Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session : Template Class “vector”-Part 1

Quick Recap of Relevant Topics

- Object-oriented programming with structures and classes
- Template classes and functions
- C++ Standard Library
  - The “string” class

Overview of This Lecture

IIT Bombay

- The template class “vector”

Acknowledgment

- Much of this lecture is motivated by the treatment in

  An Introduction to Programming Through C++

  by Abhiram G. Ranade

  McGraw Hill Education 2014

The “vector” class

IIT Bombay

- For representing and manipulating one dimensional arrays of objects
  - Template class: can be instantiated with specific type
  - Uses dynamically allocated array to store elements
    Array can grow or shrink in size
  - Dynamic memory management built in
- “vector” objects are container objects
- Must use **#include <vector>** at start of program
- Large collection of member functions
  - We’ll see only a small subset

Simple Programming using "vector"

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
   vector<int> intVec;
   vector<float> floatVec(20);
   vector<char>  charVec(5, 'a');
   cout << intVec.size() << " " << floatVec.size() << endl;
   for (int i = 0; i < 5; i++) {cout << charVec[i];}
   cout << endl;  return 0;
}
```

0 20

aaaaa

Accessing Elements using [] and at

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec(5);
  int index;
  for (int i = 0; i < 5; i++) { intVec[i] = i; }
  cout << "Give an index: ";
  cin >> index;
  cout << "Value at index " << index << " is "  << intVec.at(index) << endl;
  return 0;
}
```

intVec[100] vs intVec.at(100):
Illegal memory/garbage access
vs
out_of_range exception

index: 3
Value at index 3 is 3

Accessing Special Elements using front and back

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec(5);
  for (int i = 0; i < 5; i++) { intVec.at(i) = i;}
  cout << “Front element is: “ << intVec.front() << endl;
  cout << “Back element is: “ << intVec.back() << endl;
  return 0;
}
```

Front element is: 0
Back element is: 4

Appending Element to a Vector

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec;
  cout << “Initial size: “ << intVec.size() << endl;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  cout << “Final size: “ << intVec.size() << endl;
  for (int i = 0; i < 5; i++) { cout << intVec.at(i) << “ “; }
  return 0;
}
```

Initial size: 0

Final size: 5

0 1 2 3 4

Deleting Element From End of a Vector

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec;
  cout << “Initial size: “ << intVec.size() << endl;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  cout << “Final size: “ << intVec.size() << endl;
  intVec.pop_back();
  cout << “Size after pop back: “ << intVec.size() << endl;
  for (int i = 0; i < 4; i++) { cout << intVec.at(i) << “ “;}
  return 0;
}
```

Size after pop back: 4

0 1 2 3

Recall: C++ Iterator

- An object that points to an element in a collection of elements, and can be used to iterate through the elements in the collection
- Like a pointer, but not exactly the same
- Must support ++ (increment) and * (dereference) operations

Iterator Related Functions in "vector" Class

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  intVec.push_back(-1);
  for (vector<int>::iterator it = intVec.begin(); it != intVec.end(); it++)  {
      cout << *it << " ";
  }
  return 0;
}
```

begin(), end()
member functions

0 1 2 3 4 -1

Iterator Related Functions in "vector" Class

IIT Bombay

```
#include <iostream>
#include <vector>
using namespace std;
int main() {
  vector<int> intVec;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  intVec.push_back(-1);
  for (vector<int>::reverse_iterator rit = intVec.rbegin(); rit != intVec.rend(); rit++)  {
      cout << *rit << " ";
  }
  return 0;
}
```

rbegin(), rend()
member functions

-1 4 3 2 1 0

Inserting and Deleting Elements in the Middle

IIT Bombay

```
int main() {
  vector<int> intVec;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  for (int i = 0; i < 5; i++) { cout << intVec.at(i) << " ";}   cout << endl;
  vector<int>::iterator it = intVec.begin() + 2;
  intVec.insert(it, 0);
  cout << "Size after insert: " << intVec.size() << endl;
  for (int i = 0; i < intVec.size(); i++) { cout << intVec.at(i) << " ";}  cout << endl;
  it = intVec.begin + 3;  intVec.erase(it);
  cout << "Size after delete: " << intVec.size() << endl;
  for (int i = 0; i < intVec.size(); i++) { cout << intVec.at(i) << "
  return 0;
}
```

