Data Structures and Algorithms

Chapter 2 Linear Structures

Dr. Zhiqiang Liu

School of Software and Microelectronics, Northwest Polytechnical University

Review

- Four relationships in data structures
  - None
  - Linear: one to one
  - Hierarchical: one to many
  - Graph: many to many

Curriculum	Course name	Period
024020	Data Structure	64
024024	Operating System	48
024026	Database Theory	48

Outline

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

Next Section

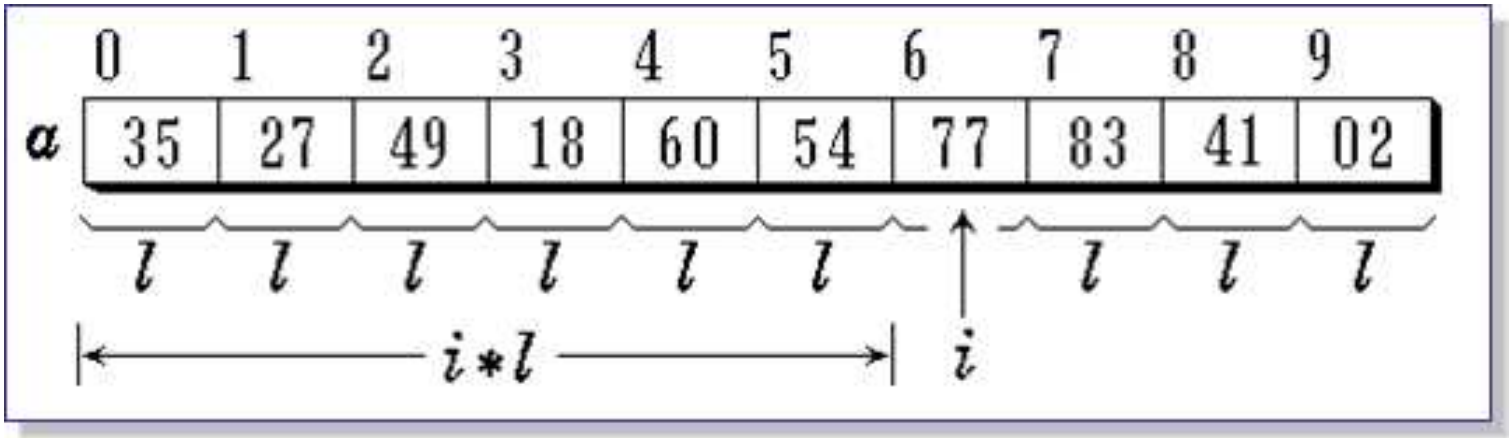
1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

Next Subsection

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.1.1 Arrays

- Array: a set of index and value
  - Data structure
    For each index, there is a value associated with that index.
  - Representation (possible)
    implemented by using consecutive memory.



Abstract Data Type(ADT) of Array

Structure Array is {

objects: A set of pairs <index, value> where for each value of index there is a value from the set item. Index is a finite ordered set of one or more dimensions, for example, 0, . . . , n-1 for one dimension, {(0,0),(0,1),(0,2),(1,0),(1,1),(1,2),(2,0),(2
for two dimensions,etc.

Functions:

for all A $\in$ Array, i $\in$ index, x $\in$ item, j, size $\in$ integer

Array Create(j, list) ::=	**return** an array of j dimensions where list is a j-tuple whose ith element is the size of the ith dimension. Items are undefined.
Item Retrieve(A, i) ::=	**if** (i $\in$ index) **return** the item associated with index value i in array A **else return** error

ADT of Array

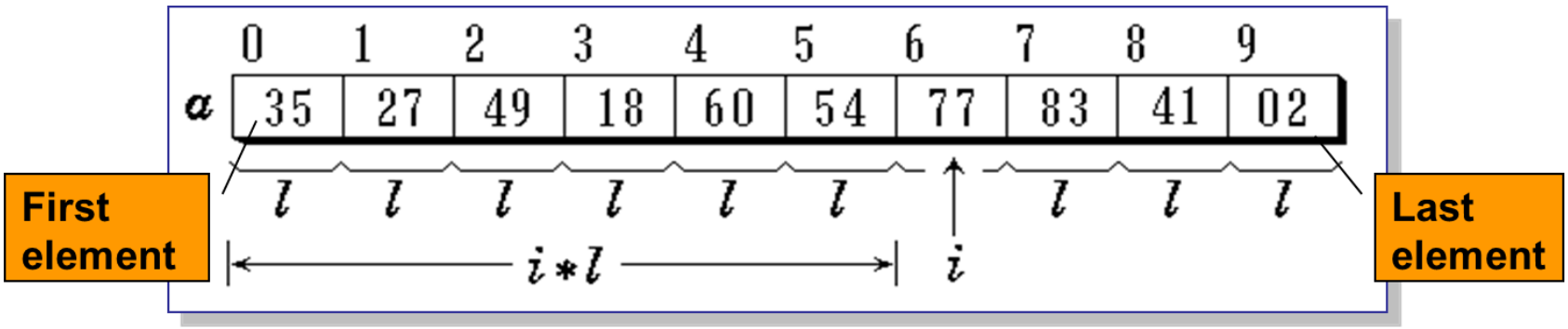
Array Store(A, i, x) ::=

if (i in index) **return** an array that is identical to array A except the new pair <i, x> has been inserted **else** **return** error

end array
}

2.1.1 Arrays

- **Characteristics** of one dimensional array
  - In contiguous storage, also called as Vector
  - Except the first element of the array, each element has and only has one predecessor
  - Except the last element of the array, each element has and only has one successor



Arrays in C++

```
1 int list[5], *plist[5];
2 /*five integers list[0], list[1], list[2], list[3], list
      [4]*/
3 list[5]
4
5 /*five pointers to integers plist[0], plist[1], plist
      [2], plist[3], plist[4]*/
6 *plist[5]
```

implementation of 1-D array

$list[0]$ $\quad baseaddress = \alpha$

$list[1]$ $\quad \alpha + sizeof(int)$

$list[2]$ $\quad \alpha + 2 * sizeof(int)$

$list[3]$ $\quad \alpha + 3 * sizeof(int)$

$list[4]$ $\quad \alpha + 4 * sizeof(int)$

2.1.1 Arrays

Define and initialization of array

```
1  #include <iostream.h>
2  // Define the element of array
3  class szcl
4  {
5      int e;
6      public:
7        szcl () { e = 0; }
8        szcl (int value) { e = value; }
9        int get_value () { return e; }
10 };
```

2.1.1 Arrays

```
1  main ()
2  {
3      szcl a1[3] = { 3, 5, 7 }, *elem;
4      for ( int i=0, i<3, i++ )
5      {
6          //print static array
7          cout << a1[i].get_value ( ) << “\”n;
8      }
9      elem = &a1;
10     for ( int i=0, i<3, i++ ) {
11         //print dynamic array
12         cout << elem->get_value( ) << “\”n;
13         elem++;
14      }
15      return 0;
16 }
```

2.1.1 Arrays

```
1  #include <iostream.h>
2  #include <stdlib.h>
3  template <class Type> class Array {
4      Type *elements;  //Storage space of array
5      int ArraySize;
6      void getArray (); //Create space
7  public:
8      Array(int Size=DefaultSize );
9      Array(const Array<Type>&x );
10     ~Array( ) { delete []elements;}
11     Array<Type> &operator=(const Array<Type> &A);
12     Type& operator [] ( int i );
13     Type* operator () const { return elements; }
14     int Length () const { return ArraySize; }
15     void ReSize ( int sz );
16 }
```

2.1.1 Arrays

```
1  template <class Type>
2  void Array<Type>::getArray ( ) {
3    //Create store space of array
4    elements = new Type[ArraySize];
5    if ( elements == 0 )
6      cerr<<"Memory Allocation Error"<<endl;
7  }
8  template <class Type>
9  void Array<Type>::Array ( int sz ) {
10   if ( sz <= 0 ) {
11     cerr << "Invalid Array Size" << endl; return;
12   }
13   ArraySize = sz;
14   getArray ( );
15 }
```

