Data Structures

(Feodor F. Dragan)
Department of Mathematics and Computer Science
Kent State University

Fall, 2000

Textbook: Mark Allen Weiss,
Data Structures and Problem Solving Using C++.
Second Edition, Addison Wesley, 2000.

Grading:

- Homework: 20%
- First Exam: 30%
- Final Exam: 30%
- Programming Project I: 10%
- Programming Project II: 10%

All the course slides will be available at
http://www.mcs.kent.edu/~dragan/CS33001-DS.html

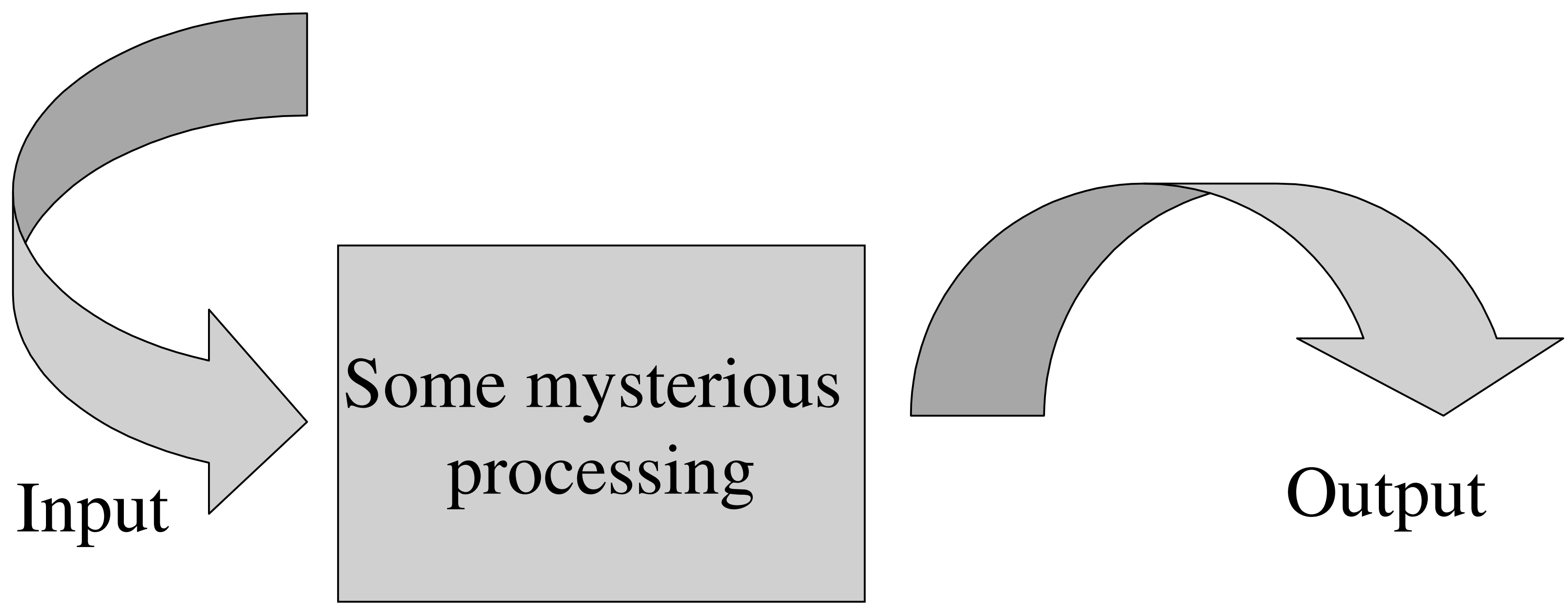
Course Outline

- Chapter 1: Arrays, Pointers, and Structures
- Chapter 2: Objects and Classes
- Chapter 3: Templates
- Chapter 4: Inheritance
- Chapter 6: Algorithm Analysis
- Chapter 7: The Standard Template Library
- Chapter 8: Recursion
- Chapter 9: Sorting
- Chapter 10: Randomization
- Chapter 12: Stacks and Compilers
- Chapter 13: Utilities
- Chapter 14: Simulation
- Chapter 15: Graphs and Paths
- Chapter 16: Stacks and Queues
- Chapter 17: Linked Lists
- Chapter 18: Trees
- Chapter 19: Binary Search Trees
- Chapter 20: Hashing
- Chapter 21: Heaps as Priority Queues
- Chapter 22: Splay Trees

Before we go into details,
what is a data structure exactly?

