to increment the iterator position till the specified number mentioned in its arguments.

Iterators in C++ STL

```
string s;
cin >> s;
string::iterator it;
for (it = s.begin(); it < s.end(); it++){
    cout << *it;
}
```

The dereference of “it” object is an char elements of string.

Vectors

Array

- cannot change the length

Vector

- the same purpose as arrays
  - ✔ except can change length while the program is running

Like an array, a vector has a base type, and like an array, a vector stores a collection of values of its base type.

Vectors

Library:

#include <vector>

Declaration:

vector <base_type> name;

Example:

```
vector  <int> v;
vector  <int> v(10);
```

Vectors

- To add an element to a vector for the first time, you normally use the member function ***push_back***.

Example:

```
vector<double> sample;
Sample[0]=1;
sample.push_back(0.0);
sample.push_back(1.1);
sample.push_back(2.2);
```

Vectors

- The number of elements in a vector is called the **size of the vector.**
- The member function **size** can be used to determine how many elements are in a vector.

Example:

```
for (int i = 0; i < sample.size( ); i++)
cout << sample[i] << endl;
```

```
// Demonstrating C++ Standard Library class template vector.
#include <iostream>
using std::cout;
using std::cin;
using std::endl;

#include <iomanip>
using std::setw;

#include <vector>
using std::vector;

void outputVector( const vector< int > & ); // display the vector
void inputVector( vector< int > & ); // input values into the
   vector

int main()
{
   vector< int > integers1( 7 ); // 7-element vector< int >
   vector< int > integers2( 10 ); // 10-element vector< int >
```

create two *vector* objects that store values of type int

By default, all the elements of each *vector* object are set to 0

```
// print integers1 size and contents
 cout << "Size of vector integers1 is " << integers1.size()
      << "\nvector after initialization:" << endl;
 outputVector( integers1 );

 // print integers2 size and contents
 cout << "\nSize of vector integers2 is " << integers2.size()
      << "\nvector after initialization:" << endl;
 outputVector( integers2 );

 // input and print integers1 and integers2
 cout << "\nEnter 17 integers:" << endl;
 inputVector( integers1 );
 inputVector( integers2 );

 cout << "\nAfter input, the vectors contain:\n"
      << "integers1:" << endl;
 outputVector( integers1 );
 cout << "integers2:" << endl;
 outputVector( integers2 );;
```

```
// use inequality (!=) operator with vector objects
 cout << "\nEvaluating: integers1 != integers2" << endl;

 if ( integers1 != integers2 )
    cout << "integers1 and integers2 are not equal" << endl;

 // create vector integers3 using integers1 as an
 // initializer; print size and contents
 vector< int > integers3( integers1 ); // copy constructor

 cout << "\nSize of vector integers3 is " << integers3.size()
      << "\nvector after initialization:" << endl;
 outputVector( integers3 );

 // use overloaded assignment (=) operator
 cout << "\nAssigning integers2 to integers1:" << endl;
 integers1 = integers2; // integers1 is larger than integers2

 cout << "integers1:" << endl;
 outputVector( integers1 );
 cout << "integers2:" << endl;
 outputVector( integers2 );
```

vector objects can be compared directly with the !=operator

a new vector object that is initialized with the contents of an existing *vector*.

the assignment (=) operator can be used with *vector* objects.

```
// use equality (==) operator with vector objects
  cout << "\nEvaluating: integers1 == integers2" << endl;

  if ( integers1 == integers2 )
     cout << "integers1 and integers2 are equal" << endl;
// use square brackets to create rvalue
  cout << "\nintegers1[5] is " << integers1[ 5 ];

  // use square brackets to create lvalue
  cout << "\n\nAssigning 1000 to integers1[5]" << endl;
  integers1[ 5 ] = 1000;
  cout << "integers1:" << endl;
  outputVector( integers1 );

  // attempt to use out-of-range subscript
  cout << "\nAttempt to assign 1000 to integers1.at( 15 )" <<
   endl;
  integers1.at( 15 ) = 1000; // ERROR: out of range
  return 0;
} // end main
```

the programmer must ensure that operations using [] do not accidentally attempt to manipulate elements outside the bounds of the vector.

