Codes on use of pointer

```
#include <iostream>                                          //Line 1
#include <iomanip>                                           //Line 2

using namespace std;                                         //Line 3

const double PI = 3.1416;                                    //Line 4

int main()                                                   //Line 5
{                                                            //Line 6
    double radius;                                           //Line 7
    double *radiusPtr;                                       //Line 8

    cout << fixed << showpoint << setprecision(2);           //Line 9

    radius = 2.5;                                            //Line 10
    radiusPtr = &radius;                                     //Line 11

    cout << "Line 12: Radius = " << radius
         << ", area = " << PI * radius * radius << endl;     //Line 12

    cout << "Line 13: Radius = " << *radiusPtr
         << ", area = "
         << PI * (*radiusPtr) * (*radiusPtr) << endl;        //Line 13

    cout << "Line 14: Enter the radius: ";                   //Line 14
    cin >> *radiusPtr;                                       //Line 15
    cout << endl;                                            //Line 16

    cout << "Line 17: Radius = " << radius << ", area = "
         << PI * radius * radius << endl;                    //Line 17
    cout << "Line 18: Radius = " << *radiusPtr
         << ", area = "
         << PI * (*radiusPtr) * (*radiusPtr) << endl
         << endl;                                            //Line 18


    cout << "Line 19: Address of radiusPtr: "
         << &radiusPtr << endl;                              //Line 19
    cout << "Line 20: Value stored in radiusPtr: "
         << radiusPtr << endl;                               //Line 20
    cout << "Line 21: Address of radius: "
         << &radius << endl;                                 //Line 21
    cout << "Line 22: Value stored in radius: "
         << radius << endl;                                  //Line 22

    return 0;                                                //Line 23
}                                                            //Line 24
```

```
//This program illustrates how to use the operators new and delete.

#include <iostream>                                          //Line 1
#include <iomanip>                                           //Line 2

using namespace std;                                         //Line 3

const double PI = 3.1416;                                    //Line 4

int main()                                                   //Line 5
{                                                            //Line 6
    double *radiusPtr;                                       //Line 7

    cout << fixed << showpoint << setprecision(2);           //Line 8

    radiusPtr = new double;                                  //Line 9

    cout << "Line 10: Enter the radius: ";                   //Line 10
    cin >> *radiusPtr;                                       //Line 11
    cout << endl;                                            //Line 12

    cout << "Line 13: Radius = " << *radiusPtr
         << ", area = " << PI * (*radiusPtr) * (*radiusPtr)
         << endl << endl;                                    //Line 13

    cout << "Line 14: Address of radiusPtr: "
         << &radiusPtr << endl;                              //Line 14
    cout << "Line 15: Value stored in radiusPtr: "
         << radiusPtr << endl;                               //Line 15
    cout << "Line 16: Value stored in the memory "
         << "location to which \n         radiusPtr "
         << "is pointing: " << *radiusPtr << endl;           //Line 16

    delete radiusPtr;                                        //Line 17

    cout << "Line 18: After using the delete operator, "
         << "the value stored in the location\n         "
         << "to which radiusPtr is pointing: "
         << *radiusPtr << endl;                              //Line 18

    double *lengthPtr = new double;                          //Line 19

    radiusPtr = new double;                                  //Line 20

    *radiusPtr = 5.38;                                       //Line 21

    cout << "Line 22: Address of radiusPtr: "
         << &radiusPtr << endl;                              //Line 22
    cout << "Line 23: Value stored in radiusPtr: "
         << radiusPtr << endl;                               //Line 23
    cout << "Line 24: Value stored in the memory "
         << "location to which radiusPtr is pointing: "
         << *radiusPtr << endl;                              //Line 24
```

```
    cout << "Line 25: Value stored in lengthPtr: "
         << lengthPtr << endl;                                  //Line 25

    return 0;                                                   //Line 26
}                                                               //Line 27
```

Use of STL vector:

```
// Demonstrate the basic sequence container operations.
//
// This example uses vector, but the same techniques can be
// applied to any sequence container.
```

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

using namespace std;

void show(const char *msg, vector<char> vect);

int main() {
  // Declare an empty vector that can hold char objects.
  vector<char> v;

  // Declare an iterator to a vector<char>.
  vector<char>::iterator itr;

  // Obtain an iterator to the start of v.
  itr = v.begin();

  // Insert characters into v. An iterator to the inserted
  // object is returned.
  itr = v.insert(itr, 'A');
  itr = v.insert(itr, 'B');
  v.insert(itr, 'C');