To answer that, we must first understand:

What is a computer program?

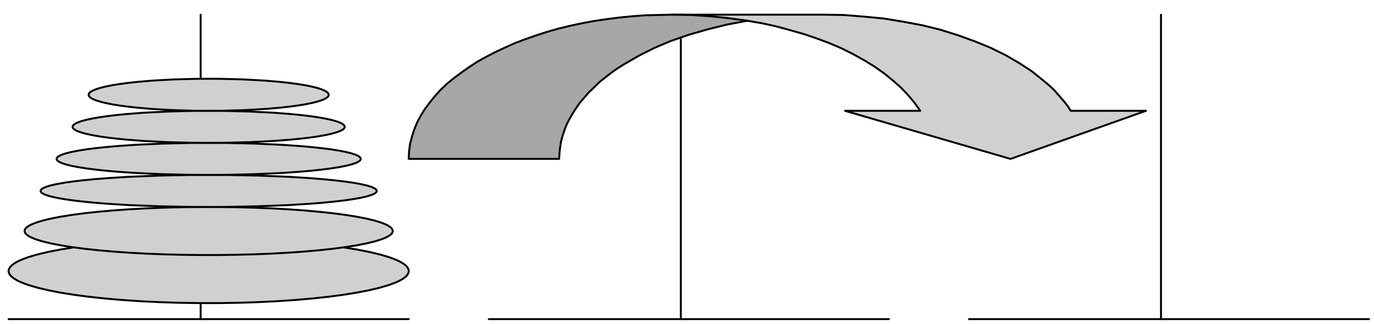


How to solve the following problems:

1. Input 3 numbers, print out the maximum.

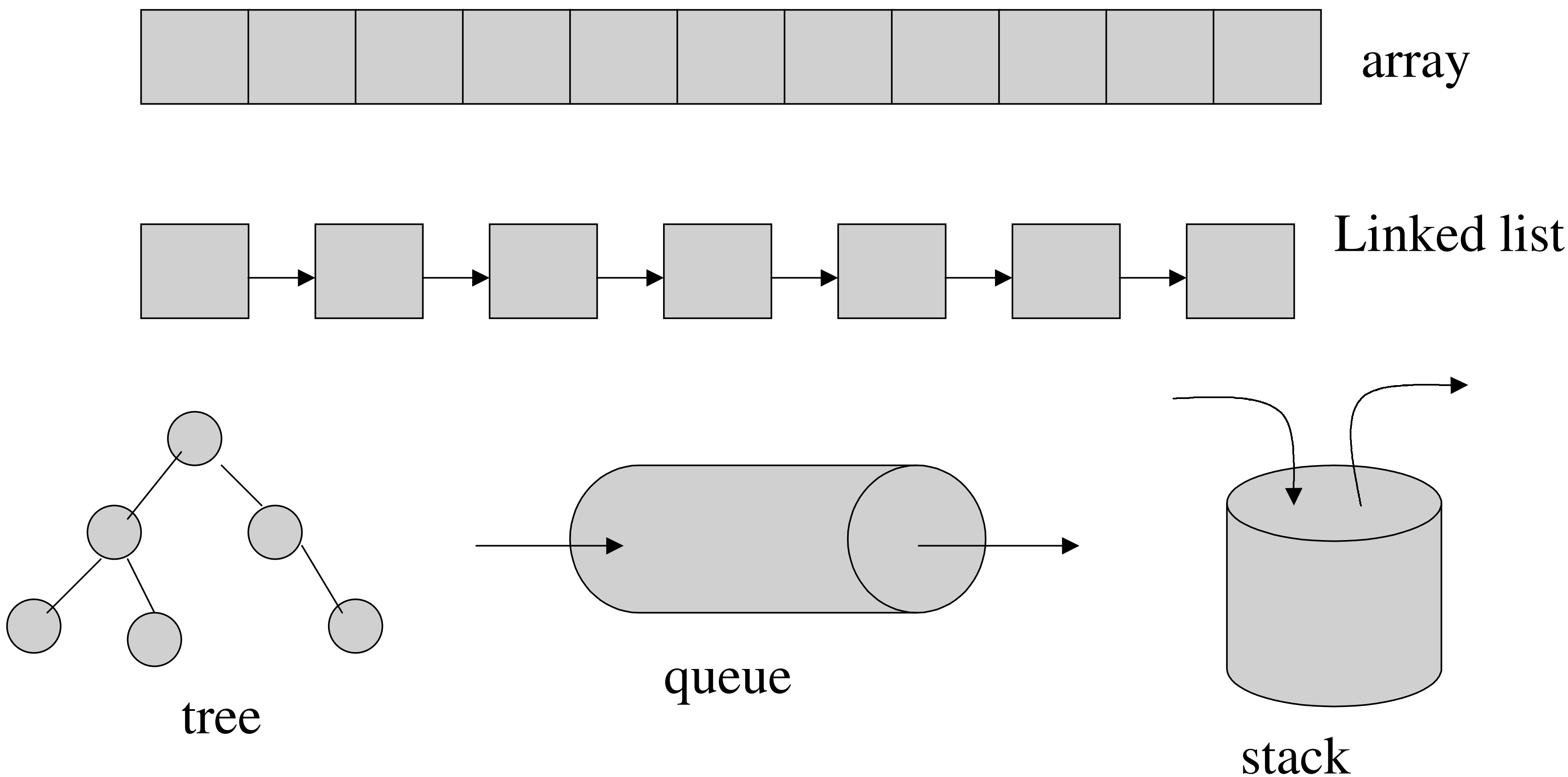
2. Input 30000 numbers, print out the largest 10 numbers.