2.1.1 Arrays

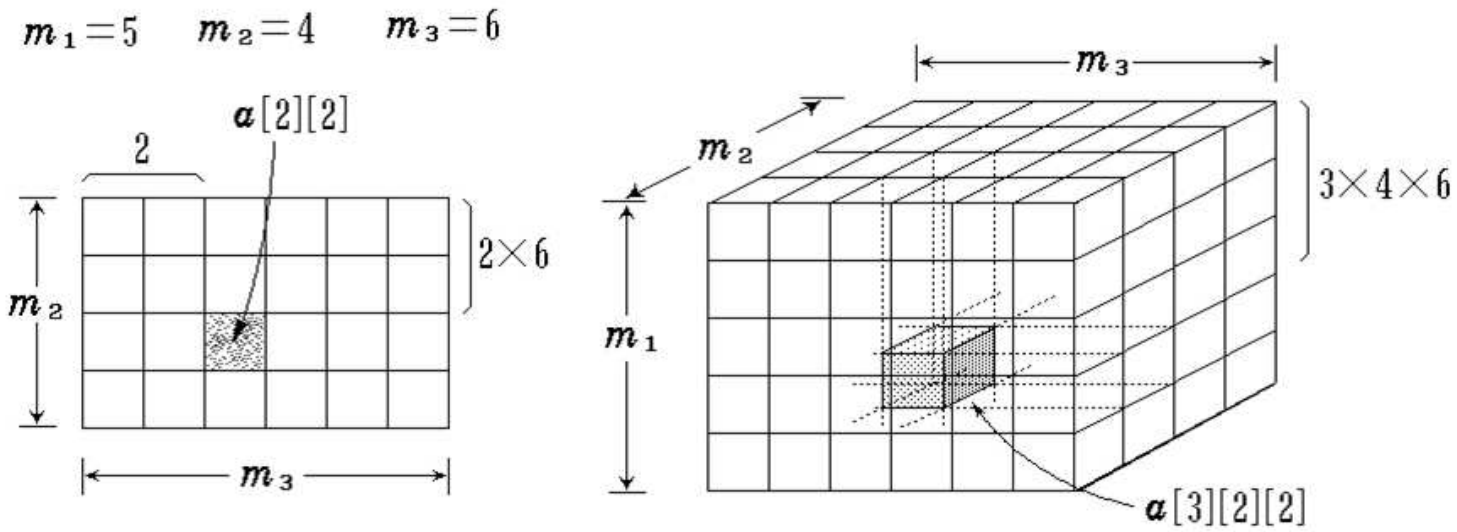
```
1  template <class Type> Array<Type>::
2  Array ( const Array<Type> & x ) {
3    //Copy constructure
4    int n = ArraySize = x.ArraySize;
5    elements = new Type[n];
6    if ( elements == 0 )
7      cerr << "Memory␣Allocation␣Error" << endl;
8      Type *srcptr = x.elements;
9      Type *destptr = elements;
10     while ( n-- ) * destptr++ = * srcptr++;
11 }
12 template <class Type>
13 Type & Array<Type>::operator [ ] ( int i ) {
14   if ( i < 0 || i > ArraySize-1 )
15     cerr << "Index␣out␣of␣Range" << endl;
16   return element[i];
17 }
```

2.1.1 Arrays

```
1  template <class Type>
2  void Array<Type>::Resize (int sz) {
3      if ( sz >= 0 && sz != ArraySize ) {
4          Type * newarray = new Type[sz];
5          if ( newarray == 0 )
6              cerr << "Memory Allocation Error" <<
                   endl;
7          int n = ( sz <= ArraySize ) ? sz :
                  ArraySize;
8          Type *srcptr = elements;
9          Type *destptr = newarray;
10         while ( n-- ) * destptr++ = * srcptr++;
11         delete [ ] elements;
12         elements = newarray; ArraySize = sz;
13     }
14 }
```

2.1.1 Arrays

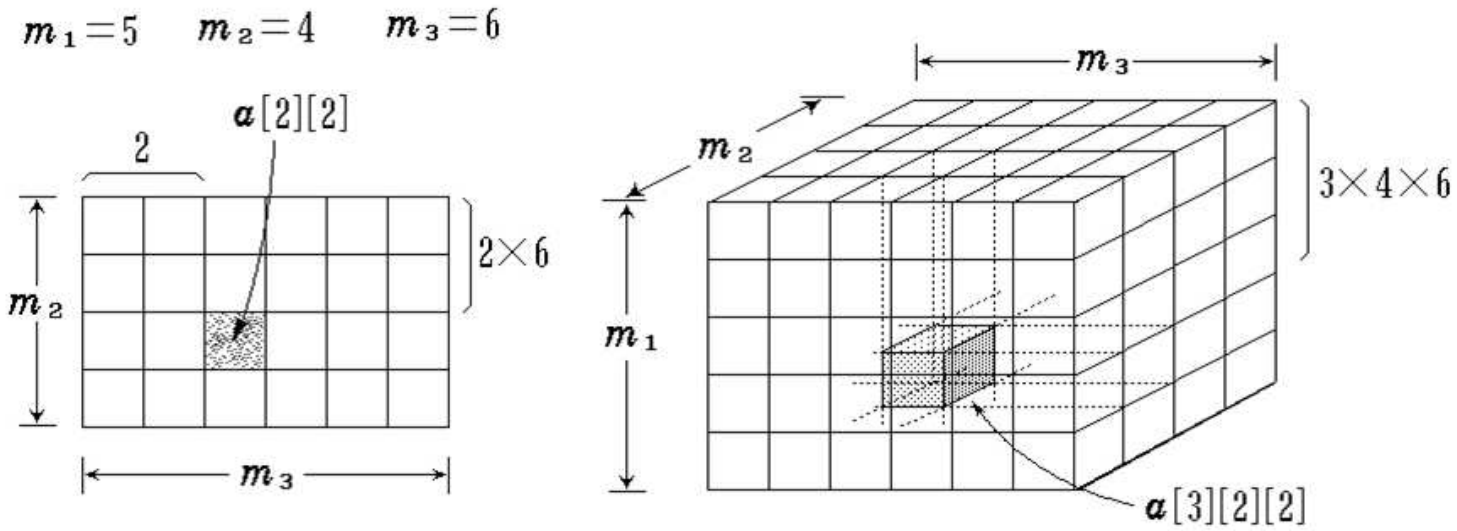
2D Array



Row subscript i, Column subscript j

2.1.1 Arrays

3D Array



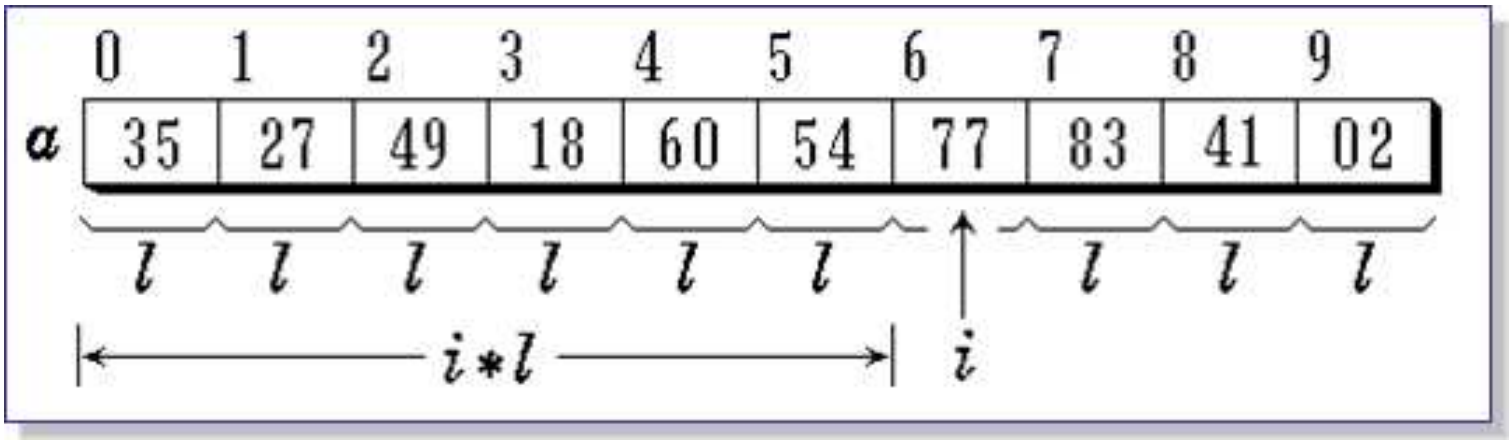
Page i, Row subscript j, Column subscript k

