Computer Programming

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

Session : Template Class “list”

Quick Recap of Relevant Topics

IIT Bombay

- Template classes and functions
- C++ Standard Library
  - The “string” class
  - The “vector” class
  - The “map” class

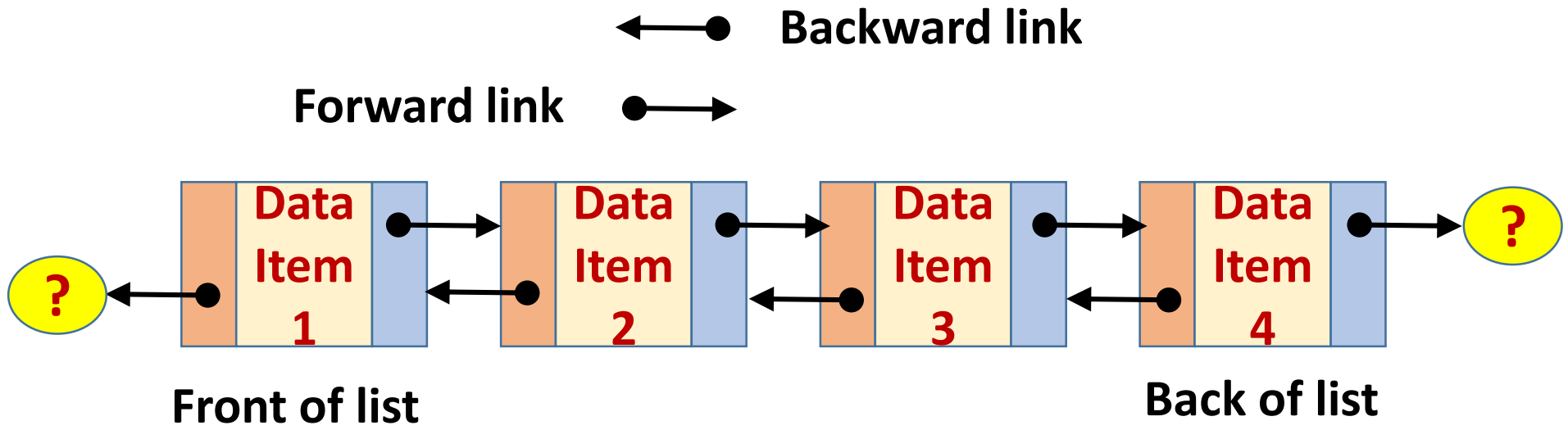
Overview of This Lecture

IIT Bombay

- The template class “list”

Doubly Linked List

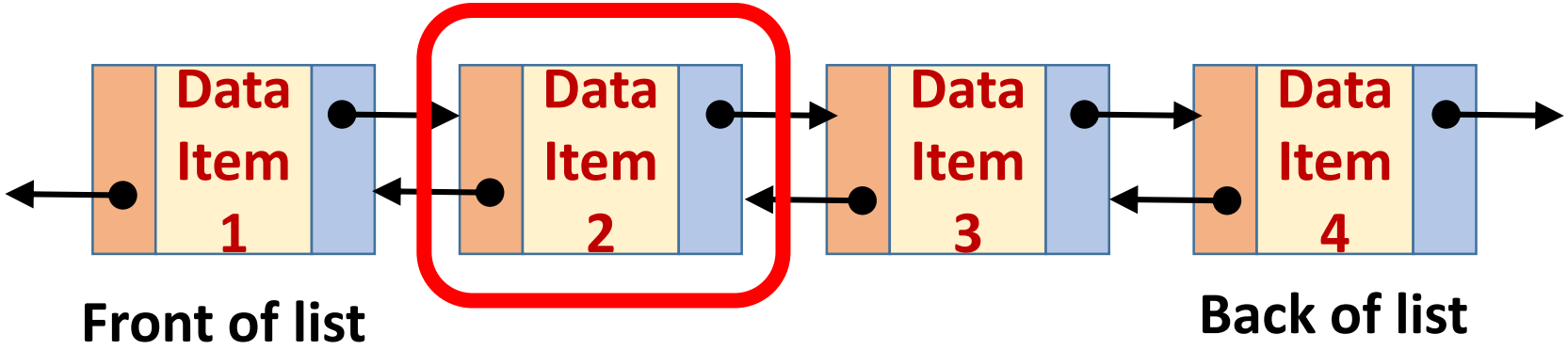
IIT Bombay



- Convenient way to represent dynamically created sequence of objects/data items
- Memory allocated for consecutive objects in list not necessarily contiguous, may not even be allocated at same time

