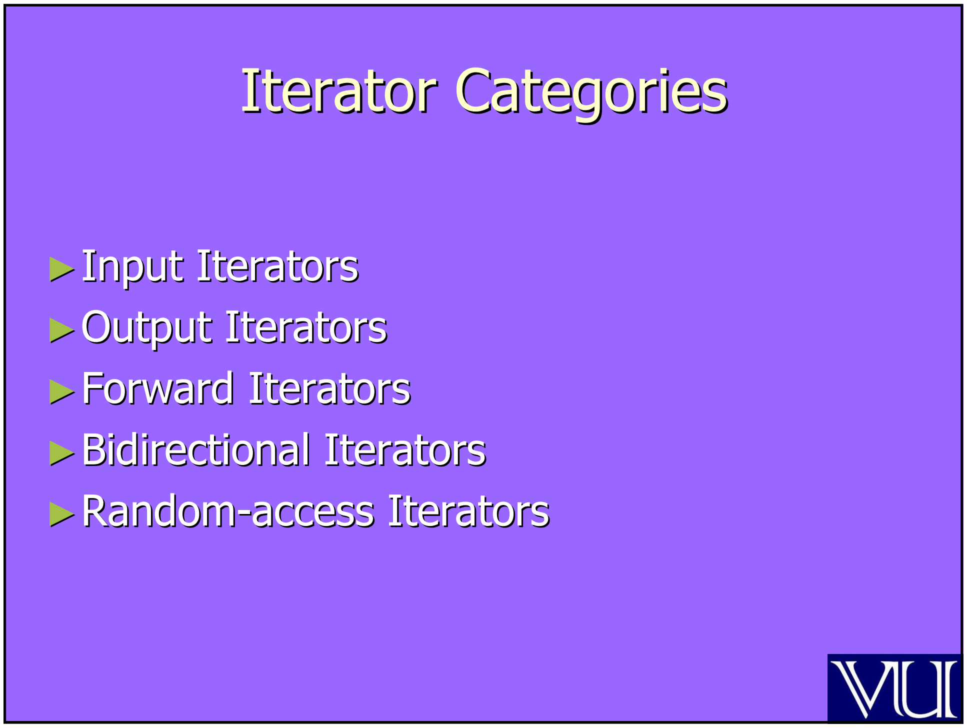
Object-Oriented Programming (OOP)
Lecture No. 42

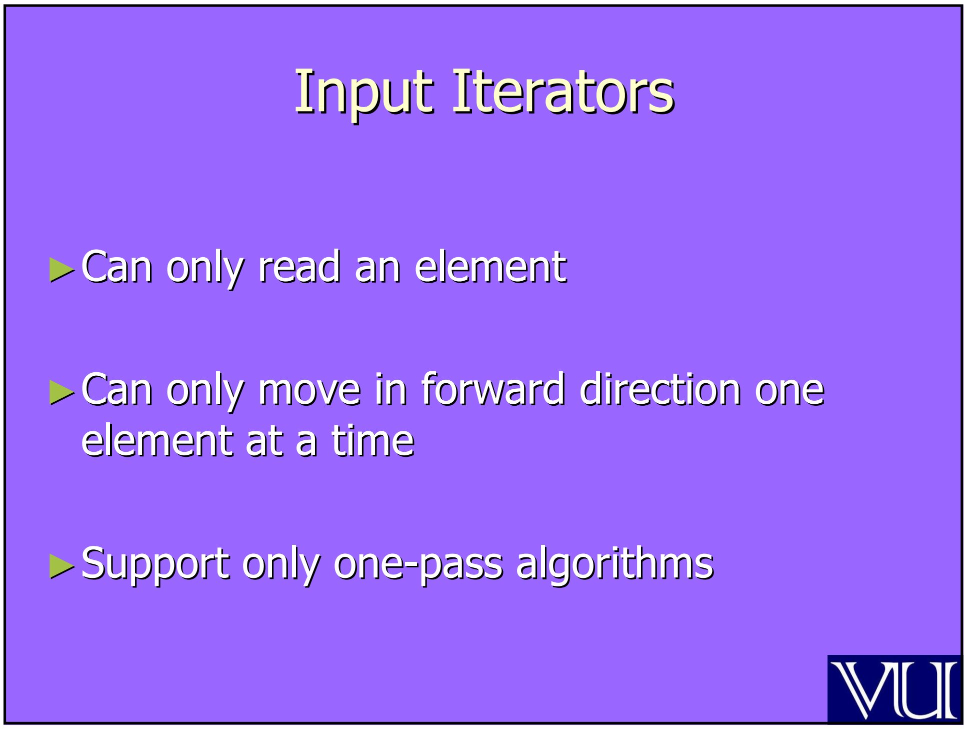
Iterators

- Iterators are types defined by STL
- Iterators are for containers like pointers are for ordinary data structures
- STL iterators provide pointer operations such as * and ++



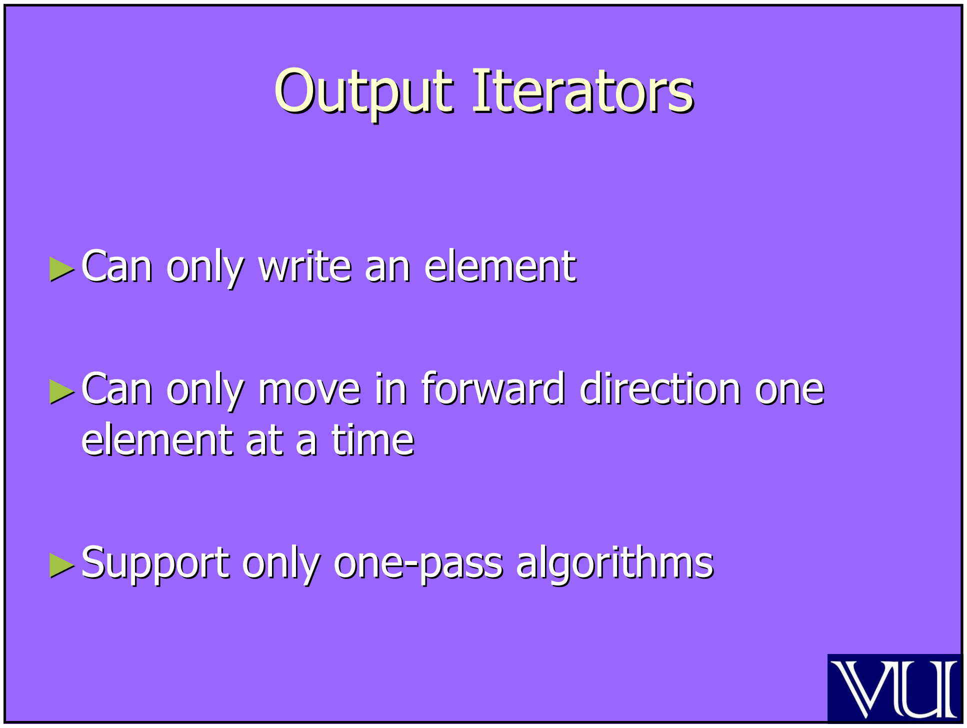
Iterator Categories

- Input Iterators
- Output Iterators
- Forward Iterators
- Bidirectional Iterators
- Random-access Iterators



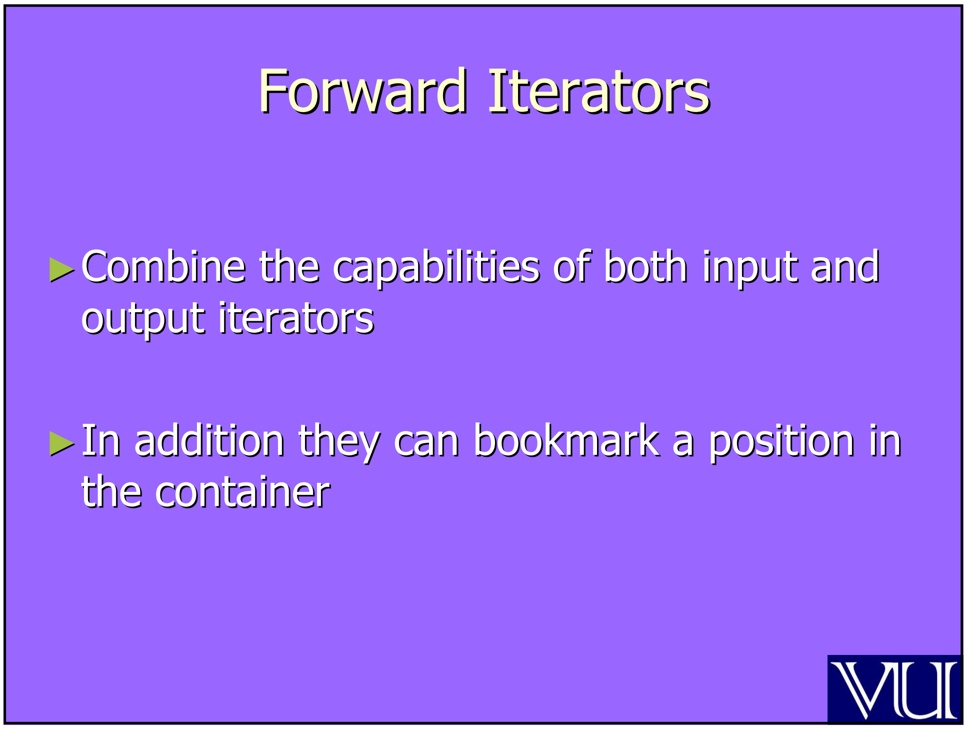
Input Iterators

- Can only read an element
- Can only move in forward direction one element at a time
- Support only one-pass algorithms



