APA 254

Data Structures

Lecture 3.1 (Array Representation)

Dept. of Information Systems Hanyang University

# Introduction

## Data Object

* + a set of instances or values
  + Examples:
    - Boolean = {false, true}
    - Digit = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
    - Letter = {A, B, C, …, Z, a, b, c, …, z}
    - String = {a, b, …, aa, ab, ac,…}
  + An individual instance is either primitive or composite.
  + An element is the individual component of a composite instance.

# Introduction

## Data Structure

* + data object together with the relationships among instances and elements that comprise an instance
  + Among instances of integer 369 < 370

280+4 = 284

* + Among elements that comprise an instance 369

3 is more significant than 6

3 is immediately to the left of 6 9 is immediately to the right of 6

# Introduction

## Abstract Data Type (ADT)

* + ADT is a collection of data and a set of operations that can be performed on the data.
  + It enables us to think abstractly about the data …
  + We can **separate concepts from implementation**.
  + Typically, we choose a data structure and algorithms that provide an implementation of an ADT.

# Linear List

## Definitions

* + Linear list is a data object whose instances are of the form (e1,e2,…,en)
  + ei is an element of the list.
  + e1 is the first element, and en is the last element.
  + n is the length of the list.
  + When n = 0, it is called an empty list.
  + e1 comes before e2, e2 comes before e3, and so on.

## Examples

* + student names order by their alphabets
  + a list of exam scores sorted by descending order

# ADT for Linear List

**AbstractDataType** LinearList {

**instances**

ordered finite collections of zero or more elements

**operations**

Create(): create an empty linear list Destroy(): erase the list

IsEmpty(): return true if empty, false otherwise Length(): return the list size

Find(k,x): return the kth element of the list in x Search(x): return the position of x in the list Delete(k,x): delete the kth element and return it in x Insert(k,x): insert x just after the kth element Output(out): put the list into the output stream *out*

}

7

# Implementations of Linear List

* Array-based (Formula-based)
  + Uses a mathematical formula to determine where (i.e., the memory address) to store each element of a list
* Linked list (Pointer-based)
  + The elements of a list may be stored in any arbitrary set of locations
  + Each element has an explicit pointer (or link) to the next element
* Indirect addressing
  + The elements of a list may be stored in any arbitrary set of locations
  + Maintain a table such that the *i*th table entry tells us where the *i* th element is stored
* Simulated pointer
  + Similar to linked representation but integers replace the C++ pointers

**Array-based Representation of Linear List**

## It uses an array to store the elements of linear list.

* Individual element is located in the array using a mathematical formula.

## typical formula

*location(i) = i – 1*

 *i*th element of the list is in position *i-1* of the array

element [0] [1] [2] [3] [4] MaxSize-1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 5 | 2 | 4 | 8 | 1 |  |  |  |

length=5

* See Figure 5.1 for different ways to store

# Array-based Class ‘LinearList’

template <class T> class LinearList {

public: LinearList(int MaxListSize = 10);

~LinearList() { delete [] element; }

bool isEmpty() const { return length == 0; } int Length() const { return length; }

bool Find(int k, T& x) const; int Search(const T& x) const;

LinearList<T>& Delete(int k, T& x);

LinearList<T>& Insert(int k, const T& x); void Output(ostream& out) const;

private: int length; int MaxSize; T \*element;

};

# Constructor ‘LinearList’

template<class T> LinearList<T>::LinearList(int MaxListSize)

{ // Constructor for array-based linear list

MaxSize = MaxListSize; element = new T[MaxSize]; length = 0;

}

## The time complexity is

**(1)**

# Operation ‘Find’

template<class T>

bool LinearList<T>::Find(int k, T& x) const

{ // Set x to the k’th element in the list if it exists if (k < 1 || k > length)

return false;

x = element[k-1]; return true;

}

## The time complexity is

**(1)**

# Operation ‘Search’

template<class T>

int LinearList<T>::Search(const T& x) const

{ // Locate x and return the position of x if found for (int i = 0; i < length; i++)

if (element[i] == x)

return ++i;

return 0;

}

* The time complexity is

**O(length)**

# Operation ‘Delete’

template<class T>

LinearList<T>& LinearList<T>::Delete(int k, T& x)

{ // Delete the k’th element if it exists. if (Find(k, x)) {

for (int i = k, i < length; i++) element[i-1] = element[i];

length--;

return \*this;

}

else throw OutOfBounds();

}

## The time complexity is

**O((length-k) *s*)** where *s* is the size of each element

# Operation ‘Insert’

template<class T>

LinearList<T>& LinearList<T>::Insert(int k, const T& x)

{ // Insert x after the k’th element.

if (k < 0 || k > length) throw OutOfBounds(); if (length == MaxSize) throw NoMem();

for (int i = length-1; i >= k; i--) element[i+1] = element[i];

element[k] = x; length++; return \*this;

}

## The time complexity is

**O((length-k) *s*)** where *s* is the size of each element

# Operation ‘Output’

template<class T>

void LinearList<T>::Output(ostream& out) const

{ // print out the list

for (int i = 0; i < length; i++) out << element[i] << “ ”;

}

* + The time complexity is

**(length)**

# Changing the Length of 1D Array

## What does it mean to change the length of an array?

* + In what situations do you need to do this?

## How can you change the length of one dimensional array?

* + Read 5.3.2 and see Program 5.2

## Read Chapter 5

18