Deletion in Doubly Linked List

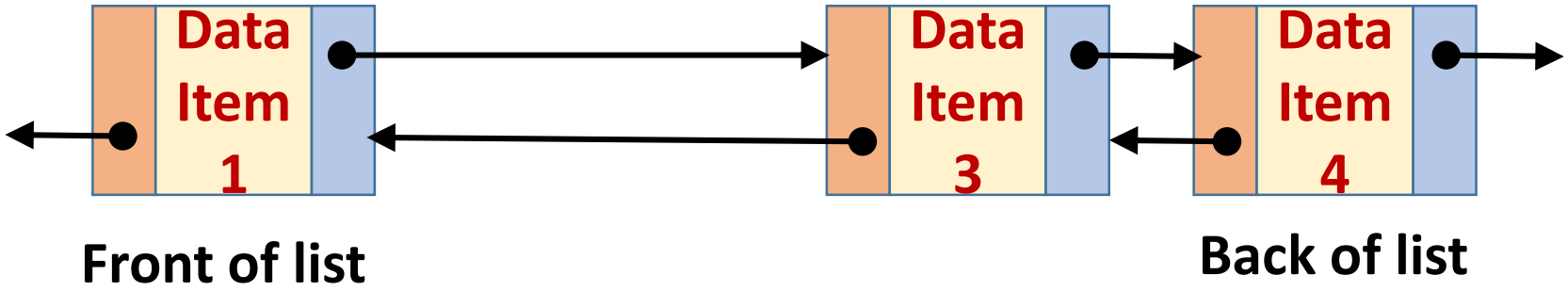
IIT Bombay



Efficient (constant time) deletion of objects

Deletion in Doubly Linked List

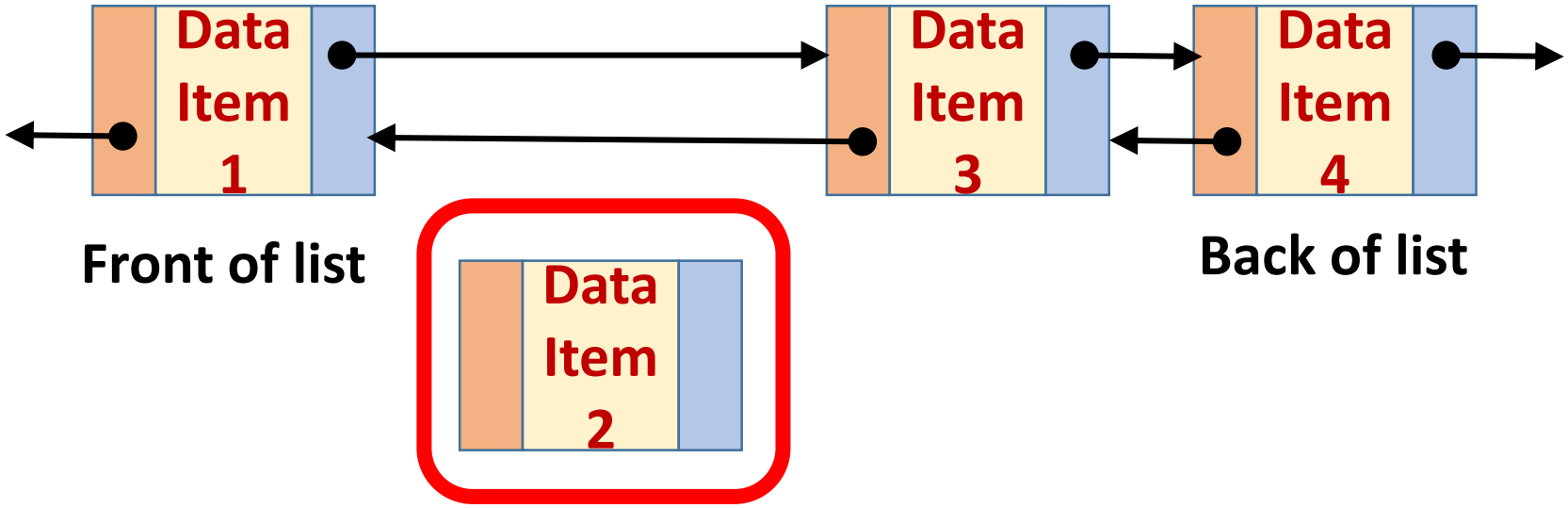
IIT Bombay



Efficient (constant time) deletion of objects

Insertion in Doubly Linked List

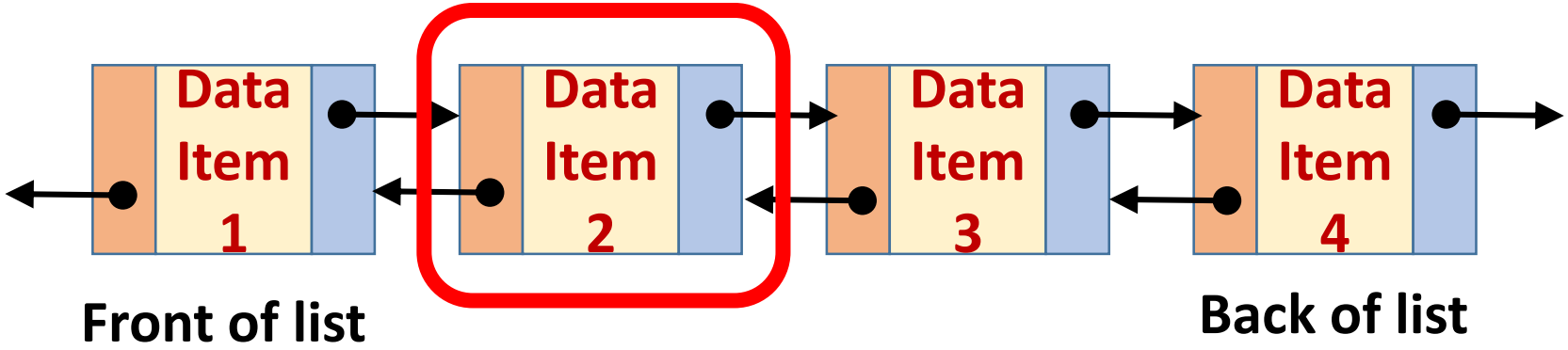
IIT Bombay



Efficient (constant time) insertion of objects

Insertion in Doubly Linked List

IIT Bombay

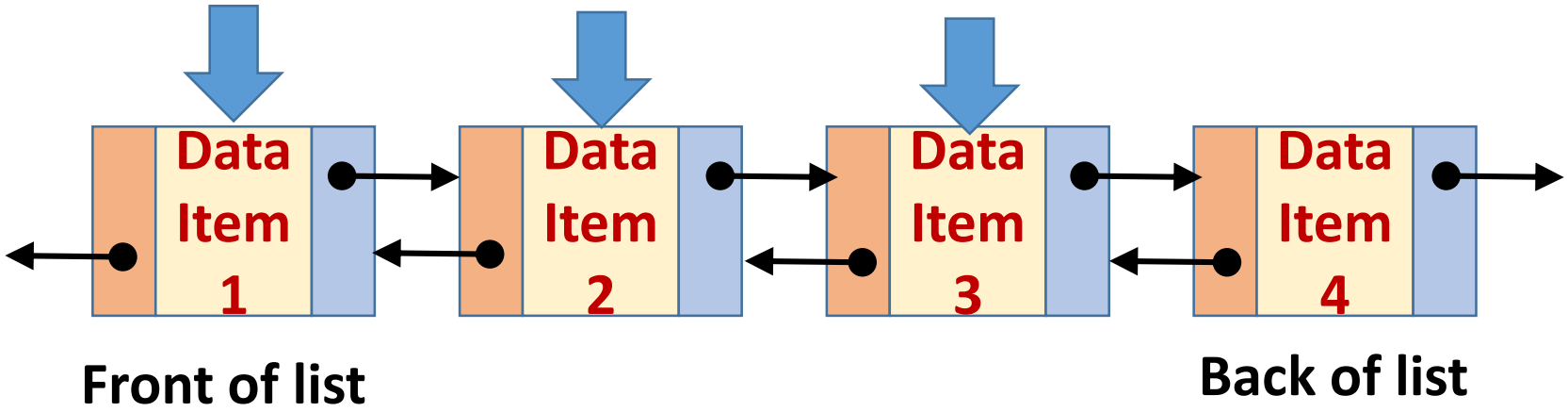


Efficient (constant time) insertion of objects

Accessing n^{th} Element in Doubly Linked List

IIT Bombay

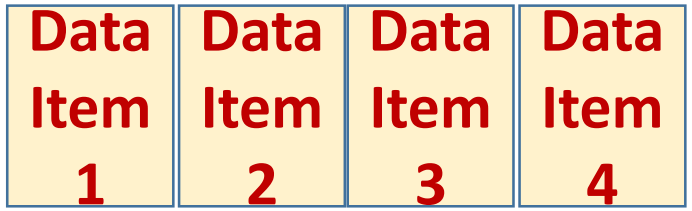
Example: Accessing 3rd element in list



Inefficient: Requires traversing list

Comparison with a Vector

IIT Bombay



- Contiguous memory locations for successive elements
- Insertion and deletion much more expensive

  Requires copying of data items/objects
- Accessing n^{th} element is efficient (constant time)

Choice of linked list vs vector depends on nature of program

E.g., Sorting: Linked list is a good choice

The “list” class

IIT Bombay

- For representing and manipulating **doubly linked lists**

  Template class : Can be instantiated with type of data item
  - Dynamically allocate/de-allocate memory
  - Dynamic memory management built in
- “**list**” objects are container objects
