

# Chapter 16



## Creating the Document and Improving the View

# Line vs. Curve

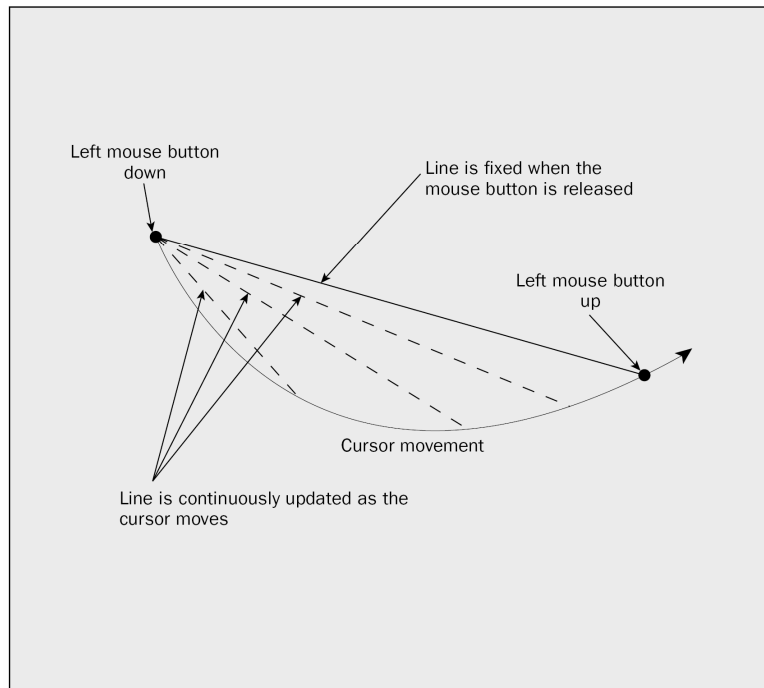


Figure 14-7

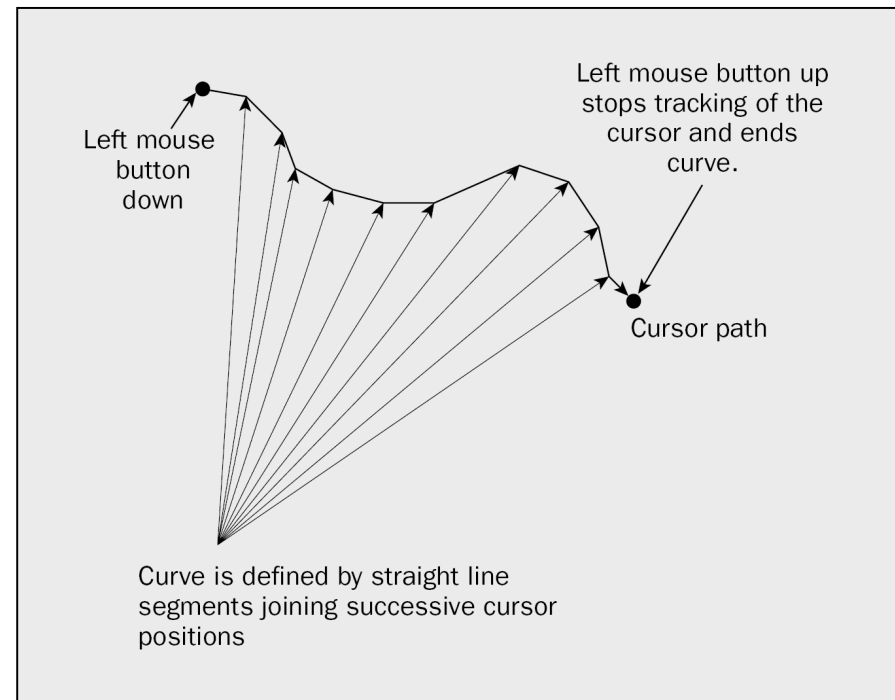


Figure 14-10

# Figure 16-6 (P.926)

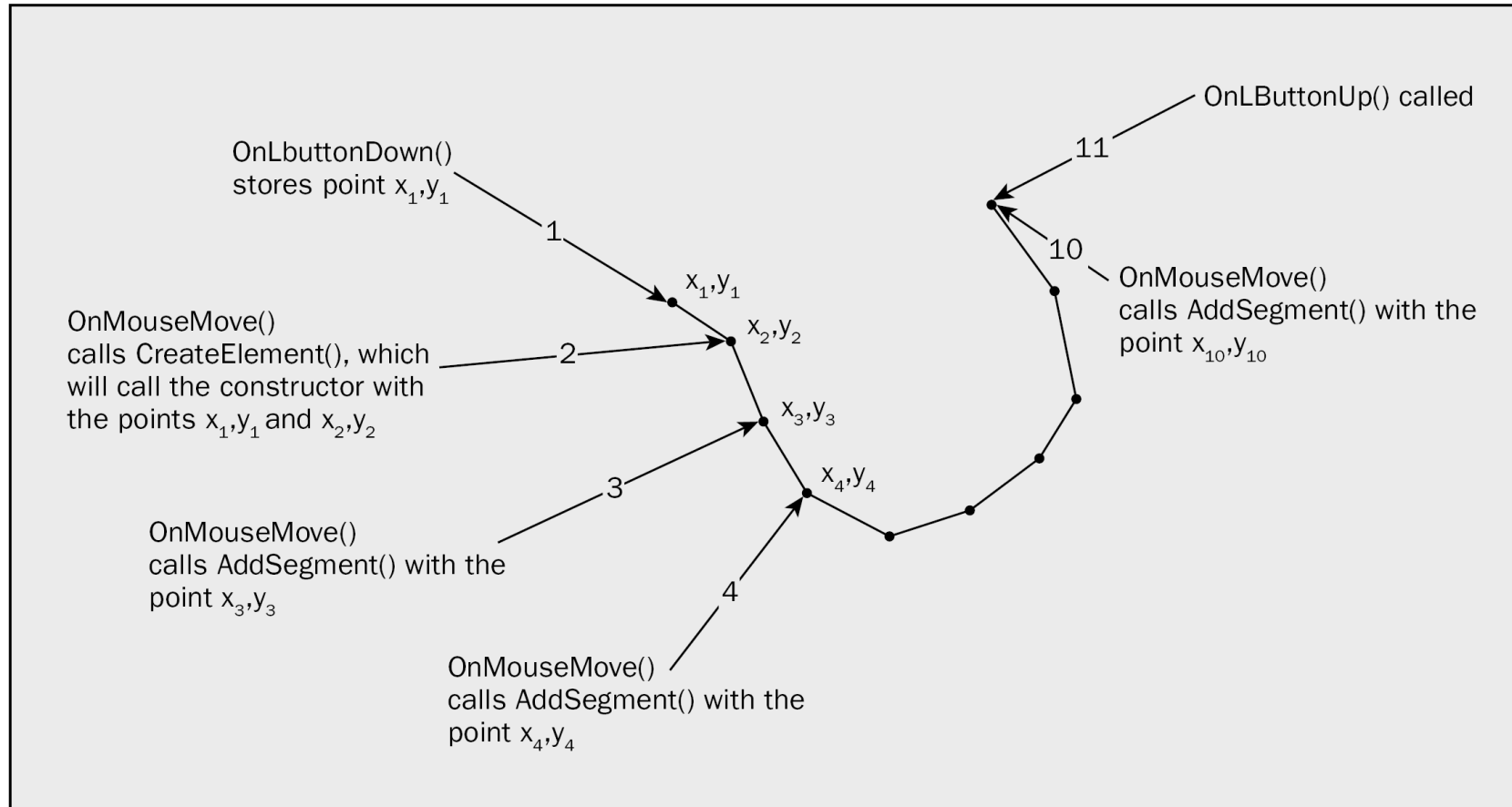


Figure 15-6

# Collection Classes

---

- MFC provides you with a large number of **collection classes** for managing data.
  - They are useful especially when you have no advance knowledge of how many items you will need to manage.
- MFC supports three kinds of collections (three *shapes*), differentiated by the way in which the data items are organized.
  - Array
  - List
  - Map

