BITS F232: FOUNDATIONS OF DATA STRUCTURES & ALGORITHMS (1ST SEMESTER 2023-24) STL VECTORS, LIST ADT

Chittaranjan Hota, PhD
Sr. Professor of Computer Sc.
BITS-Pilani Hyderabad Campus
hota[AT]hyderabad.bits-pilani.ac.in

RECAP

STL VECTORS WITH ALGORITHMS

```
#include <vector>
using std::vector;

vector<int> myVector(100);
```

vector(*n*):

size():

empty():

sort(*p*,*q*):

resize(*n*):

reserve(*n*):

random_shuffle(*p*,*q*):

operator[*i*]:

reverse(*p*,*q*):

at(*i*):

find(*p*,*q*,*e*):

front():

min_element(*p*,*q*):

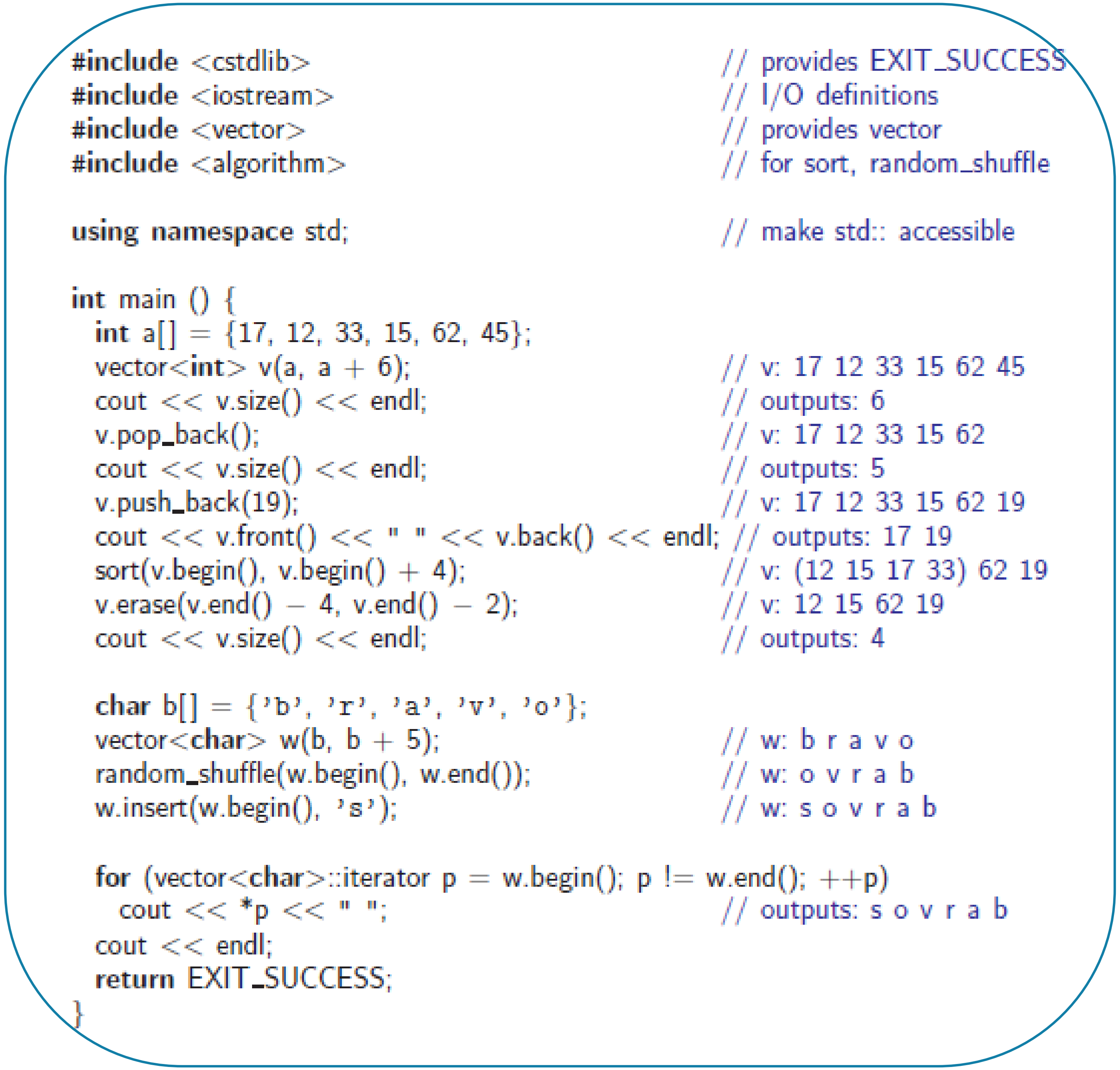
back():