2.1.1 Arrays

Sequential Storage of Arrays

- 1D array

$$LOC(i) = \begin{cases} \alpha & \text{i=0} \\ LOC(i-1) + l & \text{i>0} \end{cases}$$

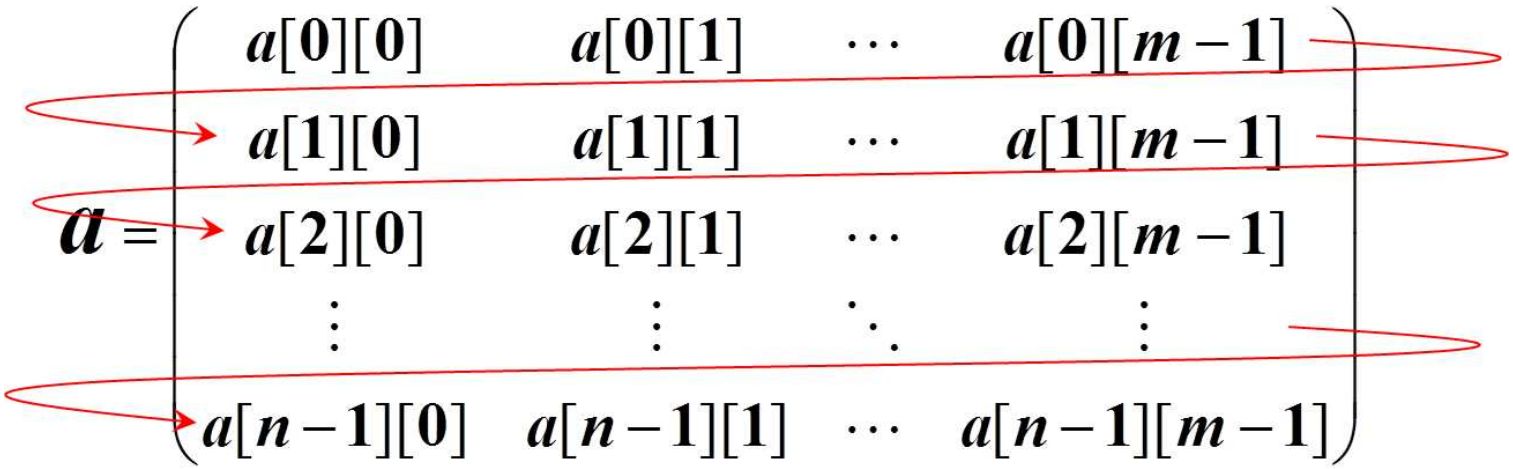


LOC(i) = LOC(i-1) + l =α+ i*l

2.1.1 Arrays

Sequential Storage of Arrays

- 2D array



Row-first:

LOC (j, k) = a + (j * m + k) * l

2.1.1 Arrays

Sequential Storage of Arrays

- N-D array
  - The dimensions are $m_1, m_2, m_3, \ldots, m_n$
  - The data element with subscripts $(i_1, i_2, i_3, \ldots, i_n)$ is in the space:

$$LOC(i_1, i_2, \ldots, i_n) = a + (i_1 * m_2 * m_3 * \ldots * m_n + i_2 * m_3 * m_4 * \ldots * m_n + \ldots\ldots + i_{n-1} * m_n + i_n) * l$$

$$= a + (\sum_{j=1}^{n-1} * \prod_{k=j+1}^{n} m_k + i_n) * l$$

Next Subsection

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.1.2 Sequential List (Sequence)

- **Definition and Property of Sequential List**
  - Definition: A list is a finite, ordered sequence of data items

    $$(a_1, a_2, \ldots, a_n)$$

    where a_1 is the item or element of list , n is the length of the list
  - Property: sequential access (put and get)
  - Important concept: Each element has a position.
  - Traversal:
    - **from the first to the last**
    - **from the last to the first**
    - **from the intermediate position to the head or the end**

```
1  // Definition of Sequential List
2  template <class Type>   class SeqList {
3   private:
4         // Array to store the sequential list
5         Type *data;
6         // Maximize the size of list
7         int MaxSize;
8         // Position of last item
9         int last;
10  public:
11        SeqList ( int MaxSize = defaultSize );
12        ~SeqList ( ) { delete [ ] data; }
13        int Length ( ) const { return last+1; }
14        int Find ( Type & x ) const;
15        int IsIn ( Type & x );
16        int Insert ( Type & x, int i );
17        int Remove ( Type & x );
18        int Next ( Type & x ) ;
19        int Prior ( Type & x ) ;
20        int IsEmpty ( ) { return last ==-1; }
```

```
21          int IsFull()
22          {
23              return last == MaxSize-1;
24          }
25
26          Type Get ( int i )
27          {
28                return i < 0 || i > last ? NULL :
                      data[i];
29          }
30  }
```

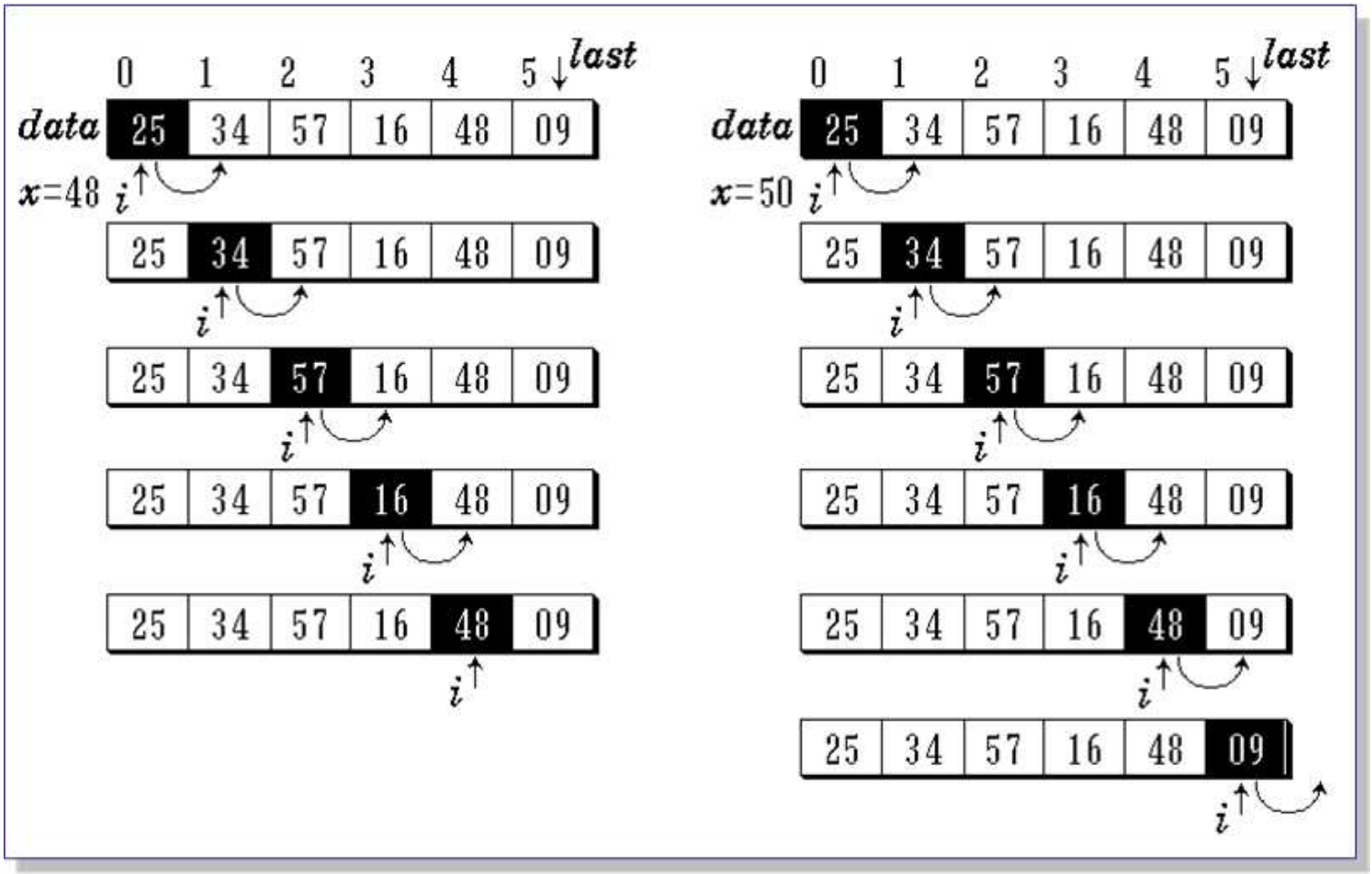
2.1.2 Sequential List (Sequence)