# Array

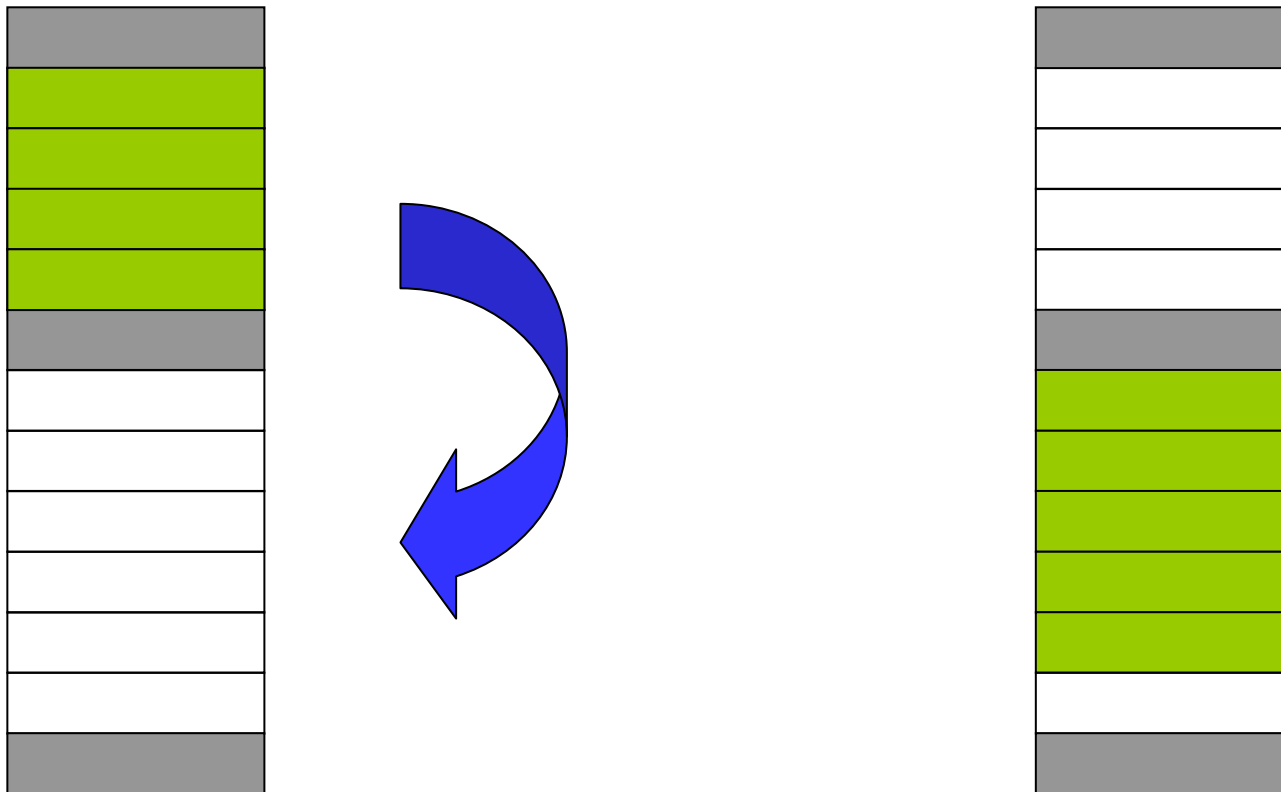
---

- Elements in array collections are indexed from 0.
- Template class: CArray
  - `CArray<CPoint, CPoint&> PointArray;`
    - To avoid the overhead in copying objects when passed by value, the second argument is usually a reference.

# The CArray Template Class (1)

---

- An array collection can automatically grow to accommodate more data items.