max_element(*p*,*q*):

push_back(*e*):

for_each(*p*,*q*,*f*):

pop_back():



```
#include <cstdlib>                              // provides EXIT_SUCCESS
#include <iostream>                             // I/O definitions
#include <vector>                               // provides vector
#include <algorithm>                            // for sort, random_shuffle

using namespace std;                            // make std:: accessible

int main () {
  int a[] = {17, 12, 33, 15, 62, 45};
  vector<int> v(a, a + 6);                      // v: 17 12 33 15 62 45
  cout << v.size() << endl;                     // outputs: 6
  v.pop_back();                                 // v: 17 12 33 15 62
  cout << v.size() << endl;                     // outputs: 5
  v.push_back(19);                              // v: 17 12 33 15 62 19
  cout << v.front() << " " << v.back() << endl; // outputs: 17 19
  sort(v.begin(), v.begin() + 4);               // v: (12 15 17 33) 62 19
  v.erase(v.end() - 4, v.end() - 2);            // v: 12 15 62 19
  cout << v.size() << endl;                     // outputs: 4

  char b[] = {'b', 'r', 'a', 'v', 'o'};
  vector<char> w(b, b + 5);                     // w: b r a v o
  random_shuffle(w.begin(), w.end());           // w: o v r a b
  w.insert(w.begin(), 's');                     // w: s o v r a b

  for (vector<char>::iterator p = w.begin(); p != w.end(); ++p)
    cout << *p << " ";                          // outputs: s o v r a b
  cout << endl;
  return EXIT_SUCCESS;
}
```

POSITION ADT & ITERATORS: LIST ADT

- What is a Position ADT?
- It gives a unified view of diverse ways of storing data, such as:
  - a cell of an array
  - a node of a linked list
- Just one method:
  - object p.element(): returns the element at position
  - In C++ it is convenient to implement this as what?

- List ADT establishes a before/after relation between positions

DOUBLY LINKED LIST

A doubly linked list provides a natural implementation of the List ADT.

Nodes implement Position and store:

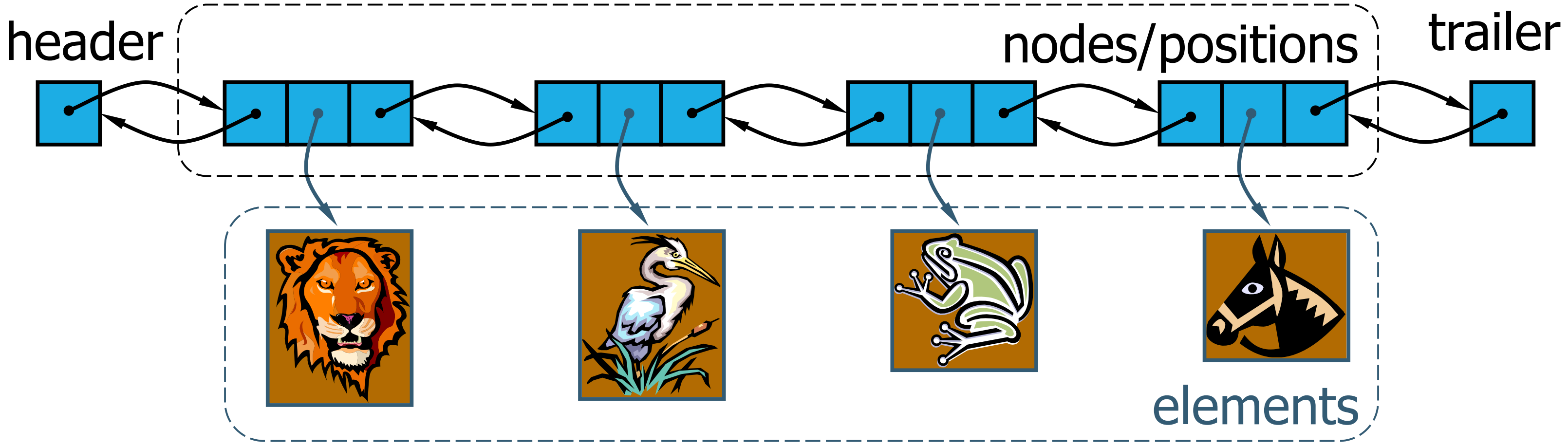
- element
- link to the previous node
- link to the next node

Special trailer and header nodes are used as Sentinels.