  // Display the contents of v.
  show("The contents of v: ", v);

  // Declare a reverse iterator.
  vector<char>::reverse_iterator ritr;

  // Use a reverse iterator to show the contents of v in reverse.
  cout << "Here is v in reverse: ";
  for(ritr = v.rbegin(); ritr != v.rend(); ++ritr)
    cout << *ritr << " ";
  cout << "\n\n";

  // Create another vector that is the same as the first.
  vector<char> v2(v);
  show("The contents of v2: ",v2);
  cout << "\n";

  // Show the size of v, which is the number of elements
  // currently held by v.
  cout << "Size of v is " << v.size() << "\n\n";

  // Compare two containers.
  if(v == v2) cout << "v and v2 are equivalent.\n\n";

  // Insert more characters into v and v2. This time,
  // insert them at the end.
  cout << "Insert more characters into v and v2.\n";
  v.insert(v.end(), 'D');
  v.insert(v.end(), 'E');
  v2.insert(v2.end(), 'X');
  show("The contents of v: ", v);
```

```
  show("The contents of v2: ", v2);
  cout << "\n";

  // Determine if v is less than v2. This is a
  // lexicographical compare. Therefore, the first
  // non-matching element determines which
  // container is less than another.
  if(v < v2) cout << "v is less than v2.\n\n";

  // Now, insert Z at the start of v.
  cout << "Insert Z at the start of v.\n";
  v.insert(v.begin(), 'Z');
  show("The contents of v: ", v);
  cout << "\n";

  // Now, compare v to v2 again.
  if(v > v2) cout << "Now, v is greater than v2.\n\n";

  // Remove the first element from v2.
  v2.erase(v2.begin());
  show("v2 after removing the first element: ", v2);
  cout << "\n";

  // Create another vector.
  vector<char> v3;
  v3.insert(v3.end(), 'X');
  v3.insert(v3.end(), 'Y');
  v3.insert(v3.end(), 'Z');
  show("The contents of v3: ", v3);
  cout << "\n";

  // Exchange the contents of v and v3.
  cout << "Exchange v and v3.\n";
  v.swap(v3);
  show("The contents of v: ", v);
  show("The contents of v3: ", v3);
  cout << "\n";

  // Clear v.
  v.clear();
  if(v.empty()) cout << "v is now empty.";

  return 0;
}
```

```
// Display the contents of a vector<char> by using
// an iterator.
void show(const char *msg, vector<char> vect) {
  vector<char>::iterator itr;

  cout << msg;
  for(itr=vect.begin(); itr != vect.end(); ++itr)
    cout << *itr << " ";
  cout << "\n";
}
```

```
// Demonstrate vector.

#include <iostream>
#include <vector>

using namespace std;

void show(const char *msg, vector<int> vect);

int main() {

  // Declare a vector that has an initial capacity of 10.
  vector<int> v(10);

  // Assign its elements some values. Notice how this is
  // done using the standard array-subscripting syntax.
  // Notice that the number of elements in the vector is
  // obtained by calling size().
  for(unsigned i=0; i < v.size(); ++i) v[i] = i*i;

  show("Contents of v: ", v);

  // Compute the average of the values. Again, notice
  // the use of the subscripting operator.
  int sum = 0;
  for(unsigned i=0; i < v.size(); ++i) sum += v[i];
  double avg = sum / v.size();
  cout << "The average of the elements is " << avg << "\n\n";

  // Add elements to the end of v.
  v.push_back(100);
  v.push_back(121);

  show("v after pushing elements onto the end: ", v);
  cout << endl;

  // Now use pop_back() to remove one element.
  v.pop_back();
  show("v after back-popping one element: ", v);
  cout << endl;

  cout << "The first and last element in v as"
       << " pointed to by begin() and end()-1:\n"
       << *v.begin() << ", " << *(v.end()-1) << "\n\n";

  cout << "The first and last element in v as"
       << " pointed to by rbegin() and rend()-1:\n"
       << *v.rbegin() << ", " << *(v.rend()-1) << "\n\n";

  // Declare an iterator to a vector<int>.
  vector<int>::iterator itr;
```

```
  // Now, declare reverse iterator to a vector<int>
  vector<int>::reverse_iterator ritr;

  // Cycle through v in the forward direction using an iterator.
  cout << "Cycle through the vector in the forward direction:\n";
  for(itr = v.begin(); itr != v.end(); ++itr)
    cout << *itr << " ";
  cout << "\n\n";
  cout << "Now, use a reverse iterator to cycle through in the"
       << " reverse direction:\n";