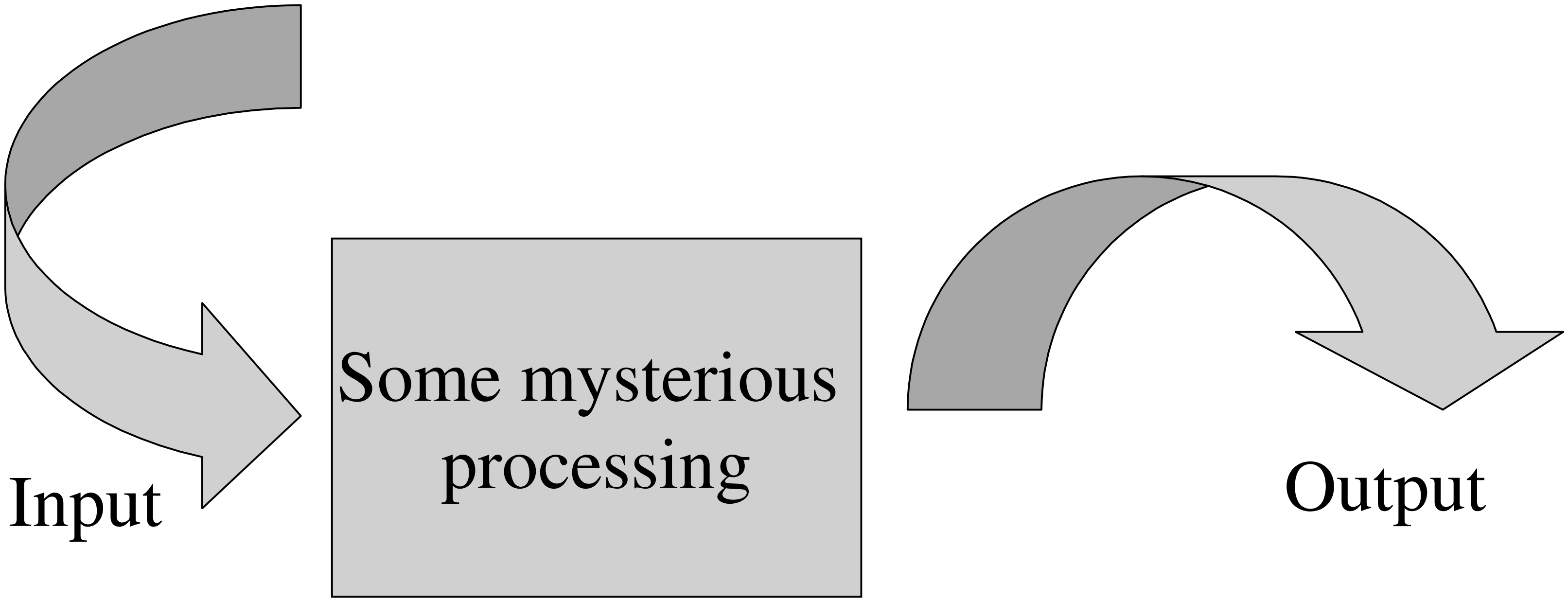
3. The Tower of Hanoi Problem.