Output Iterators

- Can only write an element
- Can only move in forward direction one element at a time
- Support only one-pass algorithms



Forward Iterators

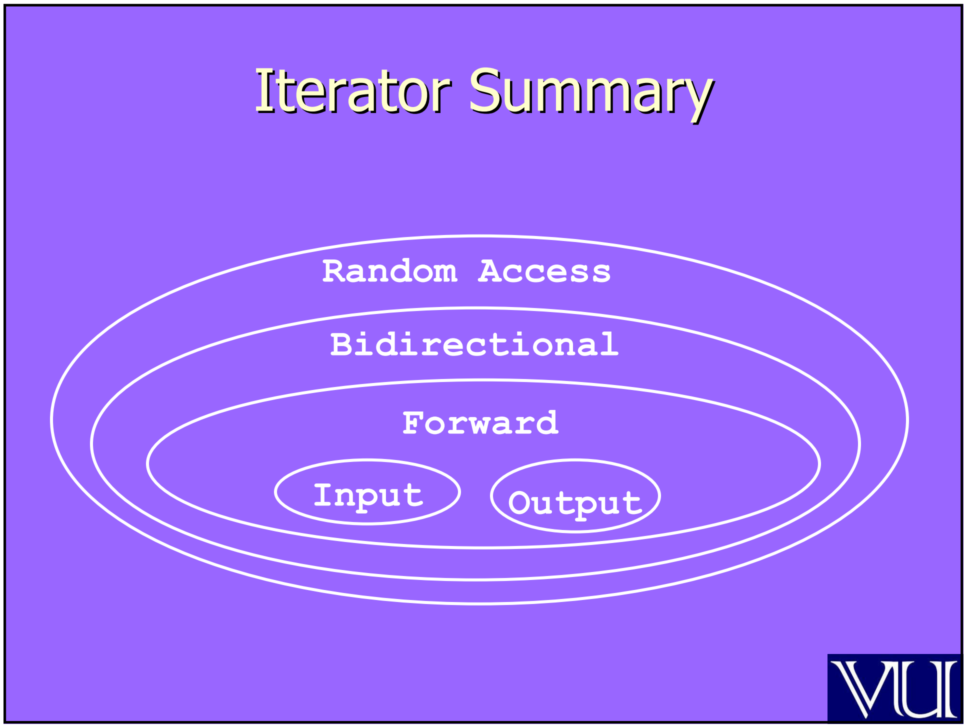
- Combine the capabilities of both input and output iterators
- In addition they can bookmark a position in the container

Bidirectional Iterators

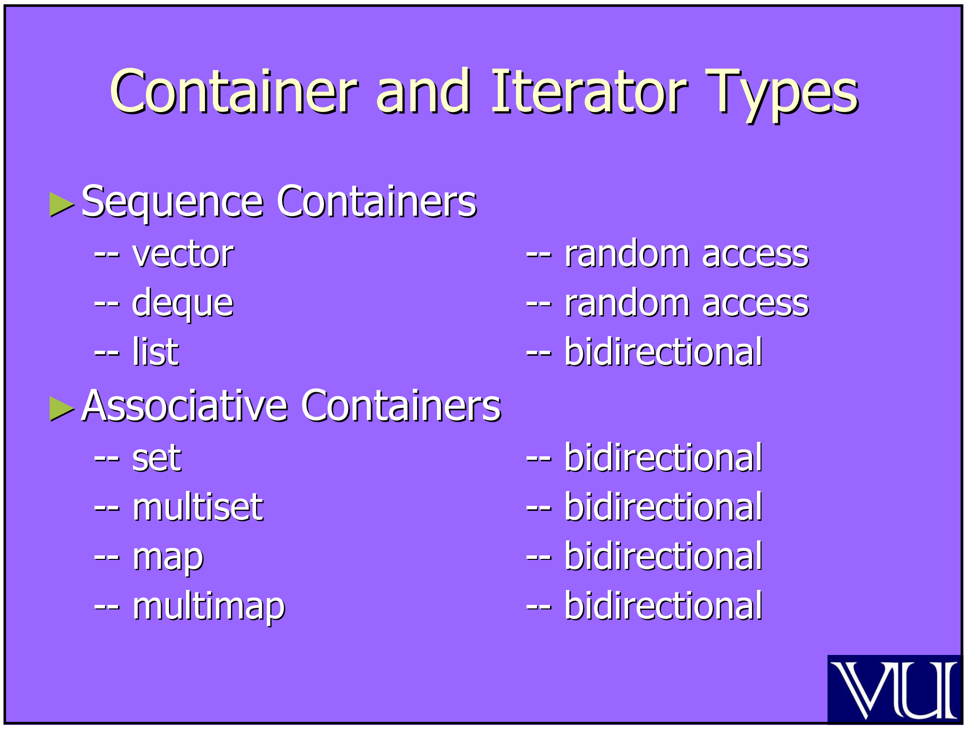
- Provide all the capabilities of forward iterators
- In addition, they can move in backward direction
- As a result they support multi-pass algorithms

Random Access Iterators

- Provide all the capabilities of bidirectional iterators
- In addition they can directly access any element of a container

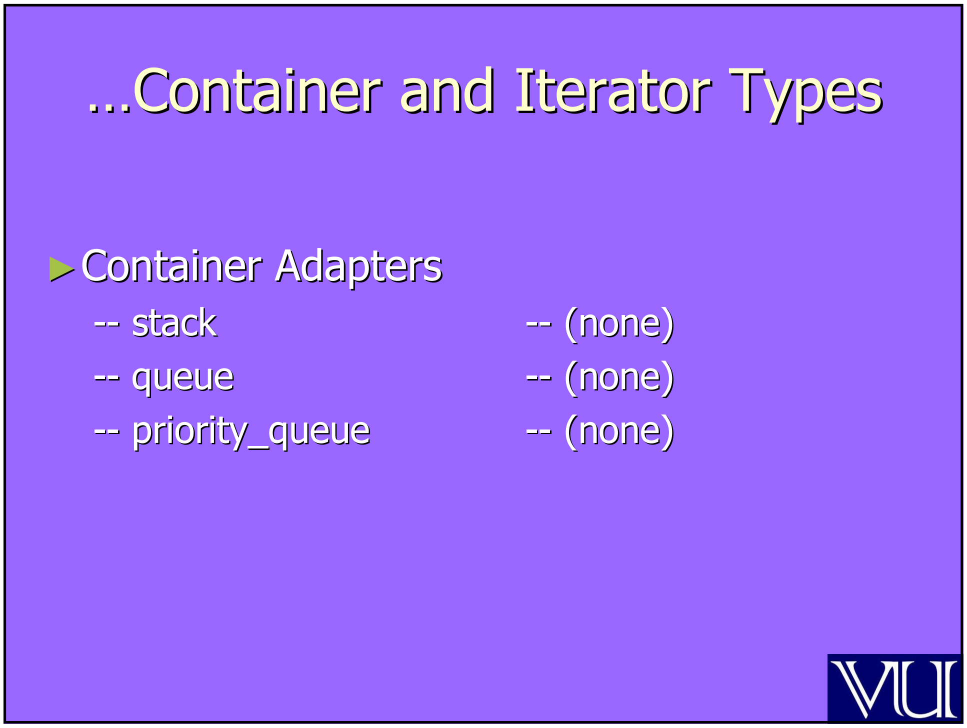


Iterator Summary



Container and Iterator Types

▶ Sequence Containers	
-- vector	-- random access
-- deque	-- random access
-- list	-- bidirectional
▶ Associative Containers	
-- set	-- bidirectional
-- multiset	-- bidirectional
-- map	-- bidirectional
-- multimap	-- bidirectional