Complexity?

```
Algorithm insert(p, e): {insert e
                        before p}
    1. Create a new node v
    2. v→element = e
    3. u = p→prev
    4. v→next = p;  p→prev = v
       {link in v before p}
    5. v→prev = u;  u→next = v
       {link in v after u}
```

```
Algorithm remove(p):
  u = p→prev
  w = p→next
  u→next = w {linking out p}
  w→prev = u
```



CONTAINERS AND ITERATORS

- What is a Container?
- Can you give some examples?
- Various notions of iterator:
  - (standard) iterator: allows read-write access to elements
  - const iterator: provides read-only access to elements
  - bidirectional iterator: supports both ++p and --p
  - random-access iterator: supports both p+i and p-i

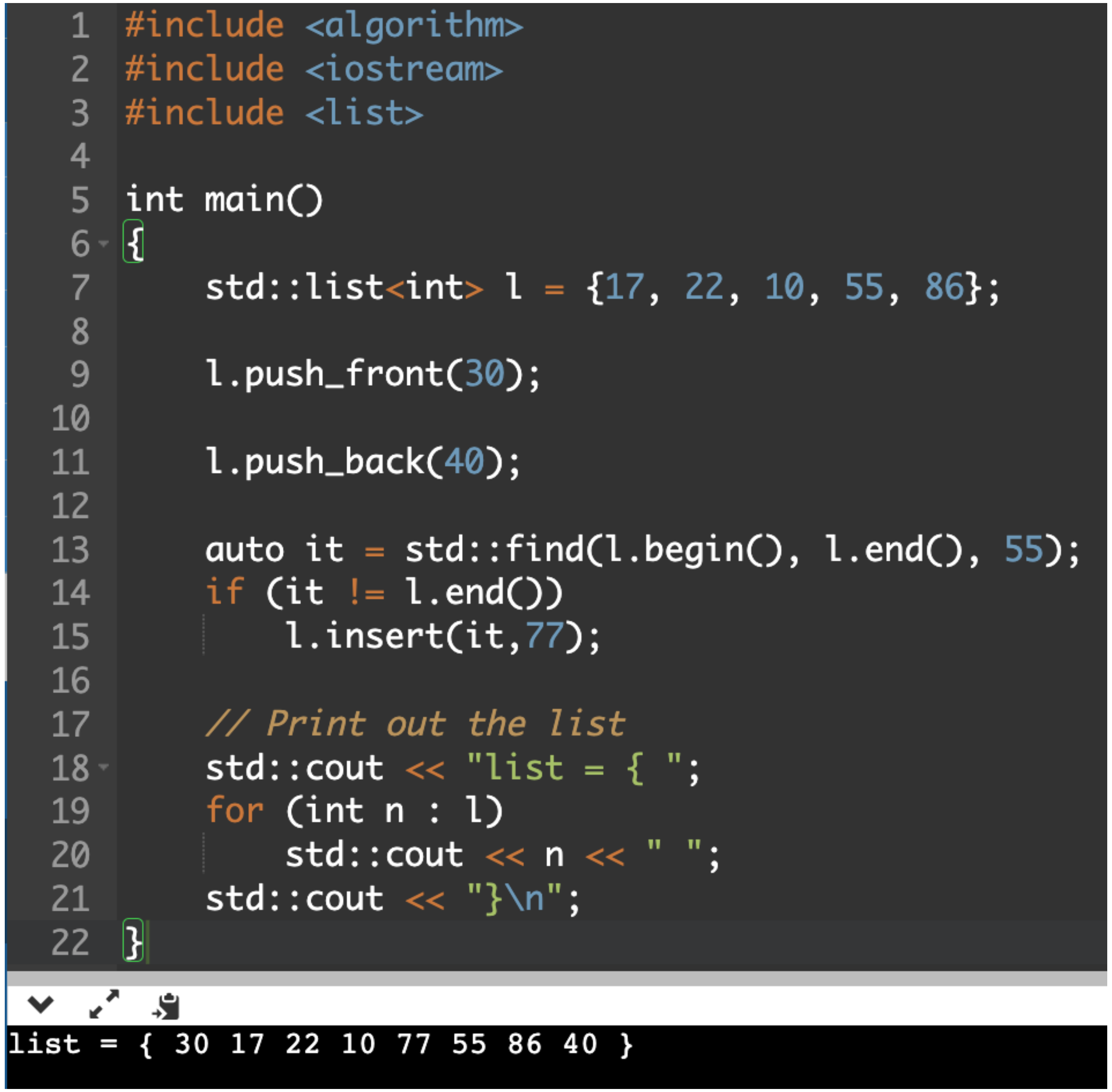
Let C be a container and p be an iterator for C:

How will you iterate through the container?