# Figure 16-1 (P.912)

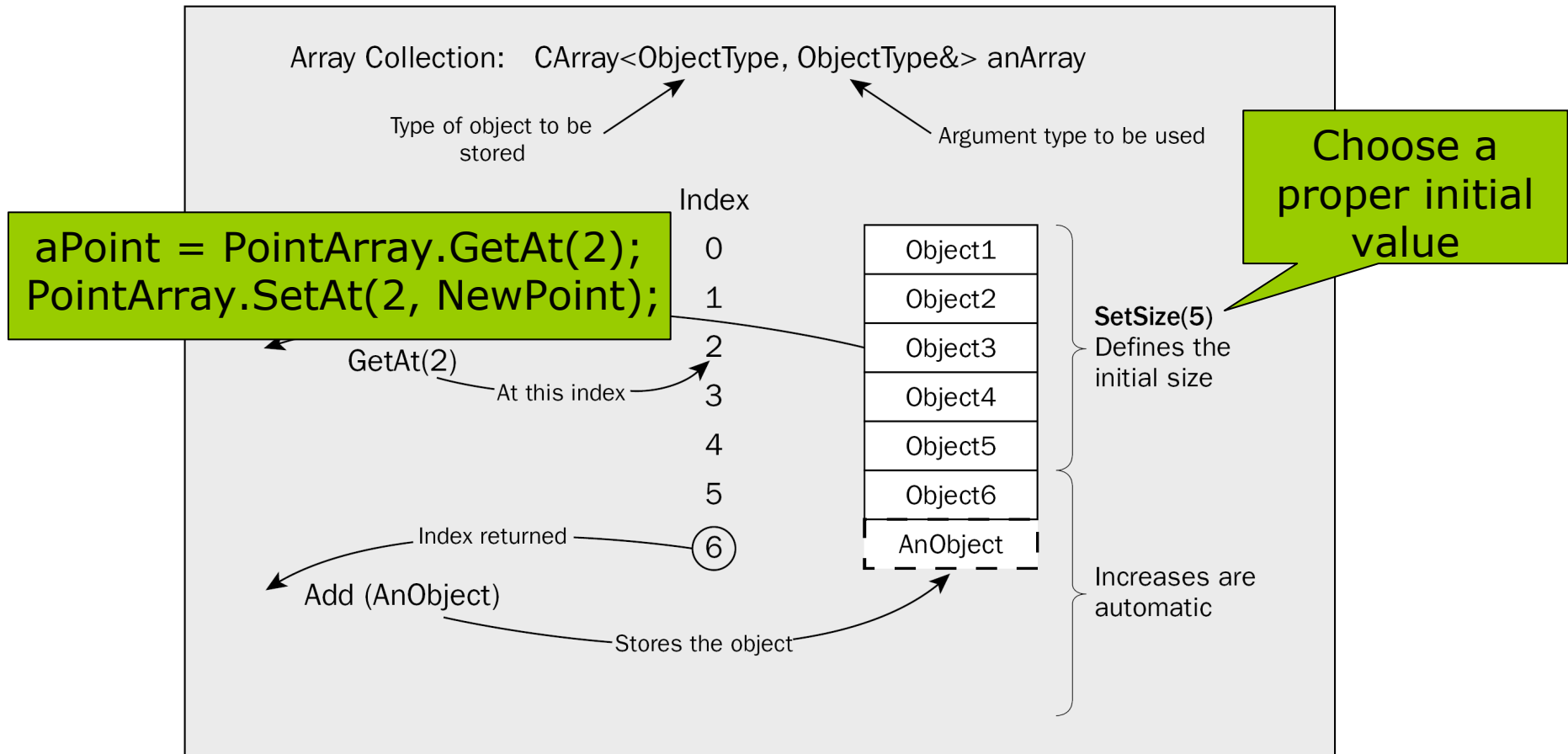


Figure 15-1

# The CList Template Class

---

- A doubly linked list
  - It has backward and forward pointing links.
  - It can be searched in either direction.
  - It grows automatically when required.
    - It is fast in adding items, compared with CArray.
    - If there are lots of data items in the list, it can be slow in searching for an item.
- `CList<ObjectType, ObjectType&> aList;`
  - `CList<CPoint, CPoint&> PointList;`



# Adding Elements to a List

- Both the `AddHead()` and `AddTail()` functions return a value of type `POSITION`, which specifies the position of the inserted object in the list.