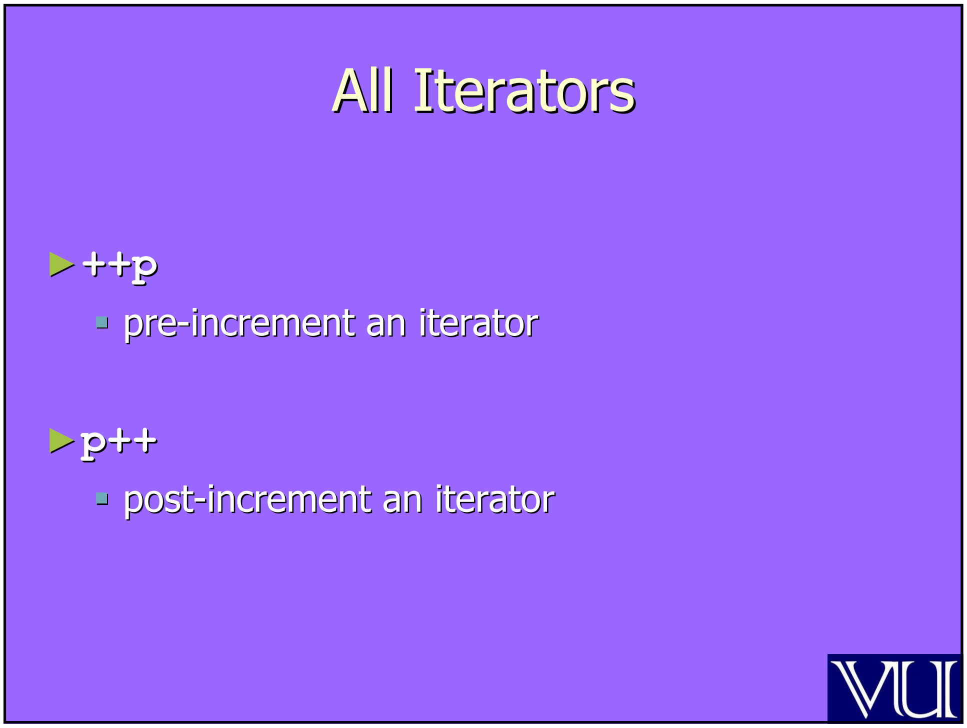


…Container and Iterator Types

- Container Adapters

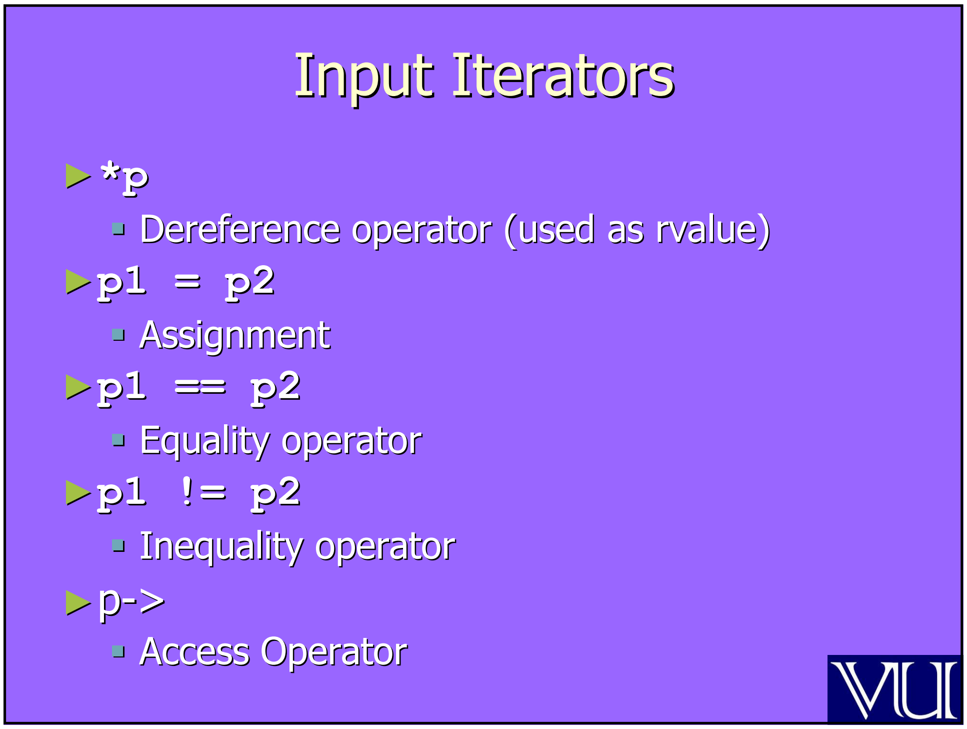
-- stack	-- (none)
-- queue	-- (none)
-- priority_queue	-- (none)

Iterator Operations



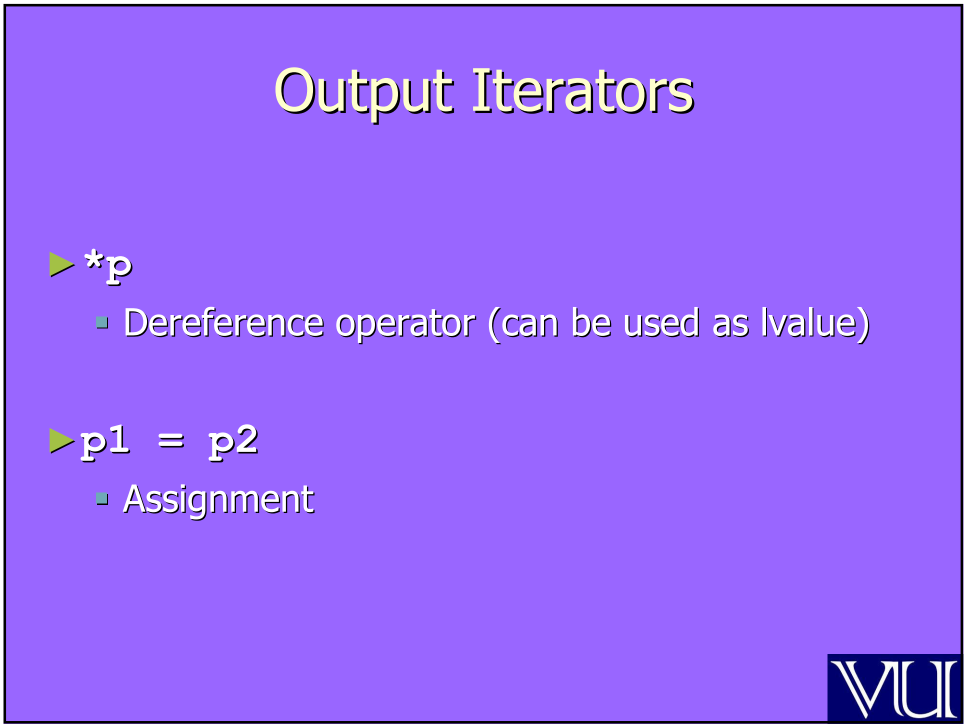
All Iterators

- `++p`
  - pre-increment an iterator
- `p++`
  - post-increment an iterator



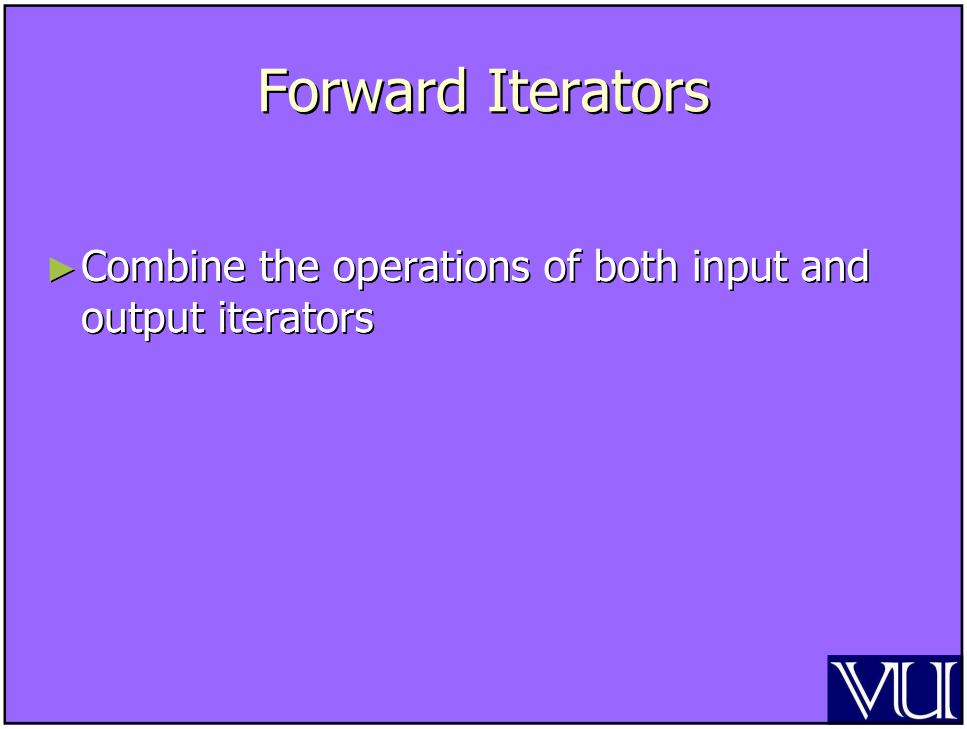
Input Iterators

- `*p`
  - Dereference operator (used as rvalue)
- `p1 = p2`
  - Assignment
- `p1 == p2`
  - Equality operator
- `p1 != p2`
  - Inequality operator
- p->
  - Access Operator



