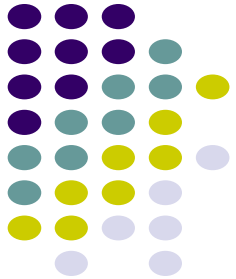
Introduction to Iterators

Content:

- **Definition:** Iterators are objects that point to elements within a container. They are used to traverse containers, allowing for element access and manipulation.
- **Types of Iterators:**
  - Input Iterator
  - Output Iterator
  - Forward Iterator
  - Bidirectional Iterator
  - Random Access Iterator

Basic Iterator Usage



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

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};
    for (std::vector<int>::iterator it = vec.begin(); it != vec.end(); ++it) {
        std::cout << *it << ' ';
    }
    return 0;
}
```

Input and Output Iterators

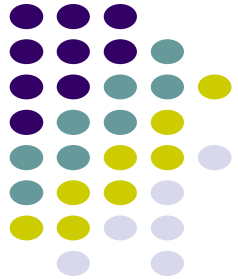
Content:

- **Input Iterator:** Reads data from the container. Used for single-pass algorithms.
- **Output Iterator:** Writes data to the container. Used for single-pass output operations.

```
#include <iostream>
#include <iterator>
#include <vector>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // Using Input Iterator
    for (std::istream_iterator<int> it(std::cin), end; it
!= end; ++it) {
        std::cout << *it << ' ';
    }
```

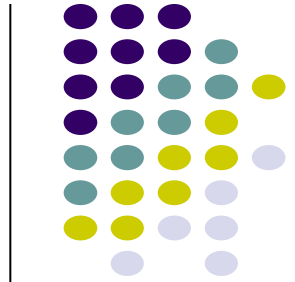
```
 // Using Output Iterator
    std::copy(vec.begin(), vec.end(),
std::ostream_iterator<int>(std::cout, " "));
    return 0;
}
```

Forward and Bidirectional Iterators



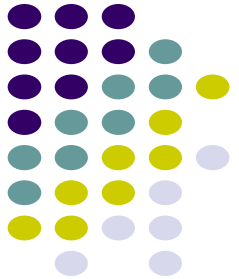
Content:

- **Forward Iterator:** Can read and write, supports multiple passes, and only moves forward.
- **Bidirectional Iterator:** Can read and write, supports multiple passes, and moves both forward and backward.



```
#include <list>
#include <iostream>

int main() {
    std::list<int> lst = {1, 2, 3, 4, 5};

    // Using Forward Iterator
    for (std::list<int>::iterator it = lst.begin(); it != lst.end(); ++it) {
        std::cout << *it << ' ';
    }

    // Using Bidirectional Iterator
    for (std::list<int>::reverse_iterator rit = lst.rbegin(); rit != lst.rend(); ++rit) {
        std::cout << *rit << ' ';
    }
    return 0;
}
```

Random Access Iterators

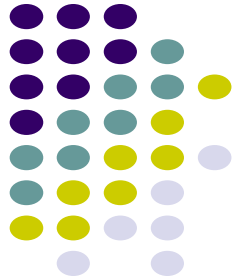


Content:

- **Random Access Iterator:** Provides the most functionality. Can read and write, supports multiple passes, and allows direct access to any element in constant time.

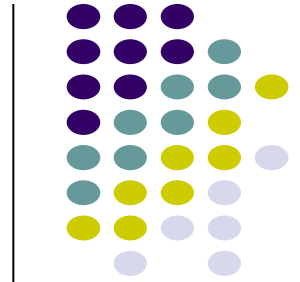
```
int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // Using Random Access Iterator
    for (std::vector<int>::iterator it = vec.begin(); it != vec.end(); ++it) {
        std::cout << *it << ' ';
    }

    // Direct access using Random Access Iterator
    std::vector<int>::iterator it = vec.begin();
    std::advance(it, 2); // Move iterator to the 3rd element
    std::cout << *it << std::endl; // Output: 3

    return 0;
}
```

Iterator Functions and Algorithms



Content:

- Common Functions:
  - `begin()`, `end()`
  - `rbegin()`, `rend()`
  - `cbegin()`, `cend()`
  - `advance()`, `distance()`



```
#include <vector>
#include <iostream>
#include <algorithm>
#include <iterator>

int main() {
    std::vector<int> vec = {1, 2, 3, 4, 5};

    // Using begin() and end()
    for (auto it = vec.begin(); it != vec.end(); ++it) {
        std::cout << *it << ' ';
    }

    // Using rbegin() and rend()
    for (auto rit = vec.rbegin(); rit != vec.rend(); ++rit) {
        std::cout << *rit << ' ';
    }

    // Using advance() and distance()
    auto it = vec.begin();
    std::advance(it, 3); // Move iterator to the 4th element
    std::cout << *it << std::endl; // Output: 4

    auto dist = std::distance(vec.begin(), it);
    // Output: 3
    std::cout << "Distance: " << dist << std::endl;

    return 0;
}
```