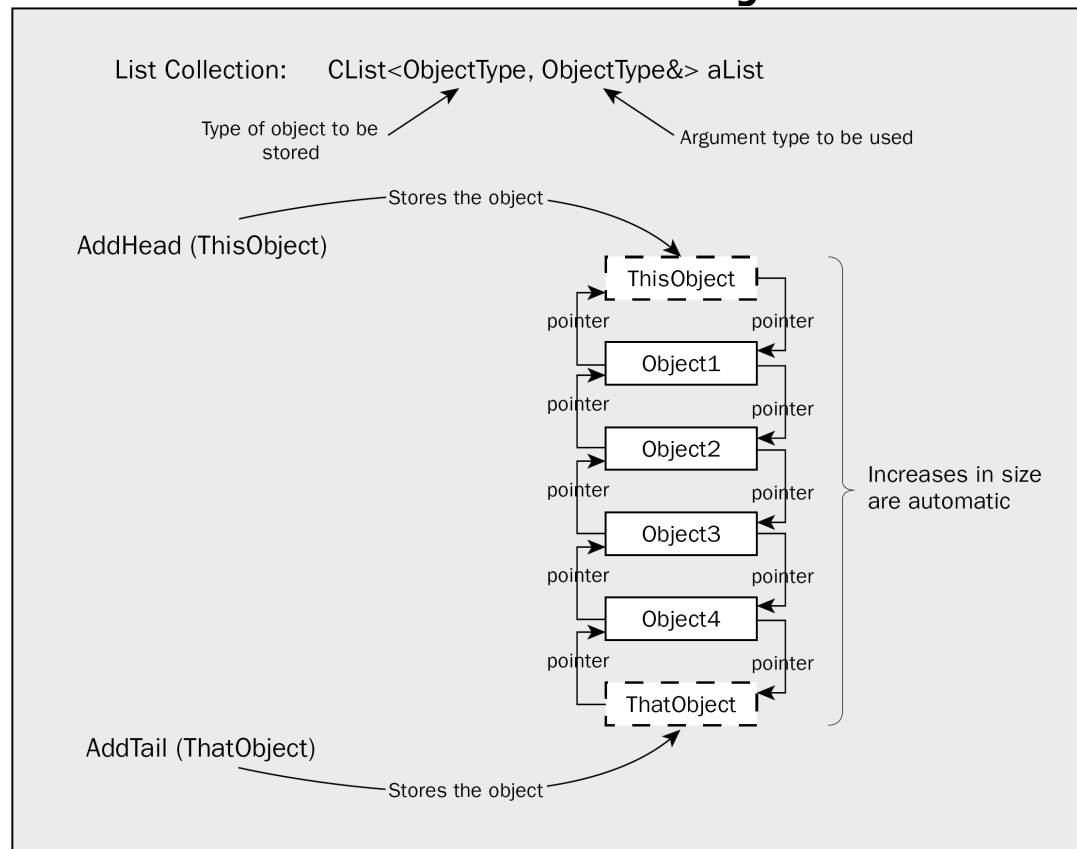


Figure 15-2

# Retrieving Elements in a List

- GetAt()
- GetNext()
- GetPrev()

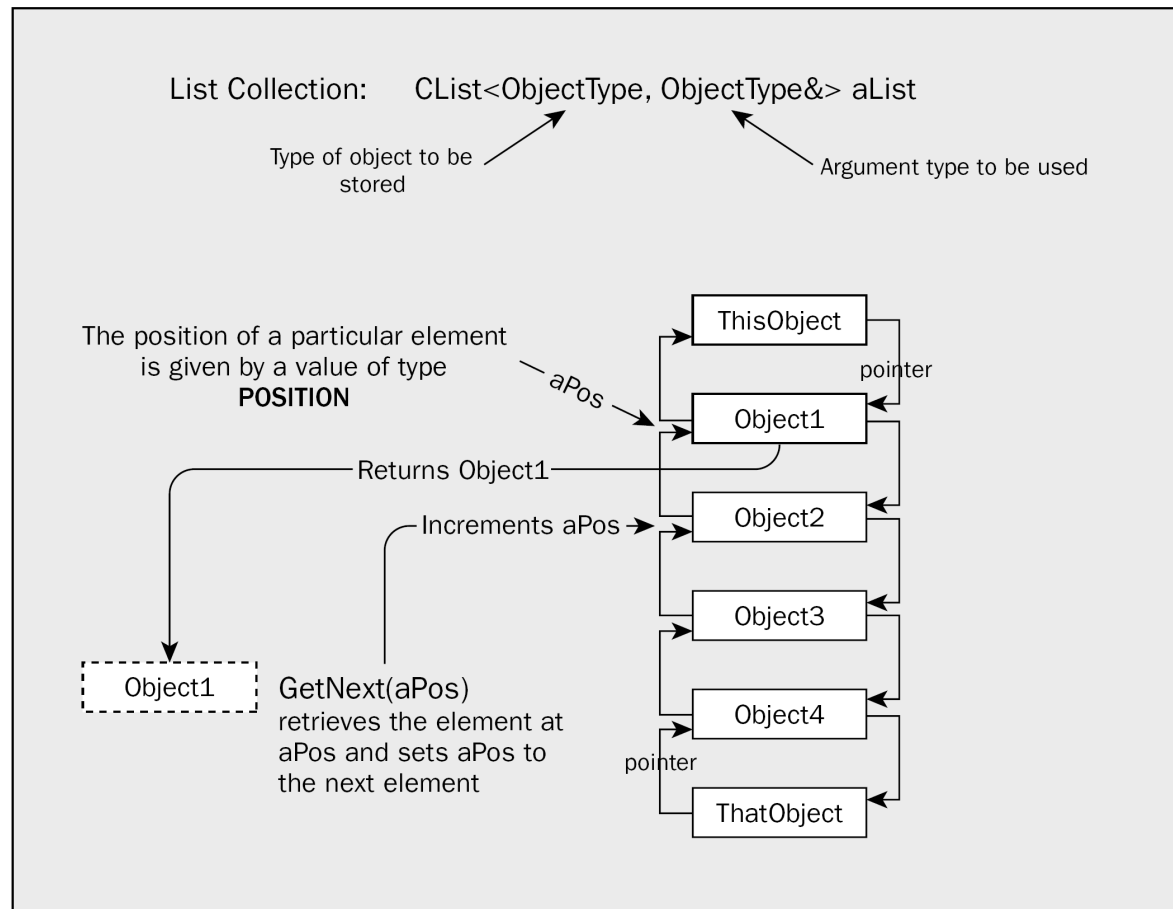


Figure 15-3

# Iterating through a List

---

- GetHeadPosition() & GetTailPosition()
- GetNext() & GetPrev()
  - The position variable will become NULL if you use GetNext() to retrieve the last object.

```
CPoint CurrentPoint(0,0);  
// Get the position of the first list element  
POSITION aPosition = PointList.GetHeadPosition();  
  
while (aPosition) // Loop while aPosition is not NULL  
{  
    CurrentPoint = PointList.GetNext(aPosition);  
    // Process the current object  
}
```

# Modifying a List

---

- `InsertBefore()`
  - `PointList.InsertBefore(aPosition, ThePoint);`
- `InsertAfter()`
- `SetAt()`
  - `PointList.SetAt(aPosition, aPoint);`

# Searching a List

---

## □ Find()

- POSITION `aPosition = PointList.Find(ThePoint);`
  - By default, this function only compares the address of the argument with the address of each object in the list.
  - This implies that if the search is to be successful, the argument must actually be an element in the list – not a copy.