Example: (with an STL vector)

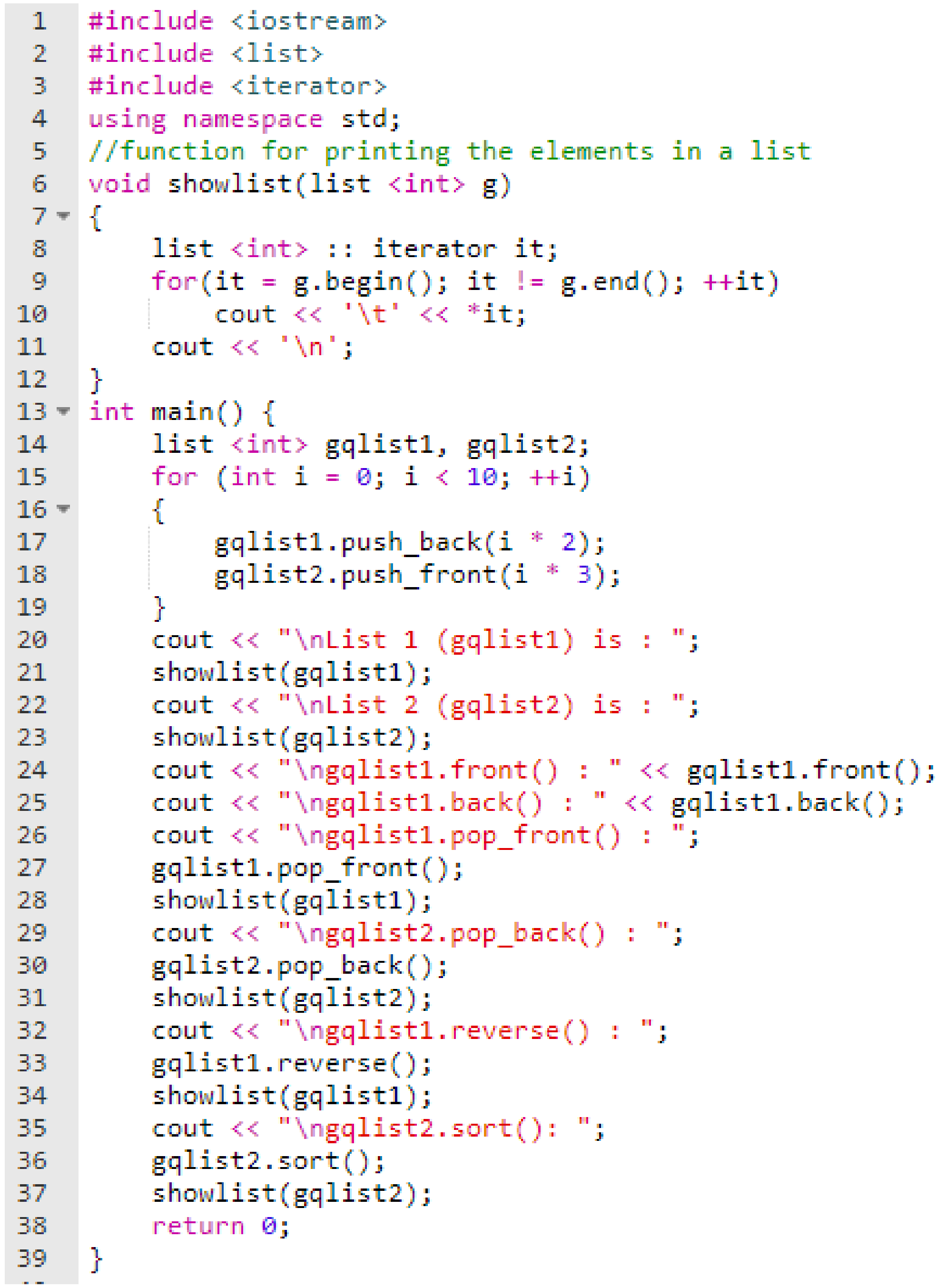
```
typedef vector<int>::iterator Iterator;
int sum = 0;
for (Iterator p = V.begin(); p != V.end(); ++p)
  sum += *p;
return sum;
```

STL LISTS IN C++



```cpp
#include <algorithm>
#include <iostream>
#include <list>

int main()
{
    std::list<int> l = {17, 22, 10, 55, 86};

    l.push_front(30);

    l.push_back(40);

    auto it = std::find(l.begin(), l.end(), 55);
    if (it != l.end())
        l.insert(it,77);

    // Print out the list
    std::cout << "list = { ";
    for (int n : l)
        std::cout << n << " ";
    std::cout << "}\n";
}
```

```
list = { 30 17 22 10 77 55 86 40 }
```