- Must use **#include <list>** at start of program
- Large collection of member functions
  - We’ll see only a small subset

Simple Programming using "list"

IIT Bombay

```
#include <iostream>
#include <list>
using namespace std;
int main() {
  list<string> names;
  list<string> books (3, "Alice in Wonderland");
  list<int> numbers (10, -1);
   … Some other code …
}
```

Creates an empty list of strings

Name of list

Simple Programming using "list"

IIT Bombay

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   list<string> names;
   list<string> books (3, "Alice in Wonderland");
   list<int> numbers (10, -1);
    … Some other code …
}
```

Creates a list of 3 strings.
Each string in the list is
"Alice in Wonderland"

Simple Programming using "list"

IIT Bombay

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   list<string> names;
   list<string> books (3, "Alice in Wonderland");
   list<int> numbers (10, -1);
    … Some other code …
}
```

Creates a list of 10 integers.
Each integer in the list is -1

Finding the Size of a "list"

IIT Bombay

```
#include <iostream>
#include <list>
using namespace std;
int main() {
   list<string> names;
   list<string> books (3, "Alice in Wonderland");
   list<int> numbers (10, -1);
   cout << "Sizes: ";
   cout << names.size() << " " << books.size()  << " " << numbers.size();
   cout << endl;
   return 0;
}
```