## □ FindIndex()

- You can also obtain the position of an element in a list by using an index value.
  - The first element is at index 0, the second at index 1, and so on.

## □ GetCount()

- Return how many objects are there in a list.

# Removing Objects from a List

---

- RemoveHead()
  - if(!PointList.IsEmpty())
  - PointList.RemoveHead();
- RemoveAt()
  - PointList.RemoveAt(aPosition);
- RemoveAll()
  - PointList.RemoveAll();

# The CMap Template Class

---

- ❑ A map stores an **object and key** combination.
  - This technique is sometimes called hashing.
  - A key is used to retrieve the item from the map, so it should be unique.
    - ❑ It is fast in storing data items and also in searching, because a key takes you directly to the item.
    - ❑ For sequential access, arrays or lists are faster.
  
- ❑ Four arguments are needed to declare a map:
  - `CMap<LONG, LONG&, CPoint, CPoint&> PointMap;`

# Retrieving Items in a Map

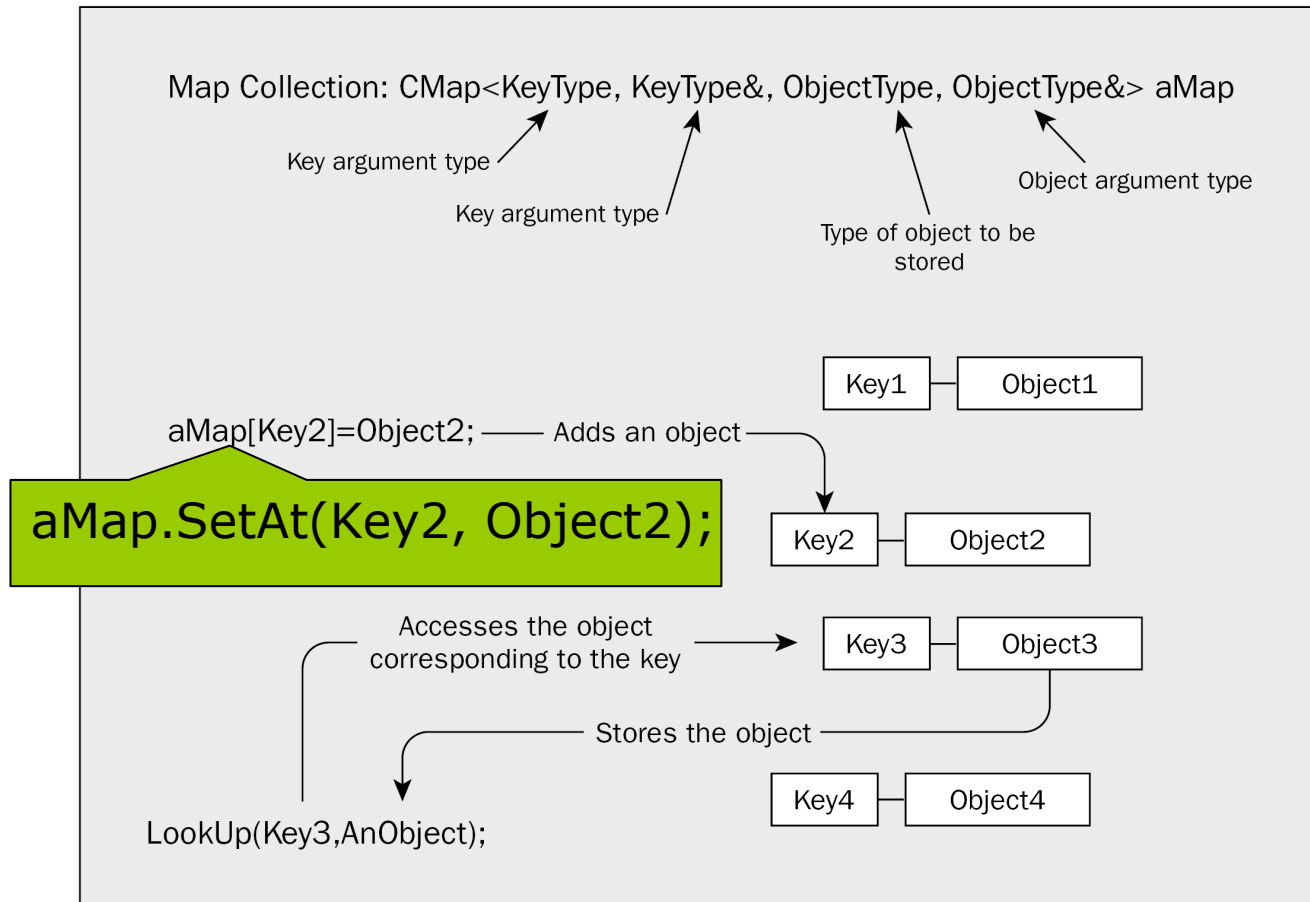


Figure 15-4



# Using the CList Template Class

---

- A curve is defined by two or more points.
  - Storing these points in a list would be a natural solution.

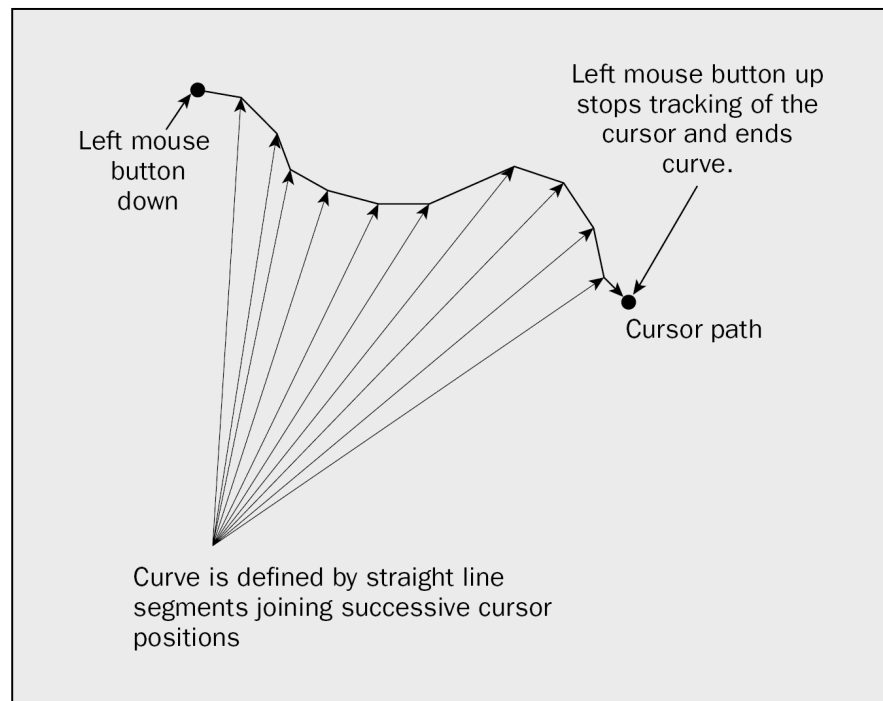