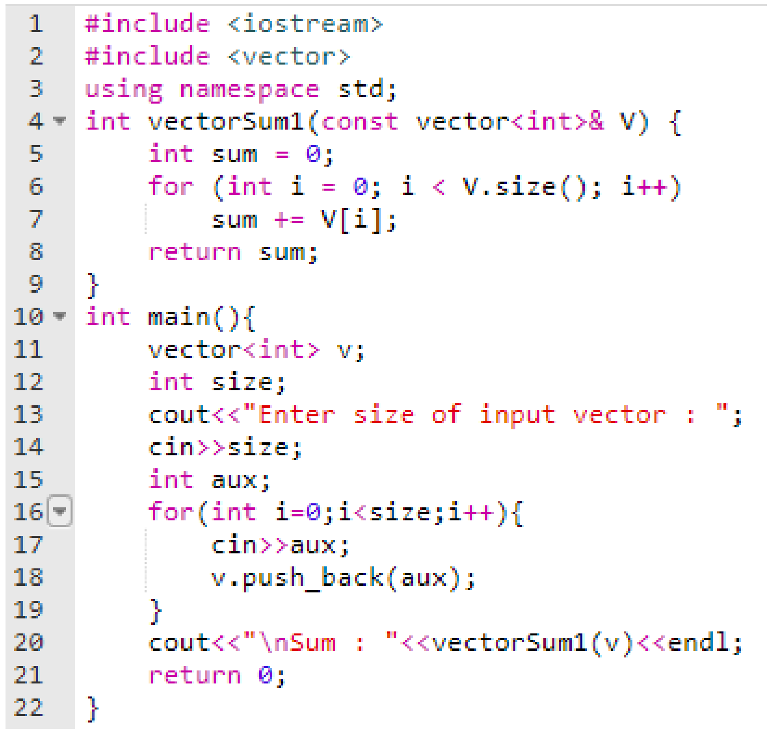


```cpp
#include <iostream>
#include <list>
#include <iterator>
using namespace std;
//function for printing the elements in a list
void showlist(list <int> g)
{
    list <int> :: iterator it;
    for(it = g.begin(); it != g.end(); ++it)
        cout << '\t' << *it;
    cout << '\n';
}
int main() {
    list <int> gqlist1, gqlist2;
    for (int i = 0; i < 10; ++i)
    {
        gqlist1.push_back(i * 2);
        gqlist2.push_front(i * 3);
    }
    cout << "\nList 1 (gqlist1) is : ";
    showlist(gqlist1);
    cout << "\nList 2 (gqlist2) is : ";
    showlist(gqlist2);
    cout << "\ngqlist1.front() : " << gqlist1.front();
    cout << "\ngqlist1.back() : " << gqlist1.back();
    cout << "\ngqlist1.pop_front() : ";
    gqlist1.pop_front();
    showlist(gqlist1);
    cout << "\ngqlist2.pop_back() : ";
    gqlist2.pop_back();
    showlist(gqlist2);
    cout << "\ngqlist1.reverse() : ";
    gqlist1.reverse();
    showlist(gqlist1);
    cout << "\ngqlist2.sort(): ";
    gqlist2.sort();
    showlist(gqlist2);
    return 0;
}
```

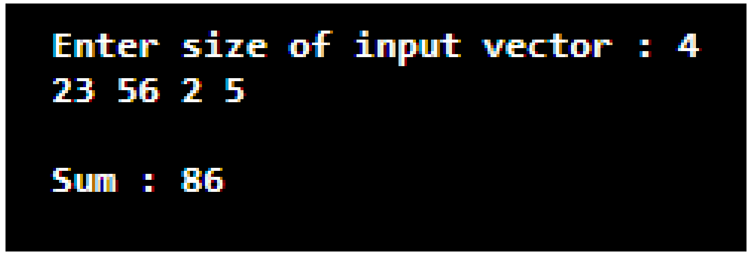
https://en.cppreference.com/ https://www.geeksforgeeks.org/