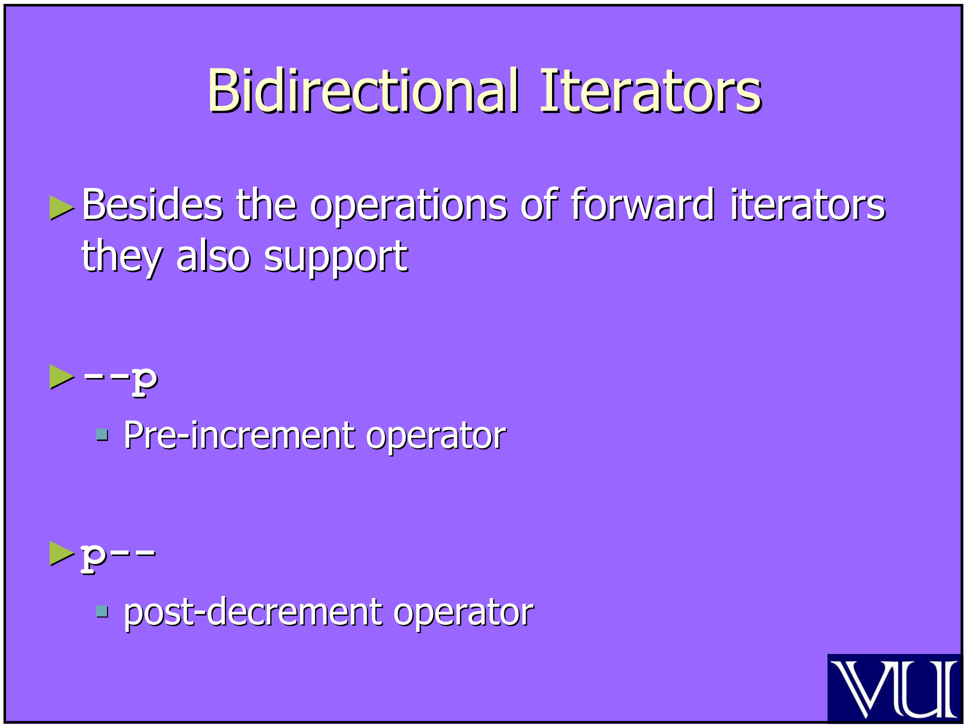
Output Iterators

- `*p`
  - Dereference operator (can be used as lvalue)
- `p1 = p2`
  - Assignment



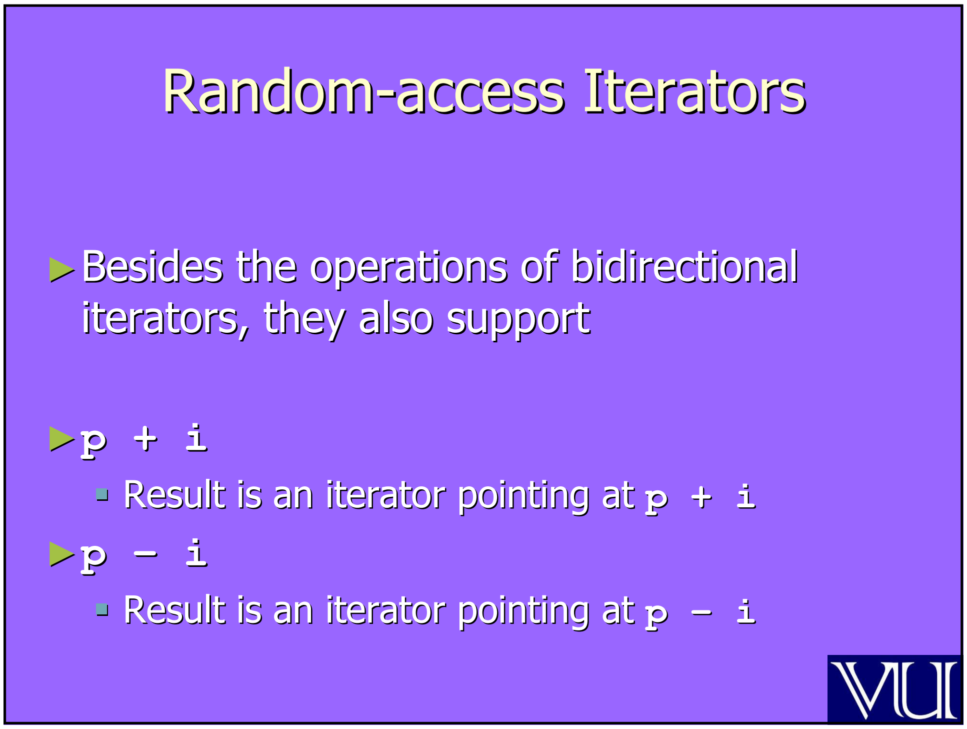
Forward Iterators

- Combine the operations of both input and output iterators



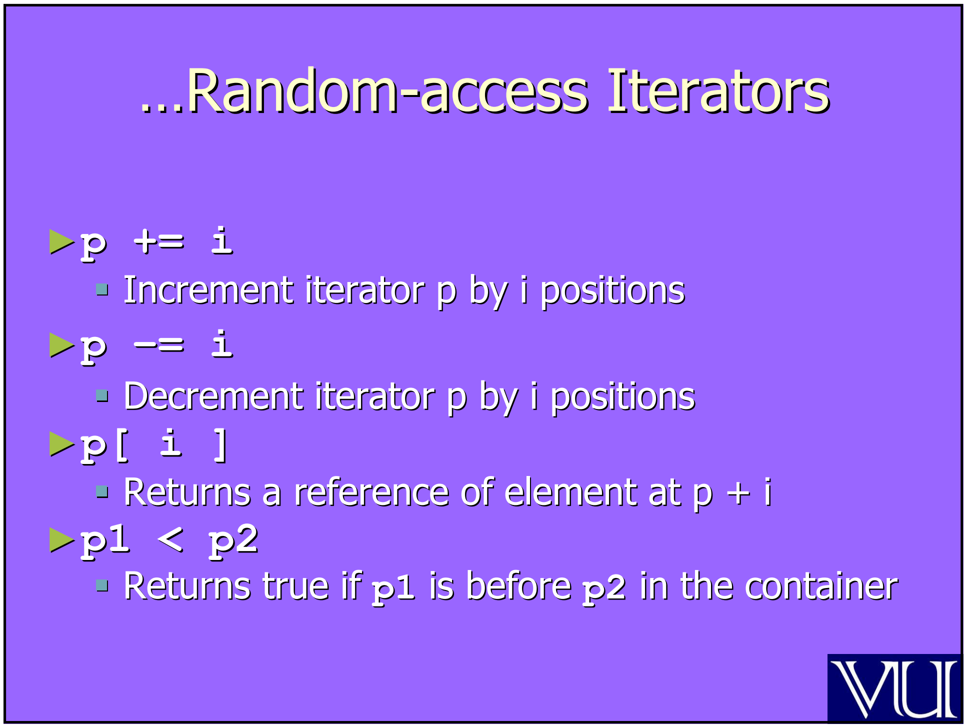
Bidirectional Iterators

- Besides the operations of forward iterators they also support
- **`--p`**
  - Pre-increment operator
- **`p--`**
  - post-decrement operator



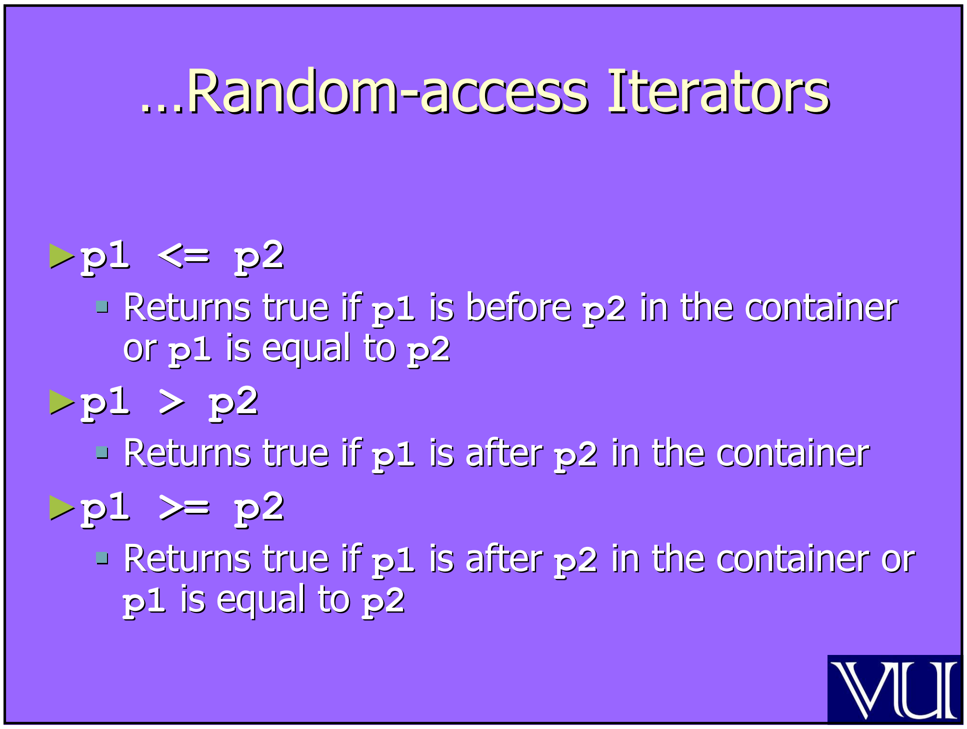
Random-access Iterators

- Besides the operations of bidirectional iterators, they also support
- **`p + i`**
  - Result is an iterator pointing at **`p + i`**
- **`p – i`**
  - Result is an iterator pointing at **`p – i`**