Implementation of Methods for Sequential List

```
1  template <class Type>
2   SeqList<Type> :: SeqList ( int sz ) {
3   // Constructor Function
4       if ( sz > 0 )
5       {
6           MaxSize = sz;  last = -1;
7           data = new Type[MaxSize];
8           if ( data == NULL )
9           {
10              MaxSize = 0;   last = -1;
11              return;
12          }
13      }
14  }
```

2.1.2 Sequential List (Sequence)

Details of Searching in the List



$x = 48$ $x = 50$

2.1.2 Sequential List (Sequence)

```
1  template <class Type>
2  int SeqList<Type> :: Find ( Type & x ) const {
3  // Searching Function: try to find out the
       position of x
4      int  i = 0;
5
6      while ( i <= last && data[i] != x )
7         i++;
8
9      if ( i > last )
10           return -1;
11     else
12           return i;
13 }
```

2.1.2 Sequential List (Sequence)

Complexity Analysis of Searching

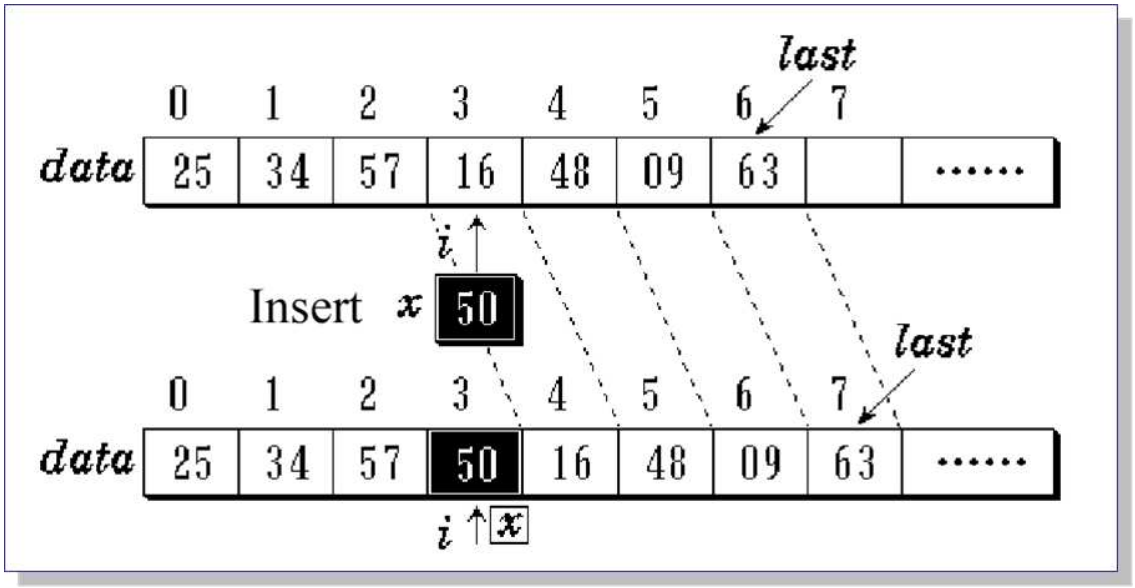
- If success
- Average Comparison Number (ACN) is

$$ACN = \frac{1}{n}\sum_{i=0}^{n-1} = \frac{1}{n}(1+2+\ldots+n) = \frac{1}{n} * \frac{(n+1)*n}{2} = \frac{n+1}{2}$$

- If fail to search x, actual comparison number is n

2.1.2 Sequential List (Sequence)

Insert an item into the List



- Average Movement Number (AMN) is

$$AMN = \frac{1}{n+1}\sum_{i=0}^{n}(n-i) = \frac{1}{n+1}(n+\ldots+1+0) = \frac{1}{n+1}\frac{n*(n+1)}{2} = \frac{n}{2}$$

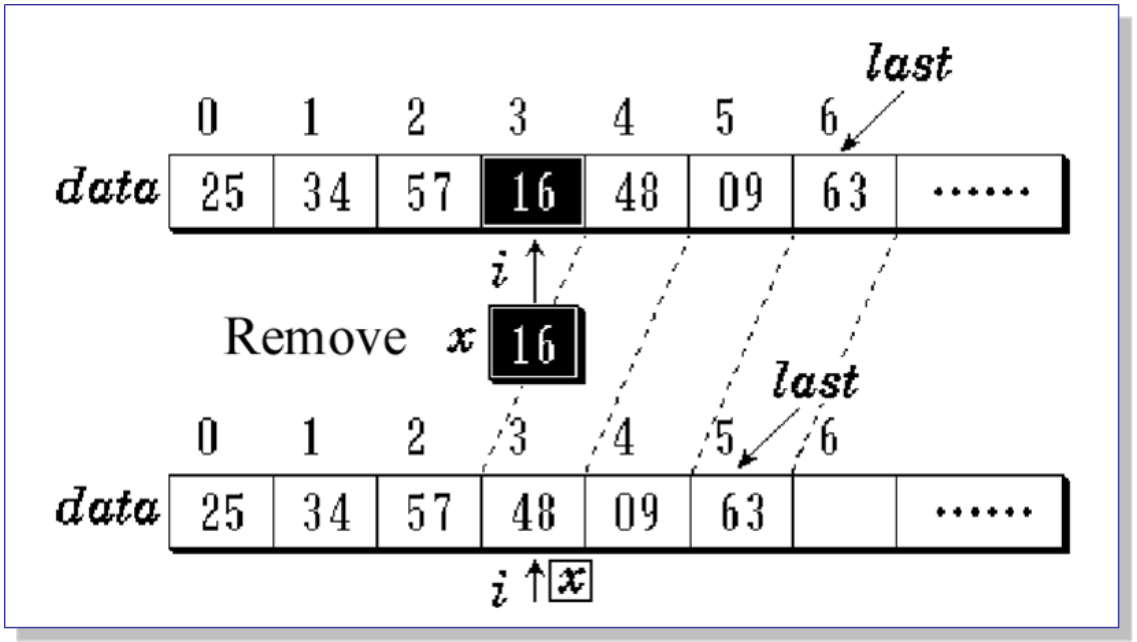
2.1.2 Sequential List (Sequence)

Insert an item into the List

```
1  template <class Type>
2  int SeqList<Type> :: Insert ( Type & x, int i )  {
3  //Insert new item (x) before pos i in the list
4      if (i < 0 || i > last+1 || last == MaxSize-1)
5      {
6          return 0;      // Fail to insert
7      } else {
8        last++;
9        for(int j = last; j > i; j--)//Move elements
10             data[j] = data[j -1];
11       data[i] = x;
12       return 1; // Success to insert
13     }
14 }
```