Sizes: 0 3 10

Adding Elements at Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  … Some other code …
}
```

names:

Adding Elements at Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  … Some other code …
}
```

names:
"Abdul"

Adding Elements at Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  … Some other code …
}
```

names:
"Ajanta" "Abdul"

Adding Elements at Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  … Some other code …
}
```

names:
"Ajanta" "Abdul" "Bobby"

Adding Elements at Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  … Some other code …
}
```

names:
"Alex" "Ajanta" "Abdul" "Bobby"

Removing Elements From Front/Back of a "list"


```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  names.pop_back();
  … Some other code …
}
```

names:
"Alex" "Ajanta" "Abdul" "Bobby"

Removing Elements From Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobb");
  names.push_front("Al");
  names.pop_back();
  names.pop_front();     names.pop_front();
  … Some other code …
}
```

names:
"Alex" "Ajanta" "Abdul"

Removing Elements From Front/Back of a "list"

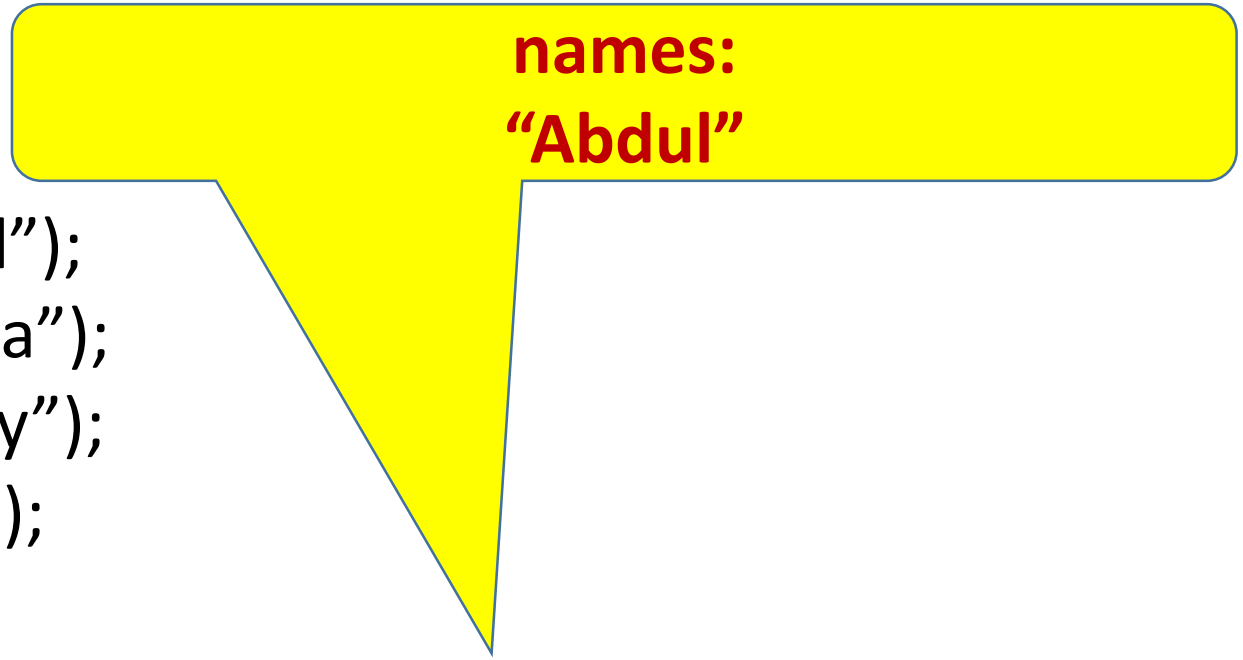

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  names.pop_back();
  names.pop_front();     names.pop_front();
  … Some other code …
}
```

names:
"Ajanta" "Abdul"

Removing Elements From Front/Back of a "list"

IIT Bombay

```
int main() {
  list<string> names;
  names.push_back("Abdul");
  names.push_front("Ajanta");
  names.push_back("Bobby");
  names.push_front("Alex");
  names.pop_back();
  names.pop_front();
  … Some other code …
}
```



Iterator Related Functions in "list" Class

IIT Bombay

```
int main() {
    list<string>  names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it;
    for (it = names.begin(); it != names.end(); it++)  {
        cout << *it << ", ";
    }
    return 0;
}
```

begin(), end()
member functions

Bobby, Ajanta, Abdul,

Iterator Related Functions in "list" Class

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::reverse_iterator rit;
    for (rit = names.rbegin(); rit != names.rend(); rit++)  {
        cout << *rit << ", ";
    }
    return 0;
}
```