Figure 14-10

# Define a CList collection class object as a member of the CCurve class

---

```
// Elements.h
class CCurve : public CElement
{
protected:
    CCurve(void);
    CList<CPoint, CPoint> m_PointList; // Type safe point list

public:
    ~CCurve(void);
    virtual void Draw(CDC* pDC); // Fucntion to display a curve

    // Constructor for a curve object
    CCurve(CPoint FirstPoint, CPoint SecondPoint, COLORREF aColor);

    void AddSegment(CPoint& aPoint); // Add a segment to the curve
};
```

# CSketcherView.cpp

---

- Modify the definition of the CreateElement() function to call the CCurve class constructor with correct arguments.

```
case CURVE:  
    return new CCurve(m_FirstPoint,  
        m_SecondPoint, pDoc->GetElementColor());
```

# OnMouseMove ( )

---

## □ P.926 (compare with P.889)

```
if (CURVE == GetDocument()->GetElementType())
// Is it a curve?
{
    static_cast<CCurve*>(m_pTempElement)->
        AddSegment(m_SecondPoint);
    m_pTempElement->Draw(&aDC);
    return;
}
```

```
aDC.SetROP2(R2_NOTXORPEN); // Set the drawing mode
```

- Move the call to SetROP2() to a position after the code processing a curve.