Condition: A larger one can not be on the top of a smaller one.

Data structures let the input and output be represented in a way that can be handled efficiently and effectively.





Data structures+Algorithms=Programs

CHAPTER 1
ARRAYS, POINTERS, AND STRUCTURES

Contents

- Array and Strings
- Pointers
- Structures
- Homework

- A **pointer** is an object that can be used to access another object.

- An **array** is a collection of identically typed objects.

- A **structure** is a collection of objects that need not be of the same type.

Array

array → primitive array (2nd class object)

array → vector, string (1st class object)

The vector and string classes are part of the Standard Template Library (STL).

- vector knows how large it is (a.size ( ))
- strings can be compared (==, <)
- both can be copied with = .

Primitive Array

- declaration: const int DAYS_IN_MONTH [ ] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
- limitation: fixed size, cannot be changed after declaration.

Memory Model

- Assume 4 byte int
- Declaration:

  int A[3] ; int i ;

&A[0]	1000	A[0]
&A[1]	1004	A[1]
&A[2]	1008	A[2]
	1012	
&i	1016	i
		...
&A	5620	A=1000

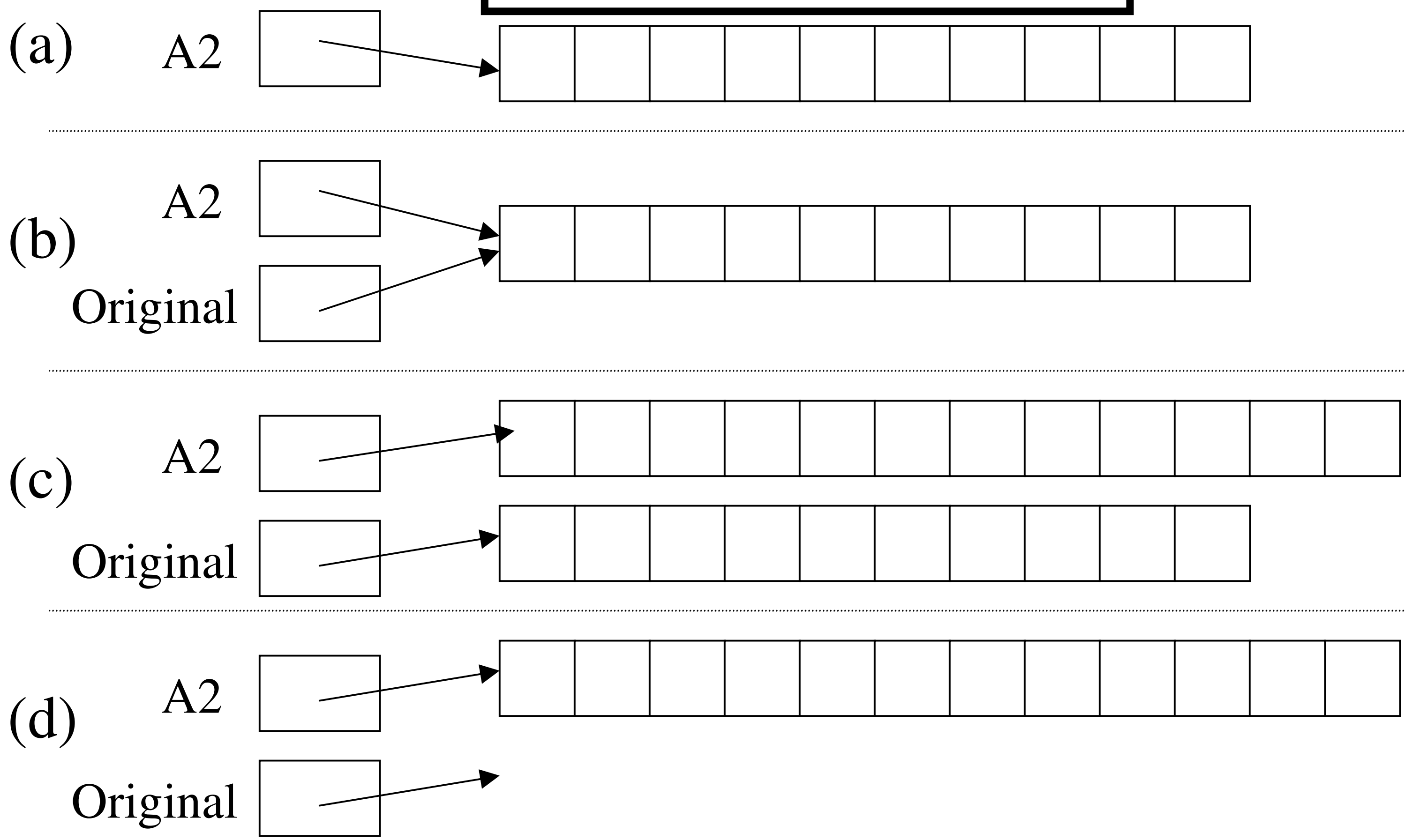
Vector

```
# include <vector>

vector<int> a(3) ; // 3 int objects: a[0], a[1], and a[2]

                  a.size( )

                  a.resize(3) ;


vector<int> a ; // 0 int objects

a.resize (3) ; // 3 int objects: a[0], a[1], and a[2]
```

```
#include <iostream>
#include <vector>
using namespace std ;

// Read an unlimited number of ints; fill the vector parameter
// with the data; its size after the return tells how many items
// were read.
void getInts ( vector<int> & array )
{
        int itemsRead = 0;
        int inputVal;
        cout << "Enter any number of integers: ";
        while ( cin >> inputVal )
        {