INDEX VS POSITION: MORE EXAMPLES

Using Indexing Operator

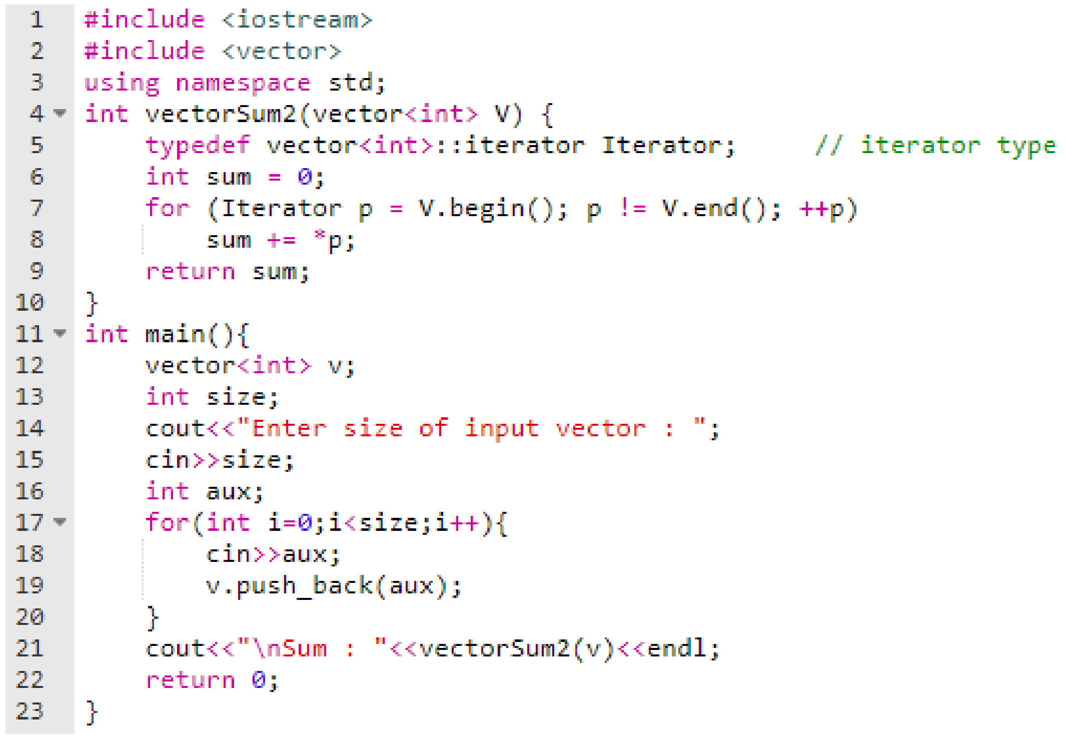


```cpp
#include <iostream>
#include <vector>
using namespace std;
int vectorSum1(const vector<int>& V) {
    int sum = 0;
    for (int i = 0; i < V.size(); i++)
        sum += V[i];
    return sum;
}
int main(){
    vector<int> v;
    int size;
    cout<<"Enter size of input vector : ";
    cin>>size;
    int aux;
    for(int i=0;i<size;i++){
        cin>>aux;
        v.push_back(aux);
    }
    cout<<"\nSum : "<<vectorSum1(v)<<endl;
    return 0;
}
```

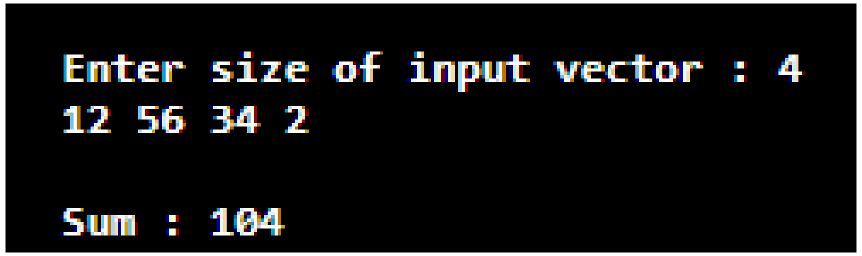


```
Enter size of input vector : 4
23 56 2 5

Sum : 86
```

Using Iterators



```cpp
#include <iostream>
#include <vector>
using namespace std;
int vectorSum2(vector<int> V) {
    typedef vector<int>::iterator Iterator;     // iterator type
    int sum = 0;
    for (Iterator p = V.begin(); p != V.end(); ++p)
        sum += *p;
    return sum;
}
int main(){
    vector<int> v;
    int size;
    cout<<"Enter size of input vector : ";
    cin>>size;
    int aux;
    for(int i=0;i<size;i++){
        cin>>aux;
        v.push_back(aux);
    }
    cout<<"\nSum : "<<vectorSum2(v)<<endl;
    return 0;
}
```