…Random-access Iterators

- **`p += i`**
  - Increment iterator p by i positions
- **`p -= i`**
  - Decrement iterator p by i positions
- **`p[ i ]`**
  - Returns a reference of element at p + i
- **`p1 < p2`**
  - Returns true if **`p1`** is before **`p2`** in the container



…Random-access Iterators

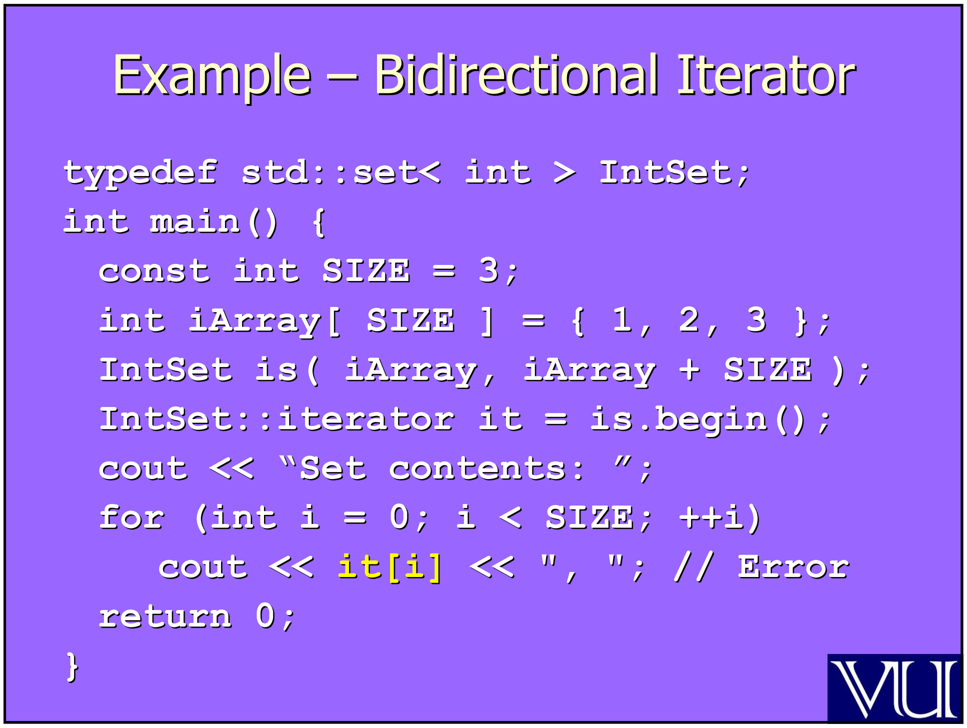
- **`p1 <= p2`**
  - Returns true if **`p1`** is before **`p2`** in the container or **`p1`** is equal to **`p2`**
- **`p1 > p2`**
  - Returns true if **`p1`** is after **`p2`** in the container
- **`p1 >= p2`**
  - Returns true if **`p1`** is after **`p2`** in the container or **`p1`** is equal to **`p2`**

Example – Random Access Iterator

```
typedef std::vector< int > IntVector;
int main() {
  const int SIZE = 3;
  int iArray[ SIZE ] = { 1, 2, 3 };
  IntVector iv(iArray, iArray + SIZE);
  IntVector::iterator it = iv.begin();
  cout << “Vector contents: ”;
  for ( int i = 0; i < SIZE; ++i )
     cout << it[i] << ", ";
  return 0;
}
```

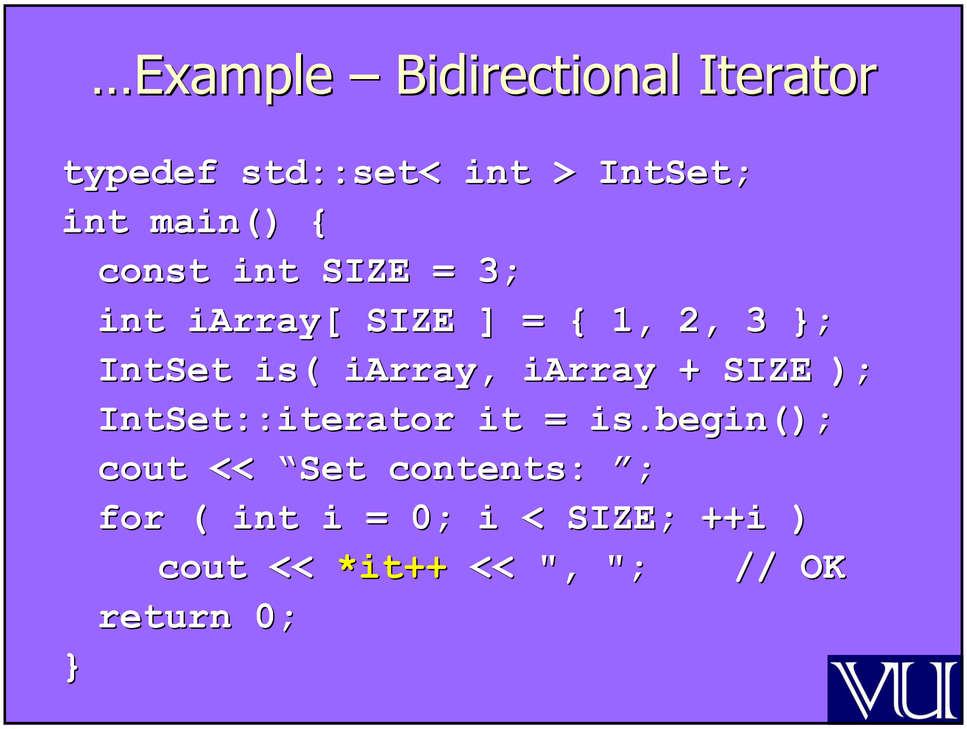
…Sample Output

```
Vector contents: 1, 2, 3,
```



Example – Bidirectional Iterator

```
typedef std::set< int > IntSet;
int main() {
  const int SIZE = 3;
  int iArray[ SIZE ] = { 1, 2, 3 };
  IntSet is( iArray, iArray + SIZE );
  IntSet::iterator it = is.begin();
  cout << “Set contents: ”;
  for (int i = 0; i < SIZE; ++i)
    cout << it[i] << ", "; // Error
  return 0;
}
```



…Example – Bidirectional Iterator

```
typedef std::set< int > IntSet;
int main() {
  const int SIZE = 3;
  int iArray[ SIZE ] = { 1, 2, 3 };
  IntSet is( iArray, iArray + SIZE );
  IntSet::iterator it = is.begin();
  cout << “Set contents: ”;
  for ( int i = 0; i < SIZE; ++i )
    cout << *it++ << ", ";    // OK
  return 0;
}
```

…Sample Output

```
Set contents: 1, 2, 3,
```

…Example – Bidirectional Iterator

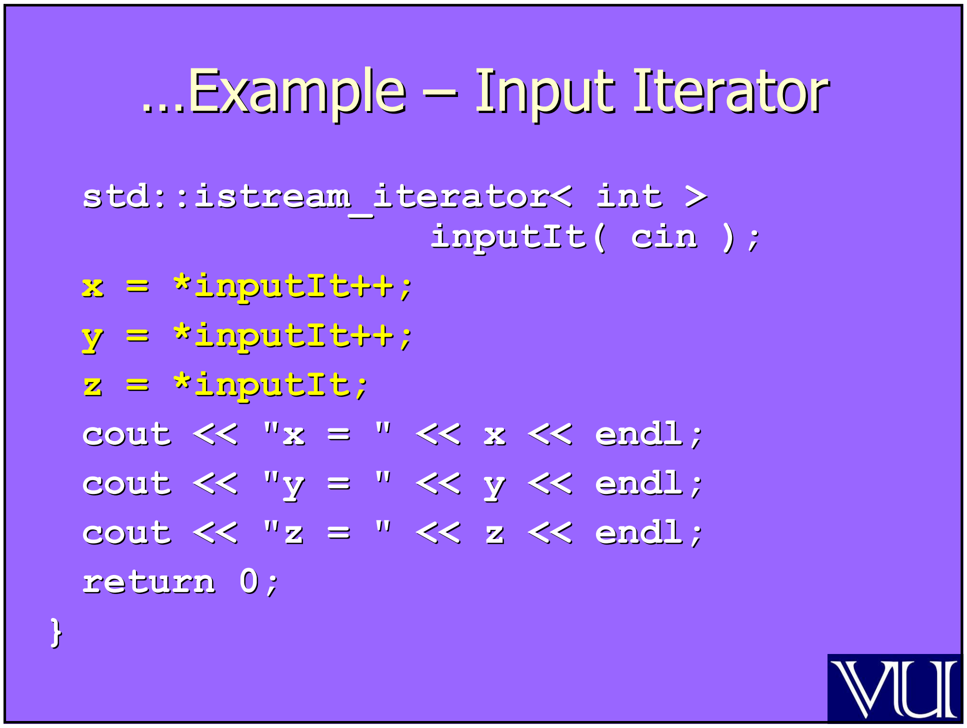
```
typedef std::set< int > IntSet;
int main() {
  const int SIZE = 3;
  int iArray[ SIZE ] = { 1, 2, 3 };
  IntSet is( iArray, iArray + SIZE );
  IntSet::iterator it = is.end();
  cout << “Set contents: ”;
  for (int i = 0; i < SIZE; ++i)
     cout << *--it << ", ";
  return 0;
}
```