# CCurve Constructor

---

```
CCurve::CCurve(CPoint FirstPoint, CPoint
    SecondPoint, COLORREF aColor)
{
    m_PointList.AddTail(FirstPoint);
    m_PointList.AddTail(SecondPoint);
    m_Color = aColor;
    m_Pen = 1;

    m_EnclosingRect = CRect(FirstPoint,
        SecondPoint);
}
```

# Enclosing Rectangle

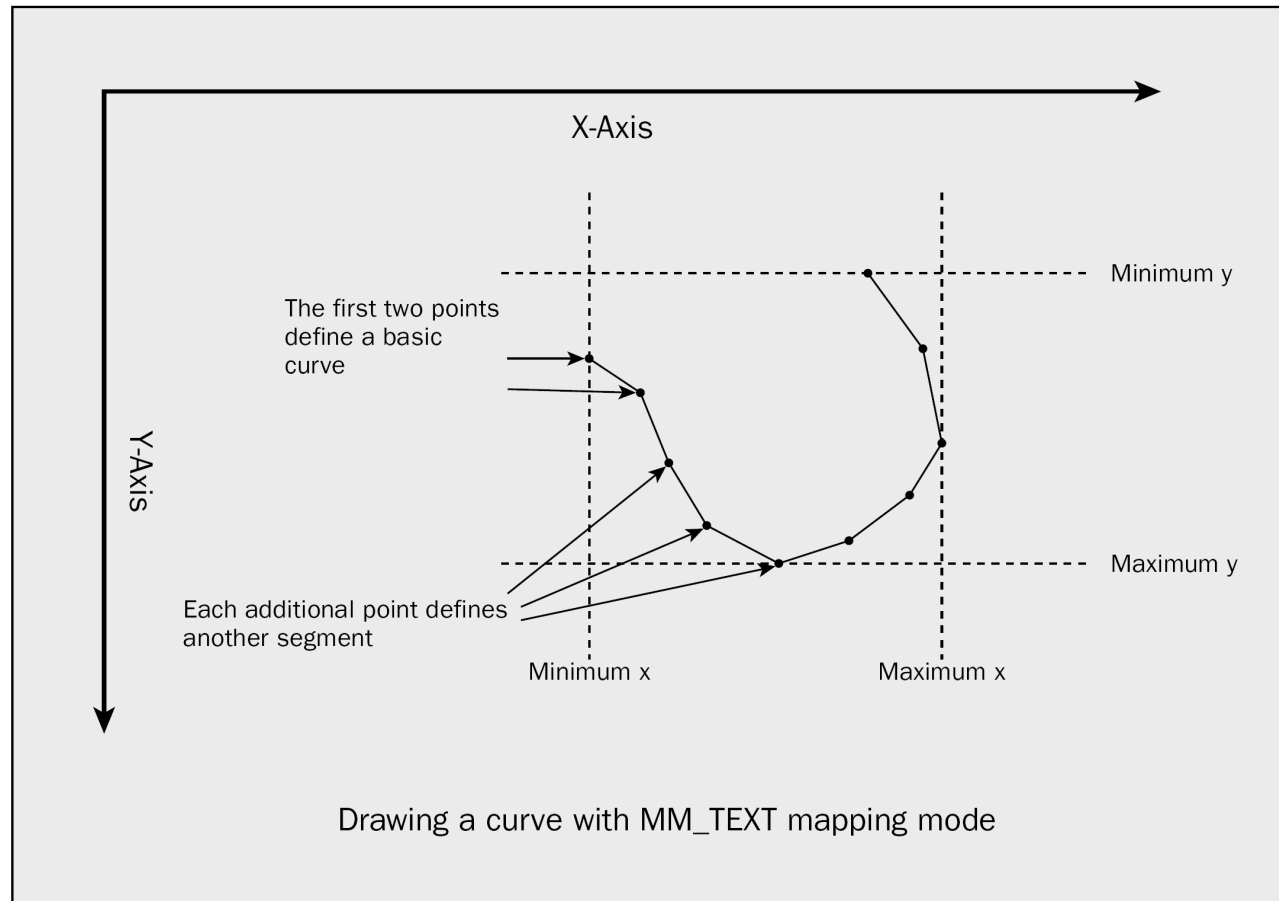


Figure 15-5

# AddSegment ( )

---

```
void CCurve::AddSegment(CPoint& aPoint)
{
    m_PointList.AddTail(aPoint);

    m_EnclosingRect =
        CRect(min(aPoint.x, m_EnclosingRect.left),
              min(aPoint.y, m_EnclosingRect.top),
              max(aPoint.x, m_EnclosingRect.right),
              max(aPoint.y, m_EnclosingRect.bottom));
}
```

# Draw ( )

---

## ▣ P.928

```
POSITION aPosition =  
    m_PointList.GetHeadPosition();  
  
    if (aPosition)  
        pDC->MoveTo(  
            m_PointList.GetNext(aPosition));  
  
    while(aPosition)  
        pDC->LineTo(  
            m_PointList.GetNext(aPosition));
```



# Exercising the CCurve Class

---



Figure 15-7