2.1.2 Sequential List (Sequence)

Remove an item from the List



- Average Movement Number (AMN) is

$$ACN = \frac{1}{n}\sum_{i=0}^{n-1}(n-i-1) = \frac{1}{n}\frac{n*(n-1)}{2} = \frac{n-1}{2}$$

2.1.2 Sequential List (Sequence)

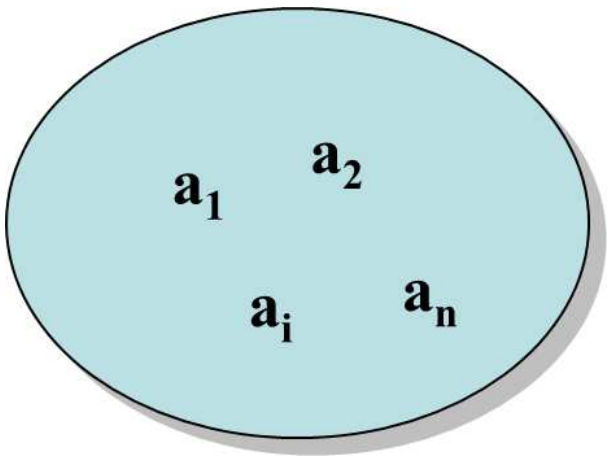
Remove an item from the List

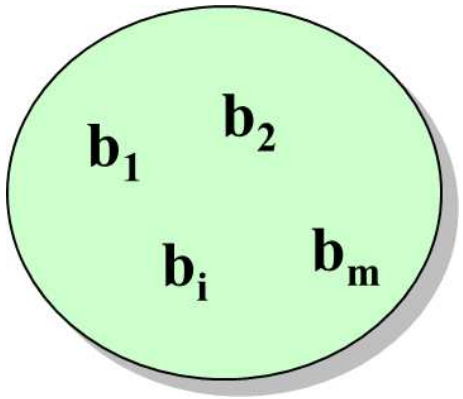
```
1  template <class Type>
2  int SeqList<Type> :: Remove ( Type & x ) {
3  // Remove existed item x from the list
4      int i = Find (x);          // Search x in the list
5      if ( i >= 0 ) {
6          last-- ;
7          for ( int j = i; j <= last; j++ )
8               data[j] = data[j+1];  // Move elements
9          return 1;             // Success to remove x
10     }
11     return 0;                 // No removal if no item x
12 }
```

2.1.2 Sequential List (Sequence)

Application of Sequential List (1)

- Union of two sets: $A = A \cup B$





Union of two sets

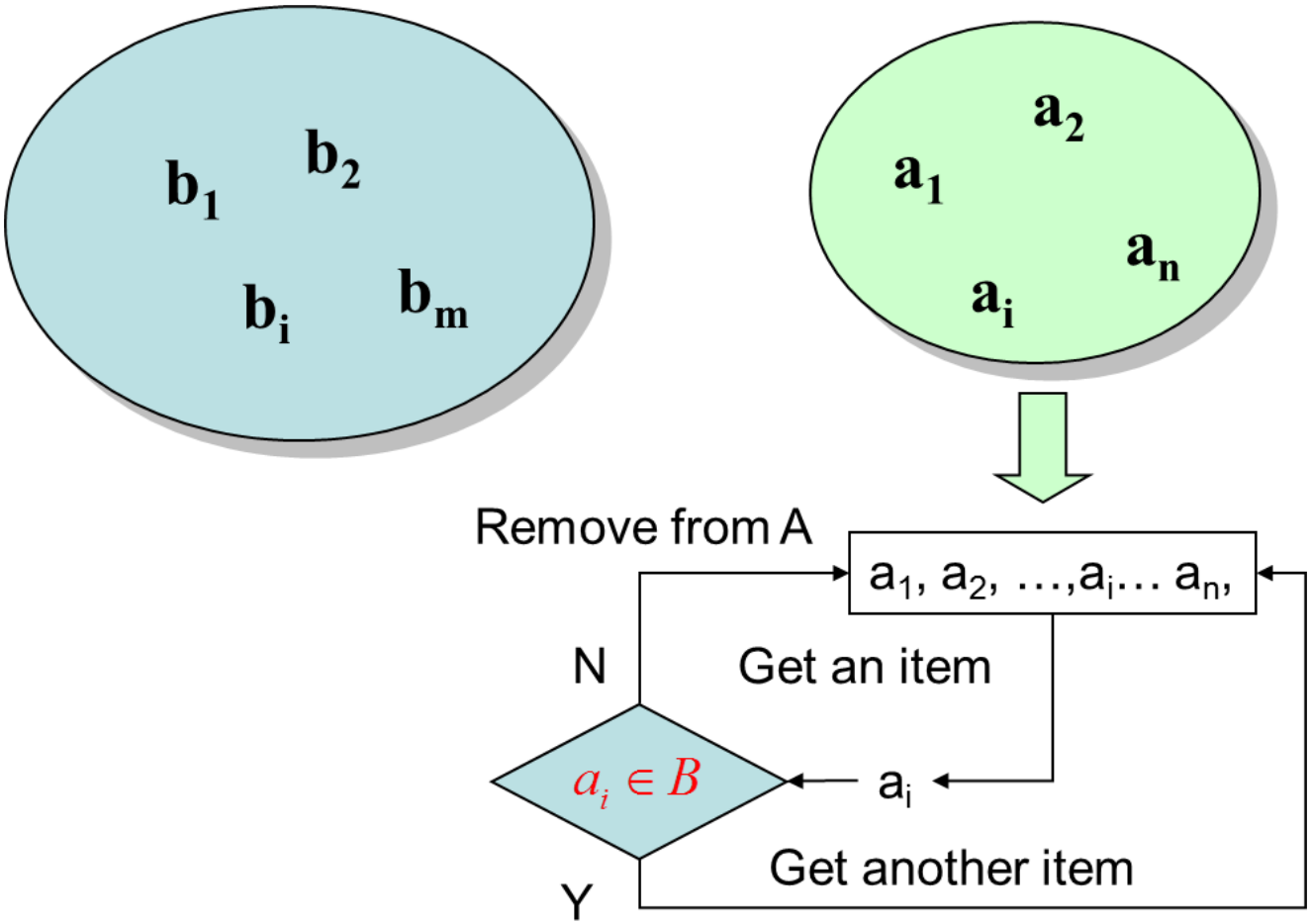
```
1  template <class Type>
2  void Union ( SeqList<Type> & LA, SeqList<Type> & LB ) {
3      int n = LA.Length ( );
4      int m = LB.Length ( );
5      for ( int i = 1; i <= m; i++ )
6      {
7          // Get an item x from Set LB
8          Type x = LB.Get(i);
9          // Search x in Set LA
10         int k = LA.Find (x);
11         // if not found, insert  x into LA
12         if ( k == -1 )
13         {
14             LA.Insert (x,  n+1);  n++;
15         }
16     }
17 }
```

How about time complexity?

2.1.2 Sequential List (Sequence)

Application of Sequential List (2)

- Intersection of two sets: A = A∩B



2.1.2 Sequential List (Sequence)

Intersection of two sets

```
1  template <class Type>
2   void Intersection ( SeqList<Type> & LA,
3                       SeqList<Type> & LB ) {
4       int n = LA.Length ( );
5       int m = LB.Length ( );  int i = 0;
6       while ( i < n ) {
7           Type x = LA.Get (i);// Get an item x from LA
8           int k = LB.Find (x);// Search x in Set LB
9           if ( k == -1 ) { LA.Remove (i);  n--; }
10          else i++;  // if not found, remove x from LA
11      }
12  }
```