vector does provide bounds checking in its member function at

```
// output vector contents
void outputVector( const vector< int > &array )
{
   size_t i; // declare control variable
   for ( i = 0; i < array.size(); i++ )
   {
      cout << setw( 12 ) << array[ i ];
      if ( ( i + 1 ) % 4 == 0 ) // 4 numbers per row of output
         cout << endl;
   } // end for

   if ( i % 4 != 0 )
      cout << endl;
} // end function outputVector

// input vector contents
void inputVector( vector< int > &array )
{
   for ( size_t i = 0; i < array.size(); i++ )
      cin >> array[ i ];
} // end function inputVector
```

Two / Three / Multi Dimensioned arrays using vector

- A two dimensional array is a vector of vectors.
- The vector contructor can initialize the length of the array and set the initial value.
- Example of a vector of vectors to represent a two dimensional array:
  - vector< vector<int> > vI2Matrix(3, vector<int>(2,0));

```
#include <iostream>
#include <vector>
using namespace std;
void main() {
    // Declare size of two dimensional array and initialize.
    vector< vector<int> > vI2Matrix(3, vector<int>(2,0));
    vI2Matrix[0][0] = 0;
    vI2Matrix[0][1] = 1;
    vI2Matrix[1][0] = 10;
    vI2Matrix[1][1] = 11;
    vI2Matrix[2][0] = 20;
    vI2Matrix[2][1] = 21;
    cout << "Loop by index:" << endl;
    int ii, jj;
    for(ii=0; ii < 3; ii++) {
      for(jj=0; jj < 2; jj++) {
            cout << vI2Matrix[ii][jj] << endl;
      }
    }
}
```

```
Loop by index:
0
1
10
11
20
21
```

Two / Three / Multi Dimensioned arrays using vector

- A three dimensional vector would be declared as:

```
#include <iostream>
#include <vector>
using namespace std;
void main() {
    // Vector length of 3 initialized to 0
    vector<int> vI1Matrix(3,0);
    // Vector length of 4 initialized to hold another
    // vector vI1Matrix which has been initialized to 0
    vector< vector<int> > vI2Matrix(4, vI1Matrix);
    // Vector of length 5 containing two dimensional vectors vector<
    vector< vector<int> > > vI3Matrix(5, vI2Matrix);
    . . .
```

```
#include <iostream>
#include <vector>
using namespace std;
void main() {
vector< vector< vector<int> > > vI3Matrix(2, vector<
   vector<int> > (3, vector<int>(4,0)) );
for(int kk=0; kk<4; kk++) {
   for(int jj=0; jj<3; jj++) {
    for(int ii=0; ii<2; ii++) {
         cout << vI3Matrix[ii][jj][kk] << endl;
    }
   }
}
}
```

```
//Example of iterators used with a two dimensional vector.
#include <iostream>
#include <vector>
using namespace std;

void main()
{
   vector< vector<int> > vI2Matrix;    // Declare two dimensional
    array
   vector<int> A, B;
   vector< vector<int> >::iterator iter_ii;
   vector<int>::iterator iter_jj;

   A.push_back(10);
   A.push_back(20);
   A.push_back(30);
   B.push_back(100);
   B.push_back(200);
   B.push_back(300);
```

```
 vI2Matrix.push_back(A);
 vI2Matrix.push_back(B);

 cout << endl << "Using Iterator:" << endl;

 for(iter_ii=vI2Matrix.begin(); iter_ii!=vI2Matrix.end(); iter_ii++)
   {
      for(iter_jj=(*iter_ii).begin(); iter_jj!=(*iter_ii).end(); iter_jj++)
      {
         cout << *iter_jj << endl;
      }
   }
}
```

```
Using Iterator:
10
20
30
100
200
300
```

Readings:

- **C++ How to Program,** By H. M. Deitel
  - Chapter 7. Arrays and Vectors

THANKS FOR YOUR ATTENTION!