CS2413: Data structure

INSTRUCTOR: SAYED ERFAN AREFIN

LAB 3 – ARRAY AND VECTOR

Topics

- C++ Standard Template Library (STL)
- std::array
- std::vector – Dynamic Array
- Lab assignment 3

C++ Standard Template Library

- A set of C++ template classes to provide common programming data structures and functions such as lists, stacks, arrays
- STL has four components.
  - Algorithms
  - Containers (array, vector, list etc.)
  - Functions
  - Iterators

std::array vs. C-style array

- std::array (array class) advantages over C-style array
- Array class knows its own size, whereas C-style arrays lack this property.
- It can be passed to or returned from functions by value
- It allows you to use it as a fundamental type (i.e., you can pass-by-value, assign, whereas a standard C array cannot be assigned or copied directly to another array)

```
int myArray[5] = {8, 21, 25, 5, 14};

array<int,5> newArray = {8, 21, 25, 5, 14};
```

Operators on Array

- front() returns the first element of array
- back() returns the last element of array
- size() returns the number of elements in array.
- This is a property that C-style arrays lack

```
1  #include <iostream>
2  #include <array>
3
4  using namespace std;
5
6  int main(void) {
7
8      array<int, 5> arry = {100, 200, 300, 400, 500};
9
10     /* print first element */
11     cout << "First element of array = " << arry.front()
12         << endl;
13
14     /* modify value */
15     arry.front() = 123;
16
17     /* print modified value */
18     cout << "After modification first element of array = " << arry.front()
19         << endl;
20
21      /* print last element */
22     cout << "Last element of array = " << arry.back()
23         << endl;
24
25      /* print size of array */
26      cout << "Size of the array = " << arry.size()
27         << endl;
28
29     return 0;
30 }
```

Operators on Array

- swap() swaps all elements of one array with another

```
1   #include <iostream>
2   #include <array>
3
4   using namespace std;
5
6   int main(void) {
7
8       array<int, 3> arry1 = {100, 200, 300};
9       array<int, 3> arry2 = {501, 502, 503};
10
11      cout << "Before swap operation\n";
12      cout << "arry1 = ";
13      for (int &i : arry1) cout << i << " ";
14      cout << endl;
15
16      cout << "arry2 = ";
17      for (int &i : arry2) cout << i << " ";
18      cout << endl << endl;
19
20      arry1.swap(arry2);
21
22      cout << "After swap operation\n";
23      cout << "arry1 = ";
24      for (int &i : arry1) cout << i << " ";
25      cout << endl;
26
27      cout << "arry2 = ";
28      for (int &i : arry2) cout << i << " ";
29      cout << endl;
30
31      return 0;
32  }
```

Operators on Array

- empty() returns true when array size is zero else returns false
- fill() is used to fill the entire array with a particular value
- Ternary Operator (?:)
- (expression 1) ? expression 2 : expression 3

```
1   #include <iostream>
2   #include <array>
3
4   using namespace std;
5
6   int main(void) {
7
8       /* Declaring array 1 and 2 */
9       array<int, 0> array1;
10      array<int, 5> array2;
11
12
13       // checkign the size of array
14
15       array2.empty() ?  cout << "array2 is empty" << endl : cout << "array2
        is not empty" << endl;
16
17       array1.empty() ?  cout << "array1 is empty" << endl : cout << "array1
        is not empty" << endl;
18
19       // fill array 1
20       array2.fill(6);
21
22       //print out array 2
23       for (int i =0; i < array2.size(); i++){
24         cout << array2[i]<< " " ;
25       }
26      cout <<   endl;
27      return 0;
28  }
```

Vector in C++ STL

- Vectors are same as dynamic arrays with the ability to resize itself automatically when an element is inserted or deleted, with their storage being handled automatically by the container
- Vector elements are placed in contiguous storage so that they can be accessed and traversed using iterators.
- In vectors, data is inserted at the end using push_back()

Operators on Vector

- **begin()** – Returns an iterator pointing to the first element in the vector
- **end()** – Returns an iterator pointing to the theoretical element that follows the last element in the vector
- **rbegin()** – Returns a reverse iterator pointing to the last element in the vector (reverse beginning). It moves from last to first element
- **rend()** – Returns a reverse iterator pointing to the theoretical element preceding the first element in the vector (considered as reverse end)

Operators on Vector

- begin()
- end()
- rbegin()
- rend()

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   int main(void) {
7
8     vector<int> v1;
9     vector<int> :: iterator i;
10    vector<int> :: reverse_iterator ir;
11
12    // pushin some values
13    for (int j=1 ; j <=10; j++){
14      v1.push_back(j);
15    }
16
17    // using the begin() and end()
18    cout << "begin and end of vector v1" << endl;
19    for (i=v1.begin(); i != v1.end(); i++){
20      cout << *i << " ";
21    }
22    cout << endl;
23
24    // using the rbegin()() and rend()
25    cout << "rbegin and rend of vector v1" << endl;
26    for (ir=v1.rbegin(); ir != v1.rend(); ir++){
27      cout << *ir << " ";
28    }
29    cout << endl;
30
31    return 0;
32  }
```

Accessing the Elements

- **reference operator [g]** – Returns a reference to the element at position ‘g’ in the vector
- **at**(g) – Returns a reference to the element at position ‘g’ in the vector
- **front**() – Returns a reference to the first element in the vector
- **back**() – Returns a reference to the last element in the vector

Accessing the Elements

- reference operator []
- at()
- front()
- back()

```
1   #include <iostream>
2   #include <vector>
3
4   using namespace std;
5
6   int main(void) {
7
8     vector<int> v1;
9
10    // pushing some values
11    for (int j=1 ; j <=10; j++){
12      v1.push_back(j);
13    }
14
15    // using the reference operator
16    cout << "reference operator: v1[3]= " << v1[3] << endl;
17
18    // using at()
19    cout << "at(): v1[3]= " << v1.at(3) << endl;
20
21    // using front()
22    cout << "front(): v1.front()= " << v1.front() << endl;
23
24    // using back()
25    cout << "back(): v1.back()= " << v1.back() << endl;
26
27
28    return 0;
29  }
```

Modifying the Elements

- **assign**(input_iterator first, input_iterator last) – Assigns new content to vector and resize
- **assign**(size_type n, const value_type g) – Assigns new content to vector and resize
- **push_back**(const value_type g) – Adds a new element ‘g’ at the end of the vector and increases the vector container size by 1
- **pop_back()** – Removes the element at the end of the vector, i.e., the last element and decreases the vector container size by 1

Modifying the Elements

```cpp
#include <iostream>
#include <vector>

using namespace std;

void printVector(vector<int> v1){
  // printing the vector
  cout << "Printing the vector" << endl;
  for (int i =0; i < v1.size(); i++){
    cout << v1[i] << " ";
  }
  cout << endl;
}

```

```cpp
int main(void) {

  vector<int> v1;

  cout << "push_back" << endl;
  // pushing some values
  for (int j=1 ; j <=10; j++){
    v1.push_back(j);
  }

  // printing the vector
  printVector(v1);

  cout << endl;
  cout << "pop_back" << endl;
  // poping some values
  v1.pop_back();
  v1.pop_back();

  // printing the vector
  printVector(v1);

  // assign value example 1
  cout << endl;
  cout << "assign (example 1)" << endl;
  v1.assign(v1.begin(), v1.begin() + 3);
  // printing the vector
  printVector(v1);

  // assign value example 2
  cout << endl;
  cout << "assign (example 2)" << endl;
  v1.assign(6, 20);
  // printing the vector
  printVector(v1);

  return 0;
}
```

Thank you!

Questions?