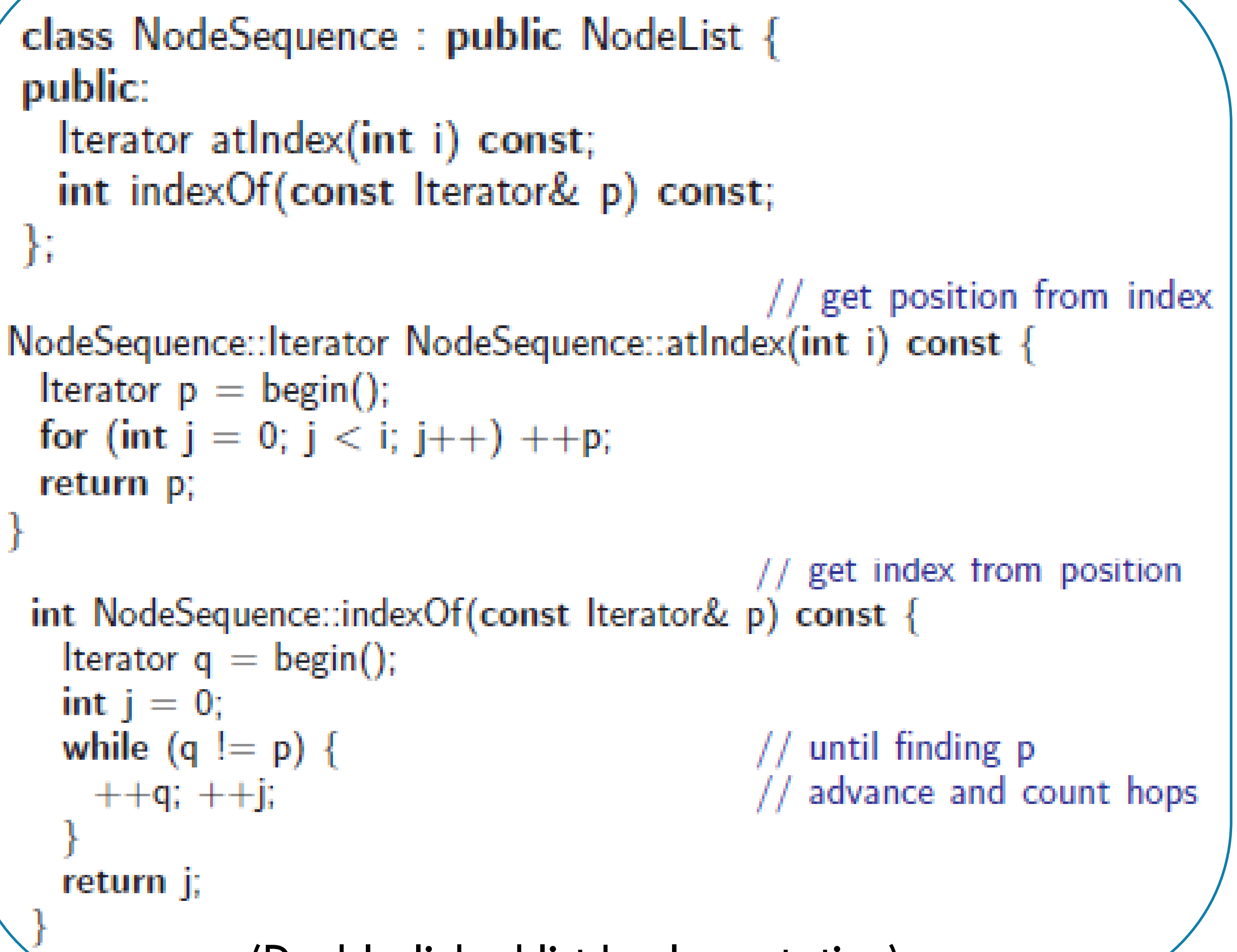
```
Enter size of input vector : 4
12 56 34 2

Sum : 104
```

SEQUENCE ADT

- The Sequence ADT generalizes the Vector and List ADTs

- Elements are accessed by:
  - Index, or
  - Position

- Methods and Usages?



```
class NodeSequence : public NodeList {
public:
  Iterator atIndex(int i) const;
  int indexOf(const Iterator& p) const;
};
                                                // get position from index
NodeSequence::Iterator NodeSequence::atIndex(int i) const {
  Iterator p = begin();
  for (int j = 0; j < i; j++) ++p;
  return p;
}
                                                // get index from position
 int NodeSequence::indexOf(const Iterator& p) const {
   Iterator q = begin();
   int j = 0;
   while (q != p) {                             // until finding p
     ++q; ++j;                                  // advance and count hops
   }
   return j;
 }
```

(Doubly-linked list Implementation)