rbegin(), rend()
member functions

Abdul, Ajanta, Bobby,

Inserting in a "list" using Iterator

IIT Bombay

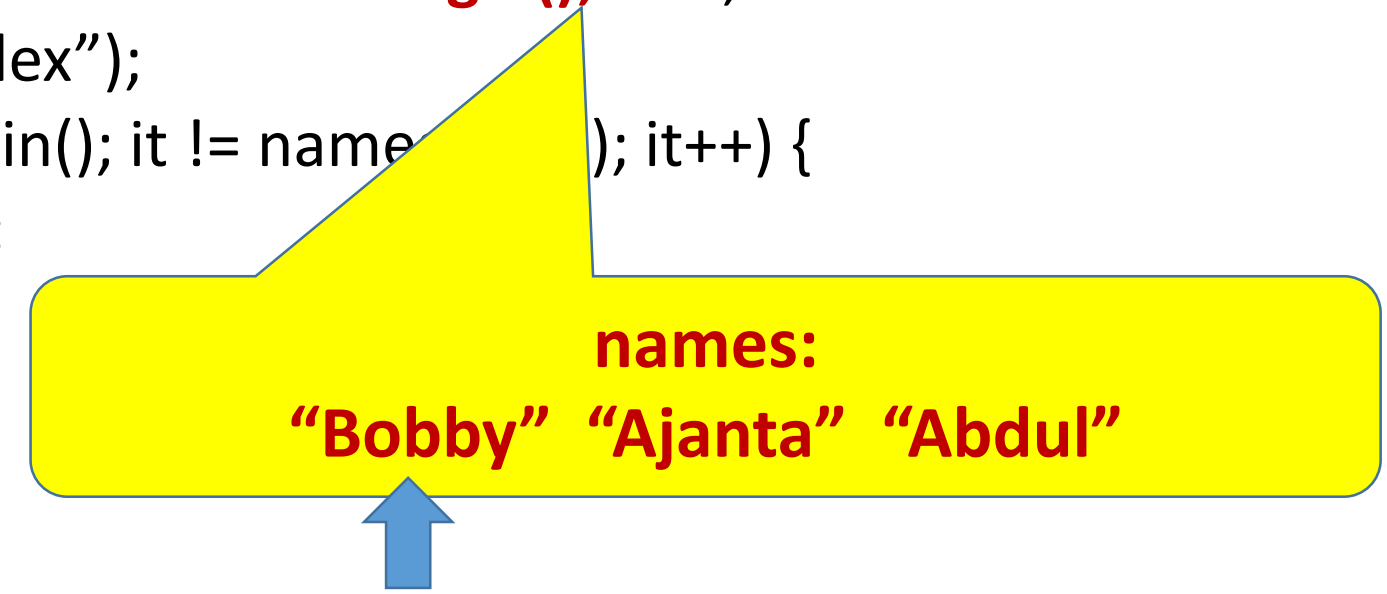
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

names:
"Bobby" "Ajanta" "Abdul"

Inserting in a "list" using Iterator

IIT Bombay

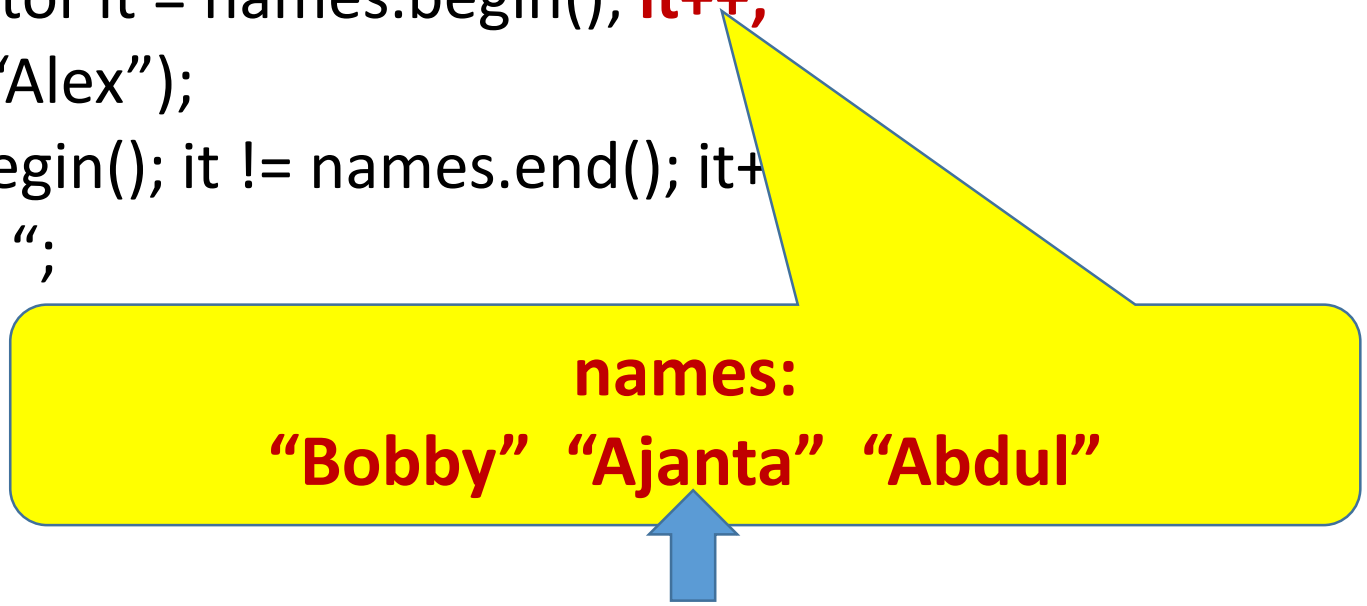
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");
    for (it = names.begin(); it != name          ); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