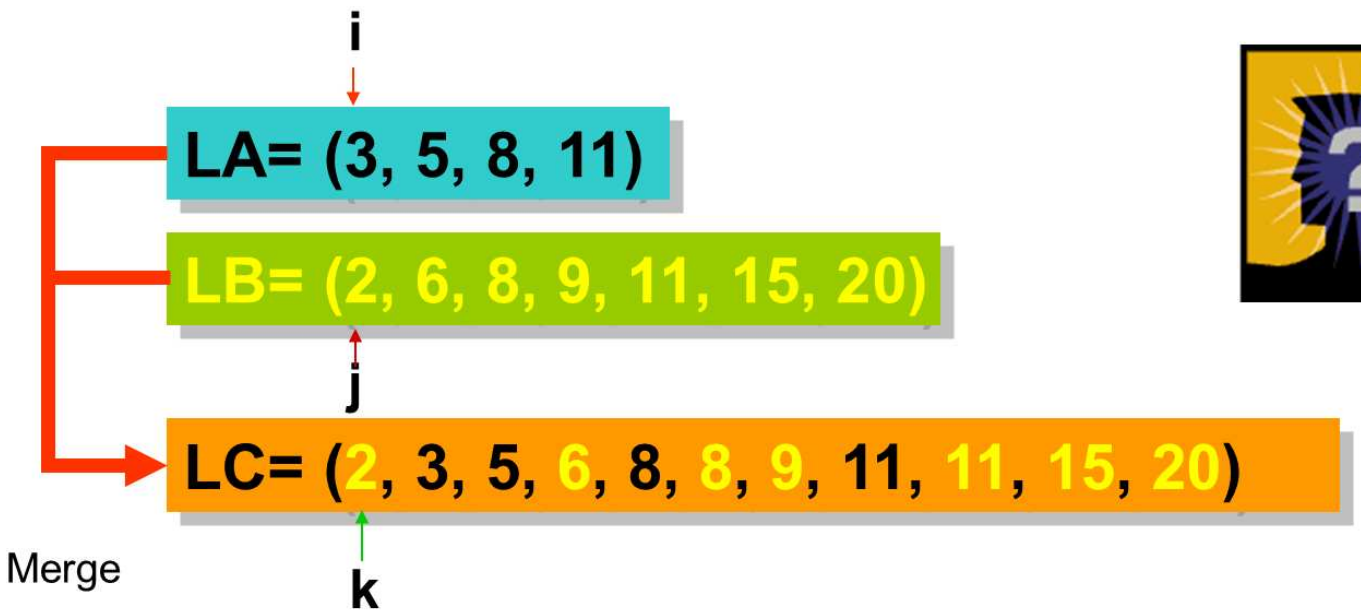
How about time complexity?

2.1.2 Sequential List (Sequence)

Application of Sequential List (3)

- Merge two sorted lists into a new list and the new one is also sorted as before.

Demo



How about time complexity?

```
template <class Type> SeqList &Merge_List
( SeqList <Type> & LA, SeqList <Type> & LB ) {
    int n = LA.Length ( );
    int m = LB.Length ( );
    SeqList LC(m+n);

    int i=j=k=0;
    while ( i < n && j<m) {
        Type x = LA.Get (i);  // Get an item x from LA
        Type y = LB.Get (j); // Get an item y from LB
        if (x <= y )
        {
            LC.Insert (k, x);
            i++; k++;
        }
        else
        {
            LC.Insert (k, y);
            j++; k++;
        }
    }
```

```
22      while ( i < n)
23      { // Insert the remains of LA into LC
24          Type x = LA.Get (i);
25          LC.Insert (k, x);
26          i++; k++;
27      }
28      while (j<m)
29      { // Insert the remains of LB into LC
30          Type y = LB.Get (j);
31          LC.Insert (k, y);
32          j++; k++;
33      }
34      return LC;
35  }
```

Next Subsection

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.1.3 Polynomial

$$P_n(x) = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n = \sum_{i=0}^{n} a_ix^i$$

- N-order polynomial $P_n(x)$ has n+1 items。
  - Coefficients: $a_0, a_1, a_2, \ldots, a_n$
  - Exponentials: 0, 1, 2, . . . , n ascending

2.1.3 Polynomial

ADT of Polynomial

```
1  class Polynomial {
2  public:
3      Polynomial ( );                    //Constructor
4      int operator ! ( );                //Is zero-polynomial
5      float Coef ( int e);
6      int LeadExp ( );                   //return max-exp
7      Polynomial Add (Polynomial poly);
8      Polynomial Mult (Polynomial poly);
9      float Eval ( float x);//compute the value of the PN
10 }
```

To computer the power of x, (Power Class)

```
1  #include <iostream.h>
2   class power {
3       double x; int e;
4       double mul;  //The value of e^x
5   public:
6       power (double val, int exp);  //constructor
7       double get_power ( ) { return mul; }  //Get e^x
8   };
9   power::power (double val, int exp) {
10  //Computer the power of val x^e
11      x = val;    e = exp;    mul = 1.0;
12      if ( exp == 0 ) return;
13      for ( ; exp>0; exp--) mul = mul * x;
14  }
15  main ( ) {
16      power pwr ( 1.5, 2 );
17      cout << pwr.get_power ( ) << "\"n;
18  }
```

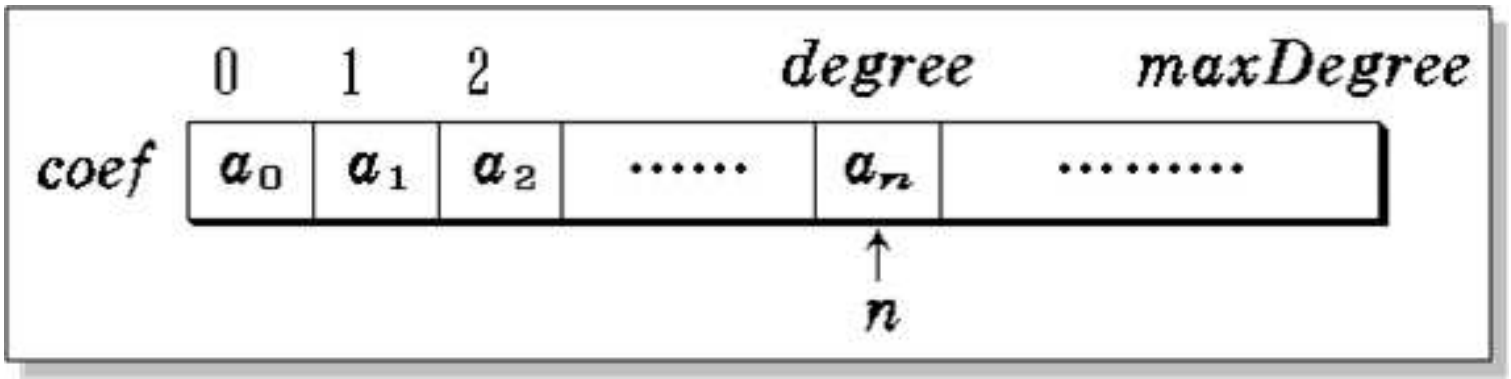
2.1.3 Polynomial

Representation of Polynomial (storage)

1^{st} method:

```
1  private:
2  int degree;
3  float coef [maxDegree+1];
```

$$P_n(x): \quad pl.degree = n$$
$$pl.coef[i] = a_i, 0 \le i \le n$$



2.1.3 Polynomial

```
1  private:
2  int degree;
3  float * coef;
4
5  Polynomial::Polynomial (int sz) {
6      degree = sz;
7      coef = new float [degree + 1];
8          }
```

The 1_{st} storages are NOT suitable for the following case

$$P_101(x) = 3 + 5x^{50} - 14x^{101}$$

2.1.3 Polynomial

2nd method:

```
1 class Polynomial;
2
3 class term {       //item definition
4 friend Polynomial; //PN class is the friend class of
      item class
5 private:
6     float coef;   //coefficient
7     int exp; //exponential
8 };
```

	0	1	2	.....	i	.....	m
coef	a_0	a_1	a_2	.....	a_i	.....	a_m
exp	e_0	e_1	e_2	.....	e_i	.....	e_m

2.1.3 Polynomial

```
1  class Polynomial {          //Polynomial class
2  public:......
3
4  private:
5      static term termArray[MaxTerms];  //items
6      static int free;                  //pos of current
           free space
7      // term Polynomial::termArray[MaxTerms];
8      // int Polynomial::free = 0;
9      int start, finish;          //start and finish pos of
           the items of              //Polynomial
10 }
```

2.1.3 Polynomial

Examples:

Two polynomials are stored in termArray

$$A(x) = 2.0x^{1000} + 1.8$$
$$B(x) = 1.2 + 51.3x^{50} + 3.7x^{101}$$

	A.start ↓	*A.finish* ↓	*B.start* ↓		*B.finish* ↓	*free* ↓ *maxTerms*
coef	1.8	2.0	1.2	51.3	3.7	.....
exp	0	1000	0	50	101	.....