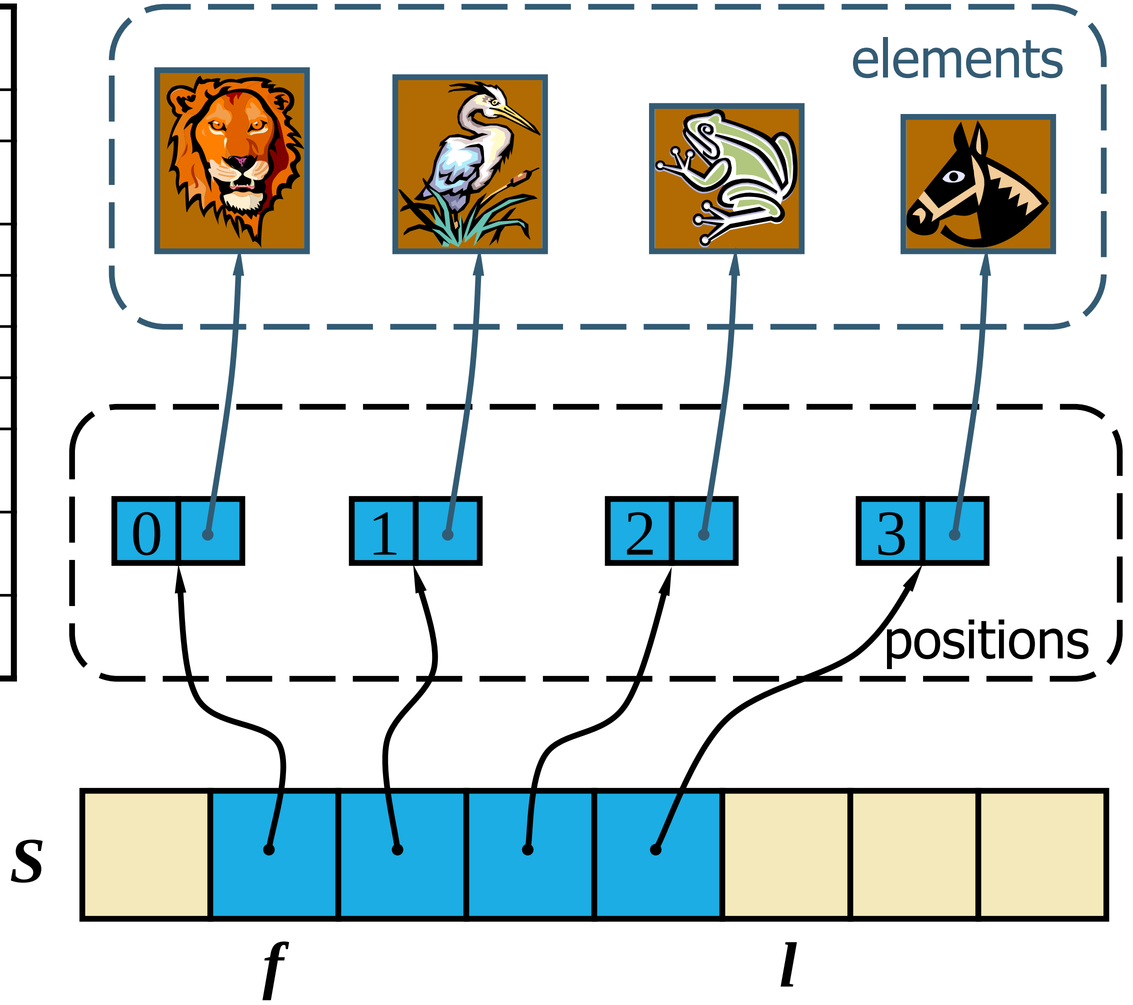
SEQUENCE ADT: ARRAY BASED

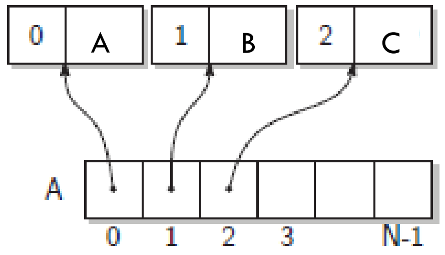
- We use a circular array storing positions.

  A position object stores:
  - Element
  - Index

- Indices *f* and *l* keep track of first and last positions.

Operation	Array	List
size, empty	1	1
atIndex, indexOf, at	1	*n*
begin, end	1	1
set(p,e)	1	1
set(i,e)	1	*n*
insert(i,e), erase(i)	*n*	*n*
insertBack, eraseBack	1	1
insertFront, eraseFront	*n*	1
insert(p,e), erase(p)	*n*	1


elements
0
1
2
3
positions
S
f
l


0
A
1
B
2
C
A
0
1
2
3
N-1