…Sample Output

```
Set contents: 3, 2, 1,
```

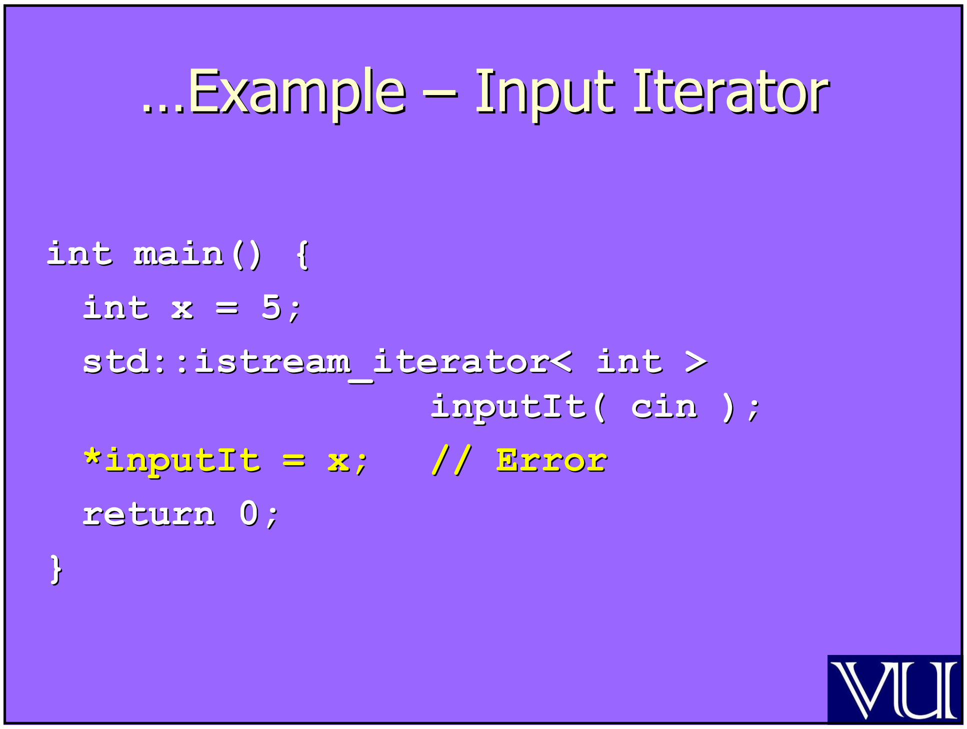
Example – Input Iterator

```
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <iterator>

int main() {
  int x, y, z;
  cout << "Enter three integers:\n";
```



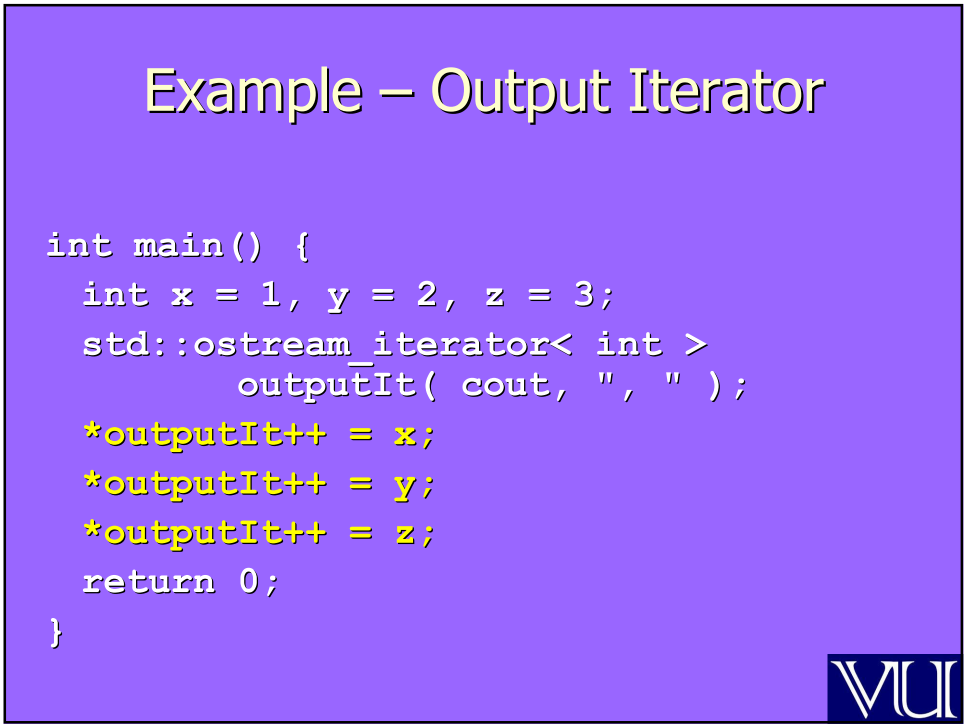
…Example – Input Iterator

```
  std::istream_iterator< int >
                     inputIt( cin );
  x = *inputIt++;
  y = *inputIt++;
  z = *inputIt;
  cout << "x = " << x << endl;
  cout << "y = " << y << endl;
  cout << "z = " << z << endl;
  return 0;
}
```



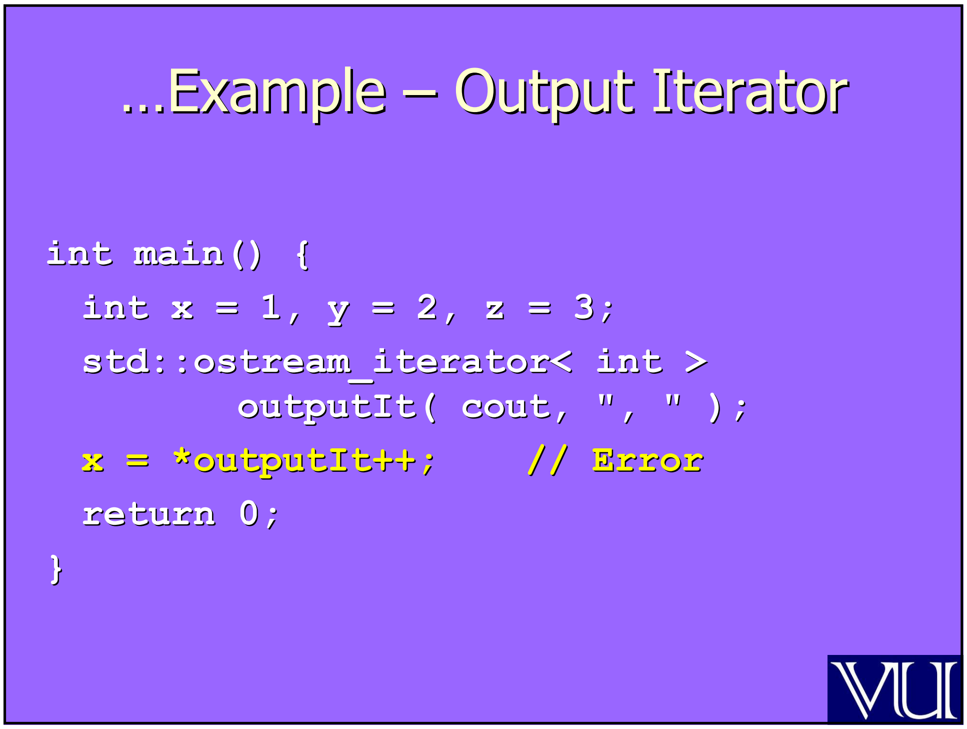
…Example – Input Iterator

```
int main() {
  int x = 5;
  std::istream_iterator< int >
                     inputIt( cin );
  *inputIt = x;   // Error
  return 0;
}
```



Example – Output Iterator

```
int main() {
  int x = 1, y = 2, z = 3;
  std::ostream_iterator< int >
          outputIt( cout, ", " );
  *outputIt++ = x;
  *outputIt++ = y;
  *outputIt++ = z;
  return 0;
}
```