  // Cycle through v in the reverse direction using a reverse_iterator
  for(ritr = v.rbegin(); ritr != v.rend(); ++ritr)
    cout << *ritr << " ";
  cout << "\n\n";

  // Create another vector that contains a subrange of v.
  vector<int> v2(v.begin()+2, v.end()-4);

  // Display the contents of v2 by using an iterator.
  show("v2 contains a subrange of v: ", v2);
  cout << endl;

  // Change the values of some of v2's elements.
  v2[1] = 100;
  v2[2] = 88;
  v2[4] = 99;
  show("After the assignments, v2 now contains: ", v2);
  cout << endl;

  // Create an empty vector and then assign it a sequence
  // that is the reverse of v.
  vector<int> v3;
  v3.assign(v.rbegin(), v.rend());
  show("v3 contains the reverse of v: ", v3);
  cout << endl;

  // Show the size and capacity of v.
  cout << "Size of v is " << v.size() << ". The capacity is "
       << v.capacity() << ".\n";

  // Now, resize v.
  v.resize(20);
  cout << "After calling resize(20), the size  of v is "
       << v.size() << " and the capacity is "
       << v.capacity() << ".\n";

   // Now, reserve space for 50 elements.
   v.reserve(50);
   cout << "After calling reserve(50), the size of v is "
        << v.size() << " and the capacity is "
        << v.capacity() << ".\n";

   return 0;
}
```

```
// Display the contents of a vector<int>.
void show(const char *msg, vector<int> vect) {
  cout << msg;
  for(unsigned i=0; i < vect.size(); ++i)
    cout << vect[i] << " ";
  cout << "\n";
}
```

```
// Demonstrate deque.

#include <iostream>
#include <deque>

using namespace std;

void show(const char *msg, deque<int> q);

int main() {

  // Declare a deque that has an initial capacity of 10.
  deque<int> dq(10);

  // Assign its elements some values. Notice how this is
  // done using the standard array-subscripting syntax.
  // Notice that the number of elements in the deque is
  // obtained by calling size().
  for(unsigned i=0; i < dq.size(); ++i) dq[i] = i*i;

  show("Contents of dq: ", dq);

  // Compute the average of the values. Again, notice
  // the use of the subscripting operator.
  int sum = 0;
  for(unsigned i=0; i < dq.size(); ++i) sum += dq[i];
  double avg = sum / dq.size();
  cout << "The average of the elements is " << avg << "\n\n";

  // Add elements to the end of dq.
  dq.push_back(100);
  dq.push_back(121);

  show("dq after pushing elements onto the end: ", dq);
  cout << endl;

  // Now use pop_back() to remove one element.
  dq.pop_back();
  show("dq after back-popping one element: ", dq);
  cout << endl;

  cout << "The first and last element in dq as"
       << " pointed to by begin() and end()-1:\n"
       << *dq.begin() << ", " << *(dq.end()-1) << "\n\n";

  cout << "The first and last element in dq as"
       << " pointed to by rbegin() and rend()-1:\n"
       << *dq.rbegin() << ", " << *(dq.rend()-1) << "\n\n";

  // Declare an iterator to a deque<int>.
```

```
  deque<int>::iterator itr;
  // Now, declare reverse iterator to a deque<int>
  deque<int>::reverse_iterator ritr;

  // Cycle through dq in the forward direction using an iterator.
  cout << "Cycle through the deque in the forward direction:\n";
  for(itr = dq.begin(); itr != dq.end(); ++itr)
    cout << *itr << " ";
  cout << "\n\n";
  cout << "Now, use a reverse iterator to cycle through in the"
       << " reverse direction:\n";

  // Cycle through dq in the reverse direction using a reverse_iterator.
  for(ritr = dq.rbegin(); ritr != dq.rend(); ++ritr)
    cout << *ritr << " ";
  cout << "\n\n";

  // Create another deque that contains a subrange of dq.
  deque<int> dq2(dq.begin()+2, dq.end()-4);

  // Display the contents of dq2 by using an iterator.
  show("dq2 contains a subrange of dq: ", dq2);
  cout << endl;

  // Change the values of some of dq2's elements.
  dq2[1] = 100;
  dq2[2] = 88;
  dq2[4] = 99;
  show("After the assignments, dq2 now contains: ", dq2);
  cout << endl;

  // Create an empty deque and then assign it a sequence
  // that is the reverse of dq.
  deque<int> dq3;
  dq3.assign(dq.rbegin(), dq.rend());
  show("dq3 contains the reverse of dq: ", dq3);
  cout << endl;