0 1 2 3 4

Size after insert: 6

0 1 0 2 3 4

Inserting and Deleting Elements in the Middle

IIT Bombay

```
int main() {
  vector<int> intVec;
  for (int i = 0; i < 5; i++) { intVec.push_back(i); }
  for (int i = 0; i < 5; i++) { cout << intVec.at(i) << " ";}   cout << endl;
  vector<int>::iterator it = intVec.begin() + 2;
  intVec.insert(it, 0);
  cout << "Size after insert: " << intVec.size() << endl;
  for (int i = 0; i < intVec.size(); i++) { cout << intVec.at(i) << " ";}  cout << endl;
  it = intVec.begin + 3;  intVec.erase(it);
  cout << "Size after delete: " << intVec.size() << endl;
  for (int i = 0; i < intVec.size(); i++) { cout << intVec.at(i) << " ";}  cout << endl;
  return 0;
}
```

Size after delete: 5

0 1 0 3 4

Resizing a "vector"

IIT Bombay

- push_back(), pop_back result in automatic resizing
- C++ allows explicit resizing of vectors

```
int main() {
  vector<int> intVec(5, 0);
   intVec.resize(10, -1);
   cout << "Size after resizing: " << intVec.size() << endl;
   for (int i = 0; i < 10; i++) { cout << intVec.at(i) << " "; }
   cout << endl;
   intVec.resize(7);  cout << "New size: " << intVec.size() << endl;
   for (int i = 0; i < 7; i++) { cout << intVec.at(i) << " "; }
   return 0;
}
```

Size after resizing: 10

Resizing a "vector"

IIT Bombay

- push_back(), pop_back result in automatic resizing
- C++ allows explicit resizing of vectors

```
int main() {
  vector<int> intVec(5, 0);
   intVec.resize(10, -1);
   cout << "Size after resizing: " << intVec.size() << endl;
   for (int i = 0; i < 10; i++) { cout << intVec.at(i) << " "; }
   cout << endl;
   intVec.resize(7);  cout << "New size: " << intVec.size() << endl;
   for (int i = 0; i < 7; i++) { cout << intVec.at(i) << " "; }
   return 0;
}
```

0 0 0 0 0 -1 -1 -1 -1 -1

Resizing a "vector"

IIT Bombay

- push_back(), pop_back result in automatic resizing
- C++ allows explicit resizing of vectors

```
int main() {
  vector<int> intVec(5, 0);
   intVec.resize(10, -1);
   cout << "Size after resizing: " << intVe                () << endl;
   for (int i = 0; i < 10; i++) { cout << int      .at(i) << " "; }
   cout << endl;
   intVec.resize(7);  cout << "New size: " << intVec.size() << endl;
   for (int i = 0; i < 7; i++) { cout << intVec.at(i) << " "; }
   return 0;
}
```

New size: 7

Resizing a "vector"

IIT Bombay

- push_back(), pop_back result in automatic resizing
- C++ allows explicit resizing of vectors

```
int main() {
  vector<int> intVec(5, 0);
   intVec.resize(10, -1);
   cout << "Size after resizing: " << intVe                    << endl;
   for (int i = 0; i < 10; i++) { cout << intV                  t(i) << " "; }
   cout << endl;
   intVec.resize(7);  cout << "New size: " << intVec.size() << endl;
   for (int i = 0; i < 7; i++) { cout << intVec.at(i) << " "; }
   return 0;
}
```

0 0 0 0 0 -1 -1

Summary

IIT Bombay

- "**vector**" class and its usage
  - Only some features studied
- We'll study some more in next lecture