…Example – Output Iterator

```
int main() {
  int x = 1, y = 2, z = 3;
  std::ostream_iterator< int >
          outputIt( cout, ", " );
  x = *outputIt++;    // Error
  return 0;
}
```

Algorithms

- STL includes 70 standard algorithms
- These algorithms may use iterators to manipulate containers
- STL algorithms also work for ordinary pointers and data structures

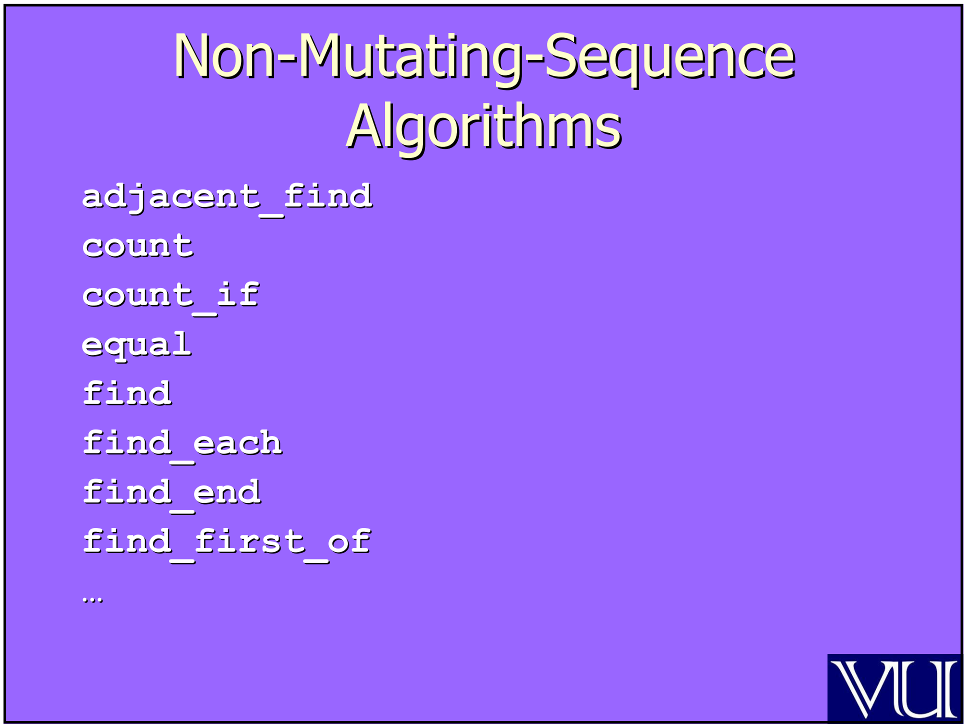
…Algorithms

- An algorithm works with a particular container only if that container supports a particular iterator category
- A multi-pass algorithm for example, requires bidirectional iterator(s) at least

Examples

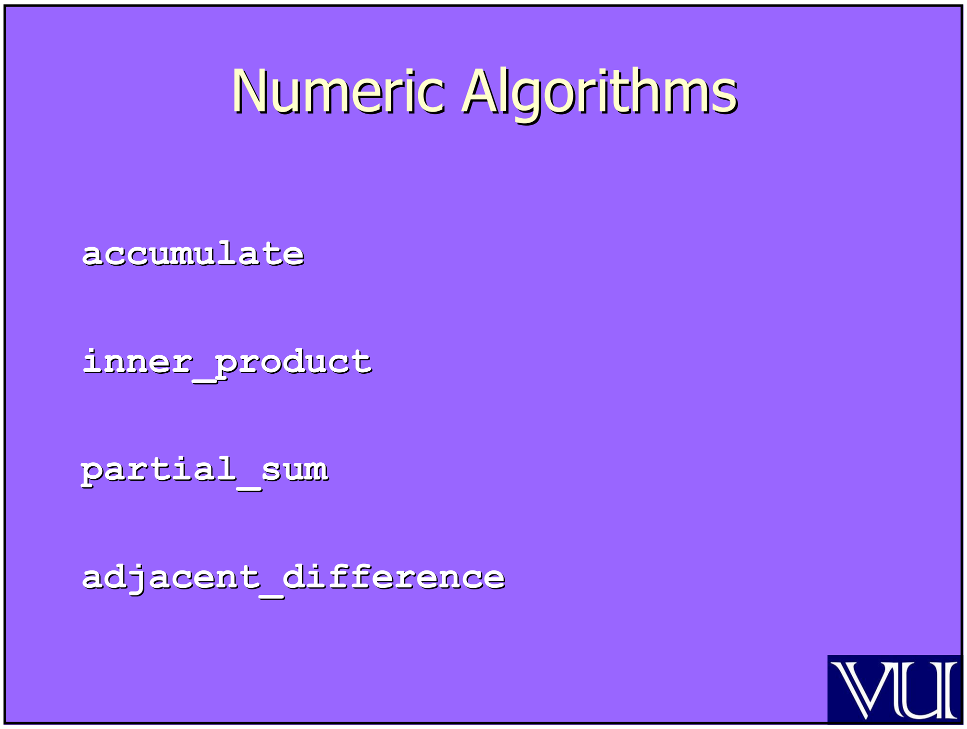
Mutating-Sequence Algorithms

```
copy
copy_backward
fill
fill_n
generate
generate_n
iter_swap
partition
…
```



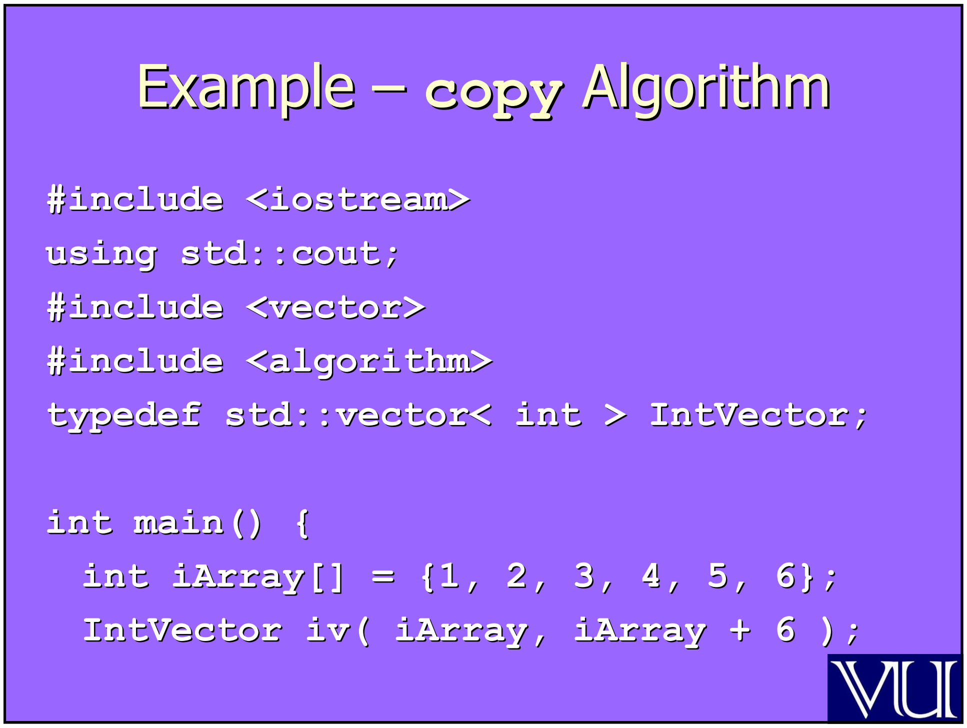
Non-Mutating-Sequence Algorithms

```
adjacent_find
count
count_if
equal
find
find_each
find_end
find_first_of
…
```



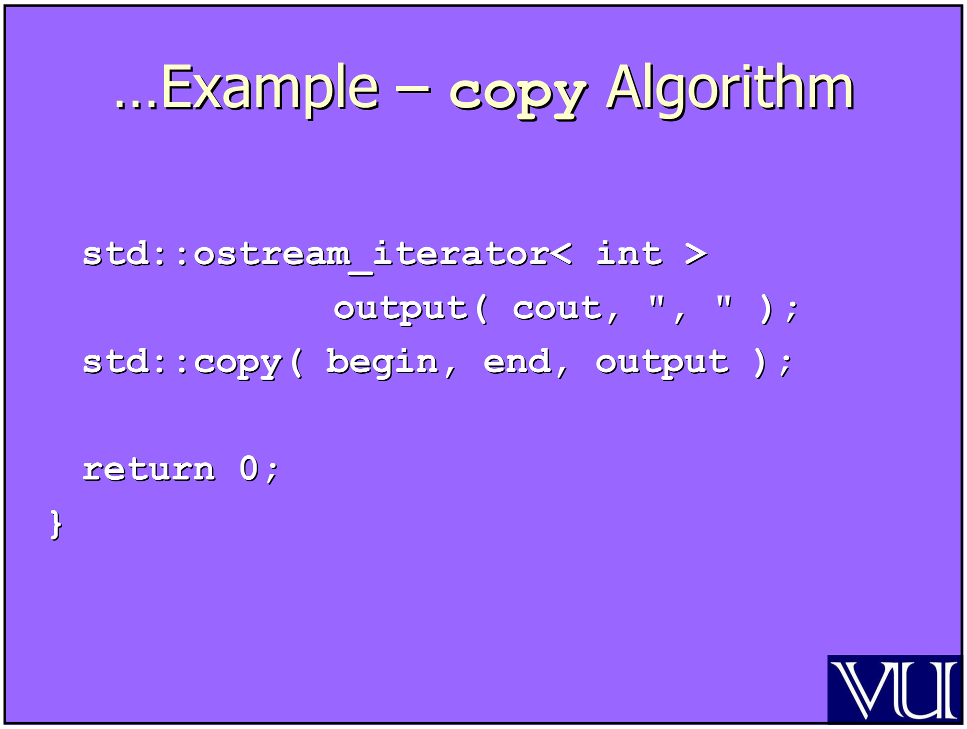
Numeric Algorithms

```
accumulate

inner_product

partial_sum

adjacent_difference
```