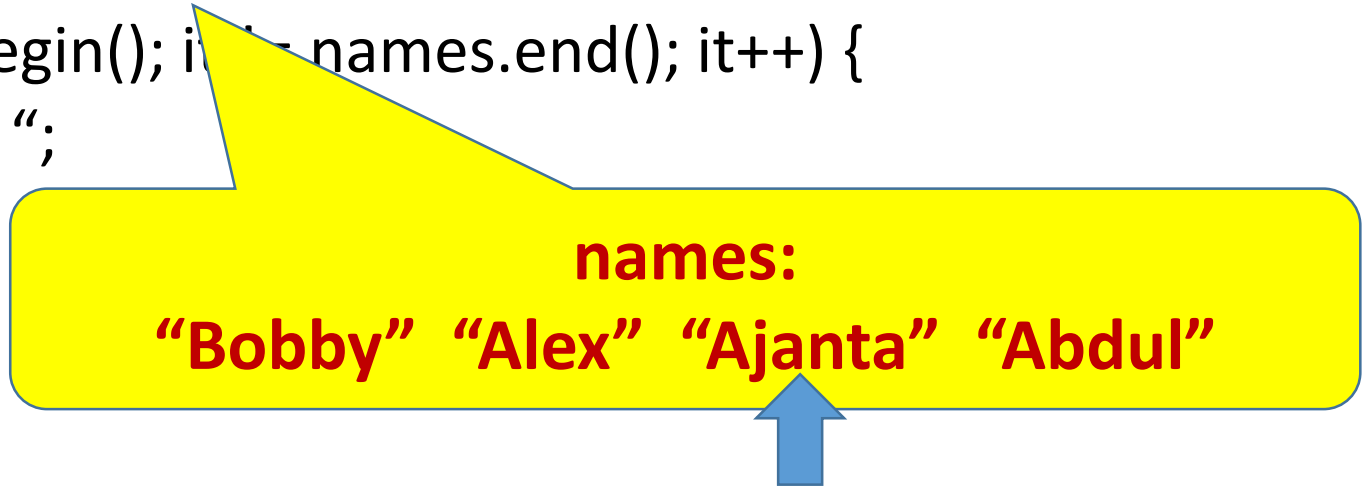
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");
    for (it = names.begin(); it != names.end(); it+
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

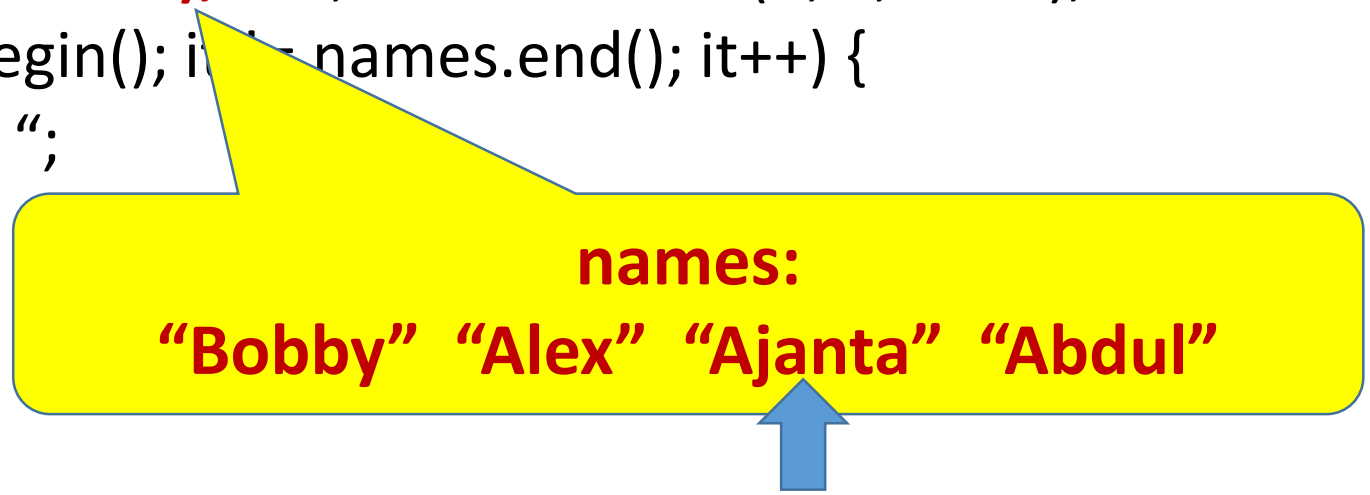
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