2.1.3 Polynomial

Addition of Polynomials

- Requirement
  - The summarization polynomial is an new one
- Method
  - To traverse two polynomials (A and B) until one of them has been traversed;
  - If the exps are equal, add two coefs.
    - **If the addition of coefs are not equal to 0, new a item and append it into C, otherwise continue to traverse.**
  - If the exps are not equal, add the item whose exp is lower into C.
- If one of A and B has been traversed completely, it is easy to duplicate the remains of another one into C

2.1.3 Polynomial

Application of Sequential List (3)

- Merge two sorted lists into a new list and the new one is also sorted as before.

Demo

2.1.3 Polynomial

```
template <class Type>
SeqList &Merge_List ( SeqList <Type> & LA, SeqList <Type> & LB ) {
    int n = LA.Length ( );
    int m = LB.Length ( );
    SeqList LC(m+n);

    int i=j=k=0;
    while ( i < n && j<m) {
        Type x = LA.Get (i);      // Get an item x from LA
        Type y = LB.Get (j);      // Get an item y from LB
        if (x <= y )
        {   LC.Insert (k, x);  i++; k++;}    // Insert x into LC
        else
        {   LC.Insert (k, y);  j++; k++;}    // Insert y into LC
    }
    while ( i < n) {          // Insert the remains of LA into LC
        Type x = LA.Get (i);
        LC.Insert (k, x);
        i++;
        k++;
    }
    while (j<m) {         // Insert the remains of LB into LC
        Type y = LB.Get (j);
        LC.Insert (k, y);
        j++;
        k++;
    }
    return LC;
}
```

```
1  Polynomial Polynomial :: Add (Polynomial B) {
2      Polynomial C;
3      int a = start; int b = B.start;  C.start = free;
4      float c;
5      while ( a <= finish && b <= B.finish ){
6          Switch ( compare ( termArray[a].exp,
7              termArray[b].exp) ) {//compare
8          case '=' : //exps are equal
9              c = termArray[a].coef +//coef
10                 termArray[b].coef;
11             if ( c ) NewTerm ( c, termArray[a].exp );
12                 a++;  b++;  break;
13         case '>' : // new item with item b in C
14             NewTerm ( termArray[b].coef, termArray[b].exp
                   );
15             b++;  break;
16         case '<': // new item with item a in C
17             NewTerm ( termArray[a].coef, termArray[a].exp
                   );
18             a++;
19     }
```

```
20      for (; a <= finish; a++ ) //A has remains
21              NewTerm ( termArray[a].coef, termArray[a].exp
                    );
22      for (; b <= B.finish; b++ ) //B has remains
23            NewTerm ( termArray[b].coef, termArray[b].exp
                  );
24      C.finish = free-1;
25      return C;
26  }
```

2.1.3 Polynomial

Add a new item in the polynomial

```
1  void Polynomial :: NewTerm ( float c, int e ) {
2  // Add a new item into polynomial
3      if ( free >= maxTerms ) {
4          cout << "Too many terms in polynomials" << endl;
5          return;
6      }
7      termArray[free].coef = c;
8      termArray[free].exp = e;
9      free++;
10 }
```

Points of Chapter Array

- Array
  - ADT of array
  - Methods
- Sequential List
  - ADT
  - Methods
  - Applications
- Polynomial
  - ADT and Representation
  - Addition

Next Section

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

Next Subsection

1 Arrays and ADT of Arrays
- Arrays
- Sequential List (Sequence)
- Polynomial

2 Implementation of Sequential List using STL
- Using the Standard string Class
- Array
- Using the Standard vector Class
- Using the STL deque Container
- Difference between vector and deque

3 Brief Summary

2.2.1 Using the Standard string Class

- C-style Strings
  - The standard string class provides support for character strings.
  - String class provides ease of use, convenience, and safety that C-style strings lack.

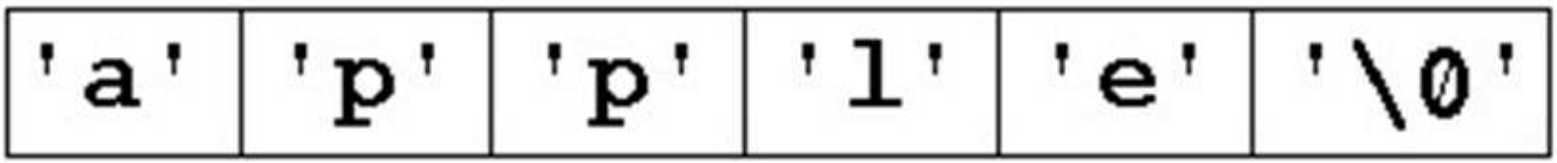


Figure 1 *A C-style string is an array of type* `char`

2.2.1 Using the Standard string Class

- String class

```
 1: int main(int argc, char* argv[]) {
 2:
 3:     string s1(argv[0]);  // convert from char*
 4:
 5:     char apple[] = "apple";
 6:     string s2(apple);    // convert from char[]
 7:
 8:     cout << s1 << endl;
 9:     cout << s2 << endl;
10:
11:     return EXIT_SUCCESS;
12: }
```

Listing 1 *Converting from a C-style string*

2.2.1 Using the Standard string Class

- Advanced String Operations

- *erase*
  Removes a sequence of characters from a `string`
- *find*
  Searches a `string` for the occurrence of another `string`
- *substr*
  Returns, as a `string`, part of another `string`
- *replace*
  Replaces a substring of characters with another string
- *insert*
  Inserts a `string` into another `string`

```
 1: string s1("Demonstrating all the advanced");
 2: string s2("string functions!!!");
 3:
 4: // erase the exclamation marks
 5: s2.erase(16, 3);
 6:
 7: // replace 'all the' with 'some'
 8: s1.replace(14, 7, "some");
 9:
10: // insert a space at the beginning of s2
11: s2.insert(0, " ");
12:
13: // extract everything after "some" from s1
14: int index = s1.find("some");
15: string s3 = s1.substr(index);
16:
17: cout << s1 << s2 << endl << s3 << endl;
```

Listing 2 *Some advanced string functions*

Next Subsection

1 Arrays and ADT of Arrays
- Arrays
- Sequential List (Sequence)
- Polynomial

2 Implementation of Sequential List using STL
- Using the Standard string Class
- Array
- Using the Standard vector Class
- Using the STL deque Container
- Difference between vector and deque

3 Brief Summary

2.2.2 Array

- C++ provides basic support for a sequence of homogeneous data objects through arrays.

```
1: // declare and create an array of integers
2: int cpp_array[10];
```

Listing 5 *An array in C++*

Example

```
 1: #include <iostream>
 2: #include <cstdlib>
 3:
 4: using namespace std;
 5:
 6: int main(int argc, char* argv[]) {
 7:
 8:     int arr[25];
 9:
10:     for (int i = 0; i < 25; i++) {
11:         arr[i] = i;
12:     }
13:
14:     cout << "The first element equals: " << arr[0] << endl;
15:     cout << "The second element equals: " << arr[1] << endl;
16:     cout << "The last element equals: " << arr[24] << endl;
17:
18:     return EXIT_SUCCESS;
19: }
20:
```

Listing 6 *Accessing elements of a C++ array*

Next Subsection

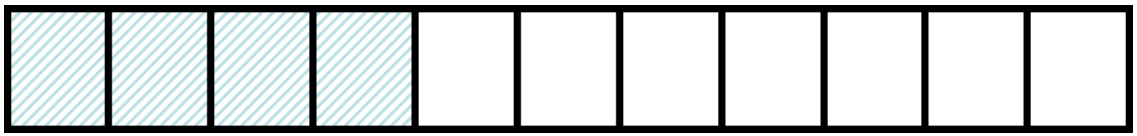
1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.2.3 Using the Standard vector Class