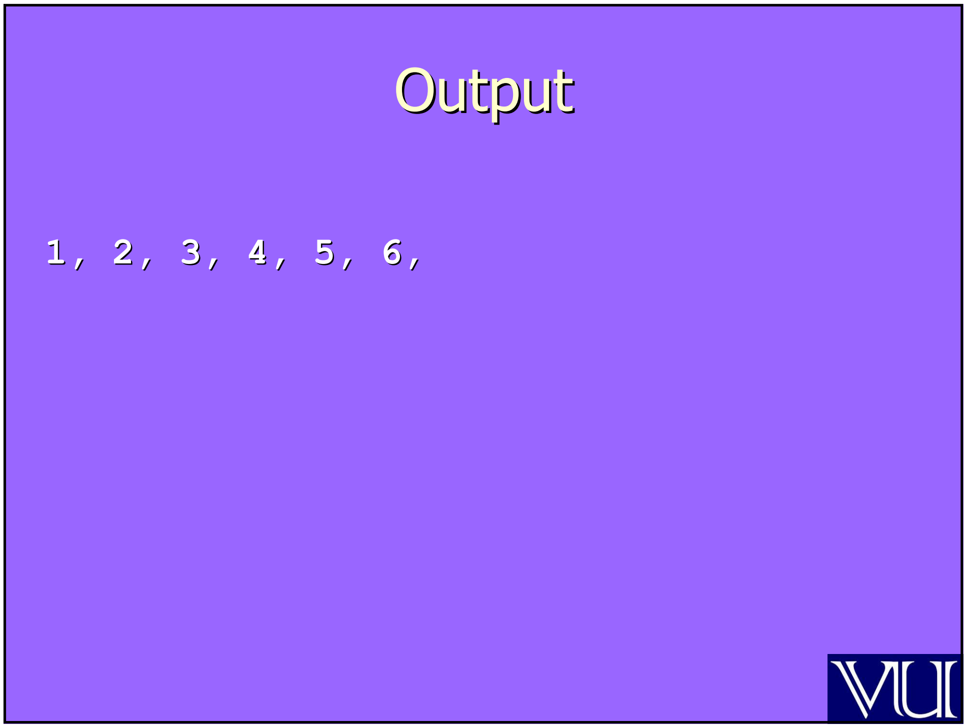
Example – **copy** Algorithm

```
#include <iostream>
using std::cout;
#include <vector>
#include <algorithm>
typedef std::vector< int > IntVector;

int main() {
  int iArray[] = {1, 2, 3, 4, 5, 6};
  IntVector iv( iArray, iArray + 6 );
```



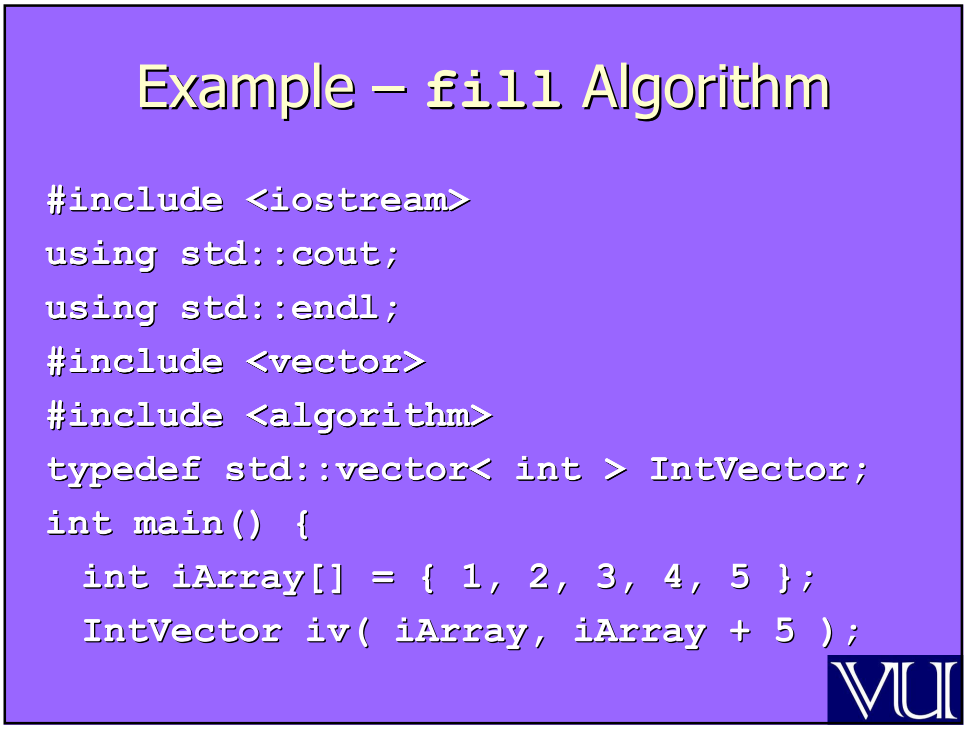
…Example – **copy** Algorithm

```
  std::ostream_iterator< int >
             output( cout, ", " );
  std::copy( begin, end, output );

  return 0;
}
```



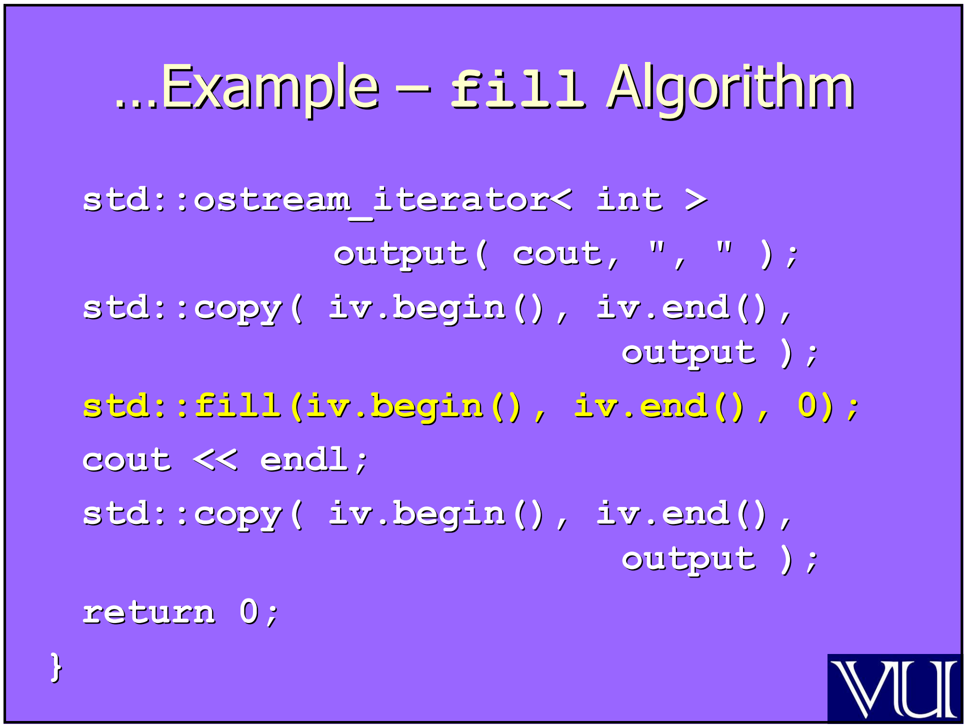
Output

```
1, 2, 3, 4, 5, 6,
```



Example – `fill` Algorithm

```
#include <iostream>
using std::cout;
using std::endl;
#include <vector>
#include <algorithm>
typedef std::vector< int > IntVector;
int main() {
  int iArray[] = { 1, 2, 3, 4, 5 };
  IntVector iv( iArray, iArray + 5 );
```



…Example – `fill` Algorithm

```
  std::ostream_iterator< int >
             output( cout, ", " );
  std::copy( iv.begin(), iv.end(),
                          output );
  std::fill(iv.begin(), iv.end(), 0);
  cout << endl;
  std::copy( iv.begin(), iv.end(),
                          output );
  return 0;
}
```