Bobby, Alex, Ajanta, Abdul,

Inserting in a "list" using Iterator

IIT Bombay

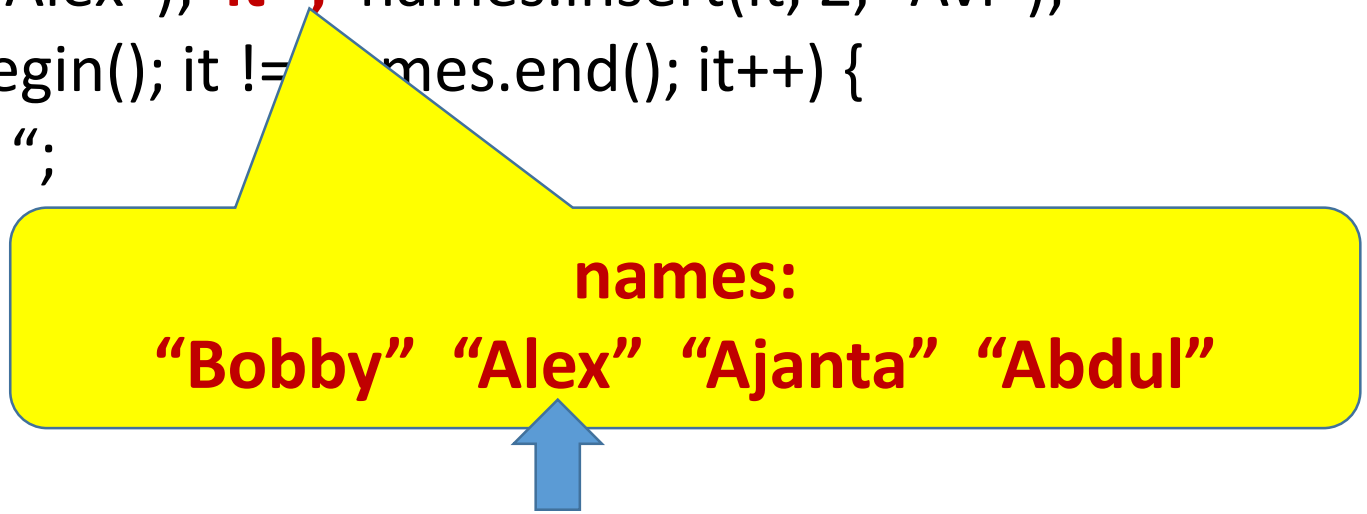
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");  it--;  names.insert(it, 2, "Avi");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

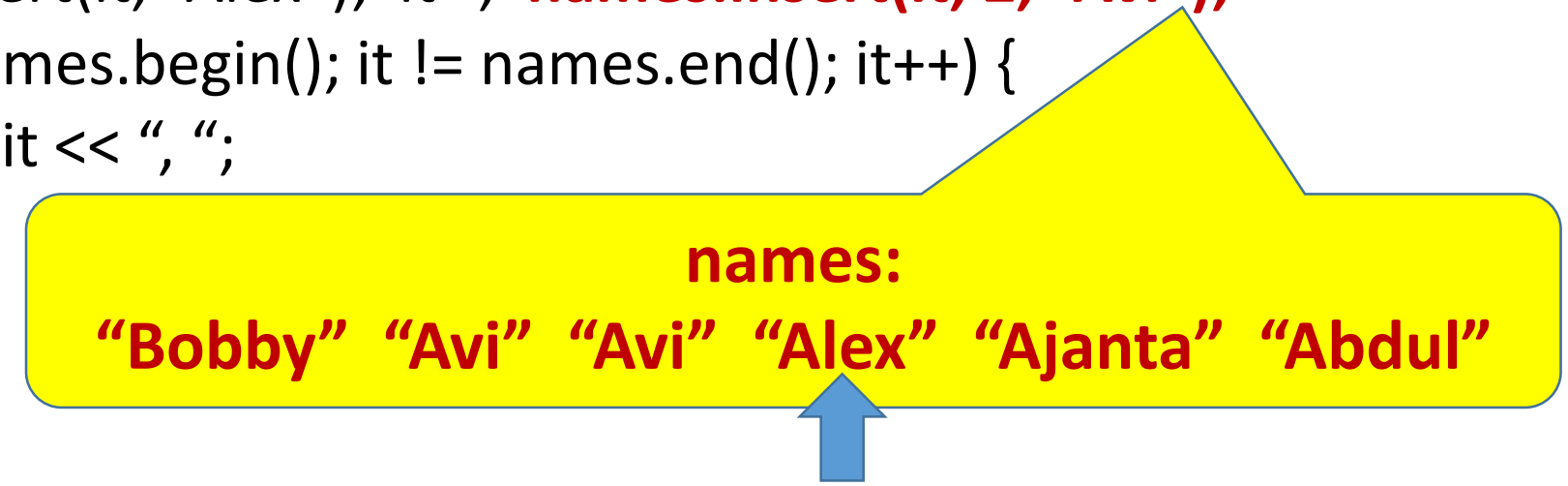
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");  it--;  names.insert(it, 2, "Avi");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");  it--;  names.insert(it, 2, "Avi");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```