  // Push an element onto the front of dq.
  dq.push_front(-31416);
  show("dq after call to push_front(): ", dq);
  cout <<endl;

  // Now, clear dq by popping elements one at a time.
  cout << "Front popping elements from dq.\n";
  while(dq.size() > 0) {
    cout << "Popping: " << dq.front() << endl;
    dq.pop_front();
  }
  if(dq.empty()) cout << "dq is now empty.\n";

   return 0;
}
```

```
// Display the contents of a deque<int>.
void show(const char *msg, deque<int> q) {
  cout << msg;
  for(unsigned i=0; i < q.size(); ++i)
    cout << q[i] << " ";
  cout << "\n";
}
```

Example

The following example demonstrates **list**:

```
// Demonstrate list

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

```
using namespace std;

void show(const char *msg, list<char> lst);

int main() {

  // Declare two lists.
  list<char> lstA;
  list<char> lstB;

  // Use push_back() to give the lists some elements.
  lstA.push_back('A');
  lstA.push_back('F');
  lstA.push_back('B');
  lstA.push_back('R');

  lstB.push_back('X');
  lstB.push_back('A');
  lstB.push_back('F');

  show("Original contents of lstA: ", lstA);
  show("Original contents of lstB: ", lstB);
  cout << "Size of lstA is " << lstA.size() << endl;
  cout << "Size of lstB is "<< lstB.size() << endl;
  cout << endl;

  // Sort lstA and lstB
  lstA.sort();
  lstB.sort();

  show("Sorted contents of lstA: ", lstA);
  show("Sorted contents of lstB: ", lstB);
  cout << endl;

  // Merge lstB into lstA.
  lstA.merge(lstB);
  show("lstA after merge: " , lstA);
  if(lstB.empty()) cout << "lstB is now empty().\n";
  cout << endl;

  // Remove duplicates from lstA.
  lstA.unique();
  show("lstA after call to unique(): ", lstA);
  cout << endl;

  // Give lstB some new elements.
  lstB.push_back('G');
  lstB.push_back('H');
  lstB.push_back('P');

  show("New contents of lstB: ", lstB);
  cout << endl;
```

```
  // Now, splice lstB into lstA.
  list<char>::iterator itr = lstA.begin();
  ++itr;
  lstA.splice(itr, lstB);
  show("lstA after splice: ", lstA);
  cout << endl;

  // Remove A and H.
  lstA.remove('A');
  lstA.remove('H');
  show("lstA after removing A and H: ", lstA);
  cout << endl;

  return 0;
}

// Display the contents of a list<char>.
void show(const char *msg, list<char> lst) {
  list<char>::iterator itr;

  cout << msg;

  for(itr = lst.begin(); itr != lst.end(); ++itr)
    cout << *itr << " ";

  cout << "\n";
}
```

The output is shown here:

```
  // Demonstrate the sequence container adaptors.

  #include <iostream>
  #include <string>
  #include <queue>
  #include <stack>

  using namespace std;
```

```
int main()
{
  // Demonstrate queue.
  queue<string> q;

  cout << "Demonstrate a queue for strings.\n";

  cout << "Pushing one two three four\n";
  q.push("one");
  q.push("two");
  q.push("three");
  q.push("four");

  cout << "Now, retrieve those values in FIFO order.\n";
  while(!q.empty()) {
    cout << "Popping ";
    cout << q.front() << "\n";
    q.pop();
  }
  cout << endl;

  // Demonstrate priority_queue.
  priority_queue<int> pq;

  cout << "Demonstrate a priority_queue for integers.\n";

  cout << "Pushing 1, 3, 4, 2.\n";
  pq.push(1);
  pq.push(3);
  pq.push(4);
  pq.push(2);

  cout << "Now, retrieve those values in priority order.\n";
  while(!pq.empty()) {
    cout << "Popping ";
    cout << pq.top() << "\n";
    pq.pop();
  }
  cout << endl;

  // Finally, demonstrate stack.
  stack<char> stck;

  cout << "Demonstrate a stack for characters.\n";

  cout << "Pushing  A, B, C, and D.\n";
  stck.push('A');
  stck.push('B');
  stck.push('C');
  stck.push('D');

  cout << "Now, retrieve those values in LIFO order.\n";
  while(!stck.empty()) {
```

```
    cout << "Popping: ";
    cout << stck.top() << "\n";
    stck.pop();
  }

  return 0;
}
```