- Vector as an Array Class
  - More safer than array
  - Templates supported

```
 1: #include <string>
 2: #include <cstdlib>
 3: #include <iostream>
 4: #include <vector>
 5:
 6: using namespace std;
 7:
 8: int main(int argc, char* argv[]) {
 9:
10:     vector<int> v1;
11:     vector<double> v2;
12:     vector<bool> v3;
13:     vector<string> v4;
14:
15:     return EXIT_SUCCESS;
16: }
```

Listing 1 *Declaring vector objects*

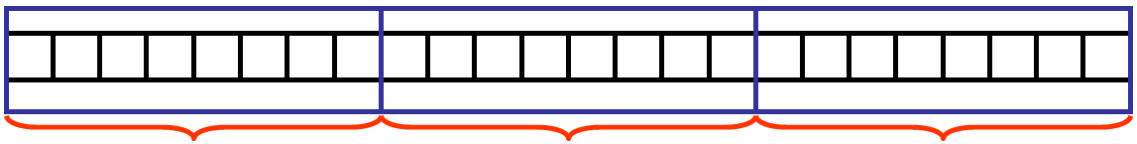


2.2.3 Using the Standard vector Class

- declare a vector of vector objects
  - an implementation for a two-dimensional data structure such as a matrix.

```
1: vector<vector<int> > matrix;
```

Listing 2 *A vector of vector objects*



2.2.3 Using the Standard vector Class

- Constructor

```
 1: #include <string>
 2: #include <cstdlib>
 3: #include <iostream>
 4: #include <vector>
 5:
 6: using namespace std;
 7:
 8: int main(int argc, char* argv[]) {
 9:
10:     vector<int> v1;          // initially empty
11:     vector<int> v2(5);       // 5 elements, initialized to 0
12:     vector<int> v3(10, 1);   // 10 elements, initialized to 1
13:     vector<int> v4(v3);      // v4 is a copy of v3
14:
15:     return EXIT_SUCCESS;
16: }
```

Listing 3 *vector constructors*

2.2.3 Using the Standard vector Class

- Element access

```
1: vector<int> v(10);
2:
3: v[1] = 2;
4: v.at(2) = 45;
5: v.front() = v.back();
```

Listing 4 *Element access*

- Other member functions

```
 1: // An initially empty vector
 2: vector<int> v;
 3:
 4: // push elements in
 5: for (int i = 0; i < 5; i++) {
 6:     v.push_back(i);
 7:     cout << "Size: " << v.size() << endl;
 8: }
 9:
10: // pop elements off
11: for (int j = 0; j < 5; j++) {
12:     v.pop_back();
13:     cout << "Size: " << v.size() << endl;
14: }
15:
16: cout << endl << v.empty() << endl;
```

Listing 5 *Other member functions*

2.2.3 Using the Standard vector Class

- Vector as an STL Container
  - As an STL container, class vector provides access to its elements via iterators. Iterators allow interoperability of vector objects and the STL algorithms.

```
 1: vector<int> v;
 2:
 3: for (int i = 0; i < 10; i++) {
 4:     v.push_back(i);
 5: }
 6:
 7: // Iterate though elements
 8: vector<int>::iterator it;
 9: for (it = v.begin(); it != v.end(); it++) {
10:     cout << *it << endl;
11: }
```

Listing 6 *Class* `vector` *iterators*

2.2.3 Using the Standard vector Class

- Iterators
  - return iterators to the invoking vector.
  - begin
    - **returns an iterator to the first element in the vector**
  - end
    - **returns an iterator positioned beyond the last element in the vector**

2.2.3 Using the Standard vector Class

- Insert and erase functions
  - Take vector iterators as parameters.
  - These iterators indicate the location to insert, or the elements to erase.

```
 1: // A vector with 10 elements
 2: vector<int> v(10);
 3:
 4: // Insert an element at the beginning
 5: v.insert(v.begin(), 50);
 6: cout << v[0] << endl;
 7: cout << v.size() << endl;
 8:
 9: // Erase the last five elements
10: vector<int>::iterator it = v.end() - 5;
11: v.erase(it, v.end());
12: cout << v.size() << endl;
```

Listing 9 *Functions* `insert` *and* `erase`

Next Subsection

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.2.4 Using the STL deque Container

- Double-ended Queue
- Interface
  - can store and provide access to a linear sequence of elements

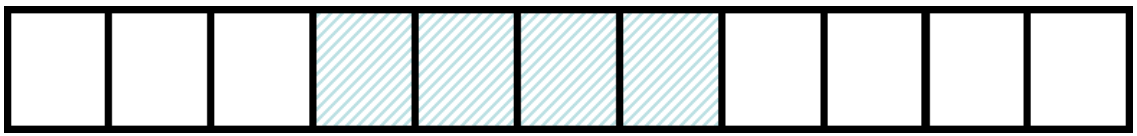
```
 1: #include <cstdlib>
 2: #include <iostream>
 3: #include <vector>
 4: #include <deque>
 5:
 6: using namespace std;
 7:
 8: int main(int argc, char* argv[]) {
 9:
10:     vector<int> v(10, 1);
11:     deque<int> d(10, 1);
12:
13:     v[9] = 2;
14:     d[9] = 2;
15:
16:     cout << v.front() << " " << v.back() << endl;
17:     cout << d.front() << " " << d.back() << endl;
18:
19:     v.push_back(3);
20:     d.push_back(3);
21:
22:     v.pop_back();
23:     d.pop_back();
24:
25:     cout << v.size() << endl;
26:     cout << d.size() << endl;
27:
28:     ostream_iterator<int> out(cout, " ");
29:     copy(v.begin(), v.end(), out);
30:     copy(d.begin(), d.end(), out);
31:
32:     return EXIT_SUCCESS;
33: }
34:
```

STL deque

- push_front and pop_front

```
1: deque<int> d(10);  // 10 elements, initialized to 0
2:
3: d.push_front(2);
4: cout << d.front() << endl; // Outputs "2"
5:
6: d.pop_front();
7: cout << d.front() << endl; // Outputs "0"
```

Listing 2 *push_front and pop_front*



2.2.4 Using the STL deque Container

- count and count_if functions:
  - Counting the number of items that possess certain properties

```
 1: // a predicate
 2: bool is_odd(int i) {
 3:     return ((i % 2) == 1);
 4: }
 5:
 6: int main(int argc, char* argv[]) {
 7:
 8:     deque<int> numbers;
 9:     for (int i = 0; i < 20; i++) {
10:         numbers.push_back(i);
11:     }
12:
13:     cout << count(numbers.begin(), numbers.end(), 10) << endl;
14:     cout << count_if(numbers.begin(), numbers.end(), is_odd)
15:     << endl;
16:
17:     return EXIT_SUCCESS;
18: }
```

Listing 3 *The count and count_if functions*

Next Subsection

1 Arrays and ADT of Arrays
- Arrays
- Sequential List (Sequence)
- Polynomial

2 Implementation of Sequential List using STL
- Using the Standard string Class
- Array
- Using the Standard vector Class
- Using the STL deque Container
- Difference between vector and deque

3 Brief Summary

2.2.5 Difference between vector and deque

- Storage strategies
  - Vectors reserve memory only at the rear of stored elements

Figure 1 *Element storage in a* `vector`

  - Deques reserve memory locations at both the front and rear of their stored elements

Figure 2 *Element storage in a* `deque`

2.2.5 Difference between vector and deque

- Operations
  - The same ones
    - **push_back**
    - **pop_back**
  - For deque only
    - **push_front**
    - **pop_front**
  - For vector
    - **Alternative method: insert or erase the first element**

Next Section

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary

2.3 Brief Summary

1. Arrays and ADT of Arrays
   - Arrays
   - Sequential List (Sequence)
   - Polynomial
2. Implementation of Sequential List using STL
   - Using the Standard string Class
   - Array
   - Using the Standard vector Class
   - Using the STL deque Container
   - Difference between vector and deque
3. Brief Summary