Inserting in a "list" using Iterator

IIT Bombay

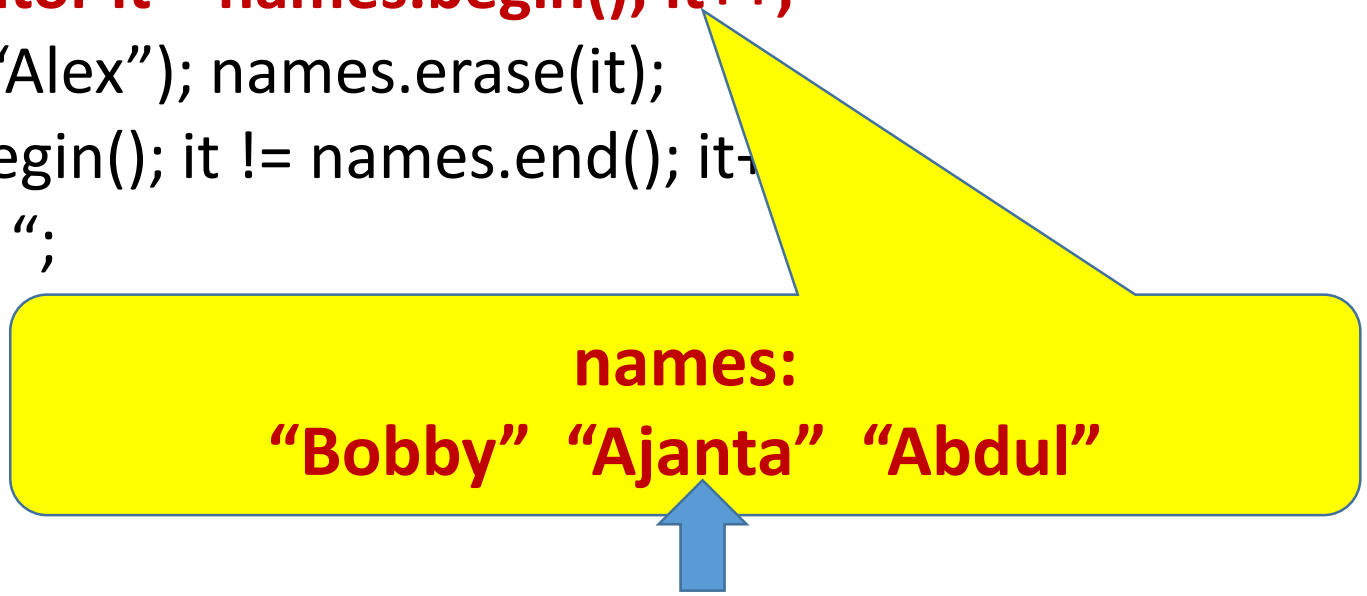
```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex");  it--;  names.insert(it, 2, "Avi");
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

Bobby, Avi, Avi, Alex, Ajanta, Abdul,

Removing from a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    for (it = names.begin(); it != names.end(); it+
      cout << *it << ", ";
    }
    return 0;
}
```



Removing from a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

names:
"Bobby" "Alex" "Ajanta" "Abdul"

Removing from a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

names:
"Bobby" "Alex" "Abdul"

Removing from a "list" using Iterator

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    for (it = names.begin(); it != names.end(); it++) {
      cout << *it << ", ";
    }
    return 0;
}
```

Bobby, Alex, Abdul,

Accessing the Front and Back Elements in a "list"


```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    cout << "Front element is: " << names.fron
    cout << "Back element is: " << names.back
    return 0;
}
```

names:
"Bobby" "Alex" "Abdul"

Accessing the Front and Back Elements in a "list"


```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it = names.begin(); it++;
    names.insert(it, "Alex"); names.erase(it);
    cout << "Front element is: " << names.front() << endl;
    cout << "Back element is: " << names.back() << endl;
    return 0;
}
```

Front element is: Bobby
Back element is: Abdul

Reversing a "list"

IIT Bombay

```
int main() {
   list<string> names;
   names.push_front("Abdul");
   names.push_front("Ajanta");  names.push_front("Bobby");
   list<string>::iterator it;
   for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
   cout << endl;
   names.reverse();
   for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
   cout << endl; return 0;
}
```

names:
"Bobby" "Ajanta" "Abdul"

Reversing a "list"

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it;
    for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
    cout << endl;
    names.reverse();
    for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
    cout << endl; return 0;
}
```

Bobby, Ajanta, Abdul,

Reversing a "list"

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  n        push_front("Bobby");
    list<string>::iterator it;
    for (it = names.begin(); i      names.end(); it++) { cout << *it << ", "; }
    cout << endl;
    names.reverse();
    for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
    cout << endl; return 0;
}
```

names:
"Abdul" "Ajanta" "Bobby"

Reversing a "list"

IIT Bombay

```
int main() {
    list<string> names;
    names.push_front("Abdul");
    names.push_front("Ajanta");  names.push_front("Bobby");
    list<string>::iterator it;
    for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
    cout << endl;
    names.reverse();
    for (it = names.begin(); it != names.end(); it++) { cout << *it << ", "; }
    cout << endl; return 0;
}
```

Abdul, Ajanta, Bobby,

Lists of Complex Data Types

IIT Bombay

- "**list**" is a template class

  Can be instantiated with any data type

  Can even have lists of lists of lists …

```
list<V3> myList1;
list<list<int *> > myList2;
```

Note the space

Summary

IIT Bombay

- "**list**" class and its usage
  - Only some features studied
- Several more features exist …