                if( itemsRead == array.size( ) )
                    array.resize( array.size( )  *2 + 1 );
                array[ itemsRead++ ] = inputVal;
        }
        array.resize( itemsRead );
}

int main ( )
{
        vector<int> array;
        getInts( array );
        for( int i = 0; i < array.size( ); i++ )
                cout << array[ i ] << endl;
        return 0;
}
```

Code to read an unlimited number of ints and write them out, using array-doubling.

Array Expansion



(a) starting point: **A2** points at 10 integers;

(b) after step 1: **Original** points at the 10 integers;

(c) after steps 2 and 3: **A2** points at 12 integers, the first 10 of which are copied from **Original**

(d) after step 4: the 10 integers are freed

```
#include <stdlib.h>
#include <iostream>
#include <vector>
using namespace std ;

// Read an unlimited number of ints; fill the vector parameter
// with the data; its size after the return tells how many items
// were read.
void getInts ( vector<int> & array )
{
        int inputVal;
        array.resize ( 0 );
        cout << “Enter any number of integers: “;
        while ( cin >> inputVal )
            array.push_back ( inputVal );
}
int main ( )
{
        vector<int> array;
        getInts( array );
        for( int i = 0; i < array.size( ); i++ )
            cout << array[ i ] << endl;
        return 0;
}
```

Code to read an unlimited number of ints and write them out, using push_back.

Parameter passing Mechanisms.

- **Call by value**
  - int findMax (vector<int> **a**) ;
  - creates a copy of **a**
- **Call by reference**
  - int findMax (vector<int> &**a**) ;
  - creates no copy of **a**; refers to the same **a**; argument might be changed
- **Call by constant reference**
  - int findMax (const vector<int> &**a**) ;
  - creates no copy of **a**; refers to the same **a**; **a** is unchanged

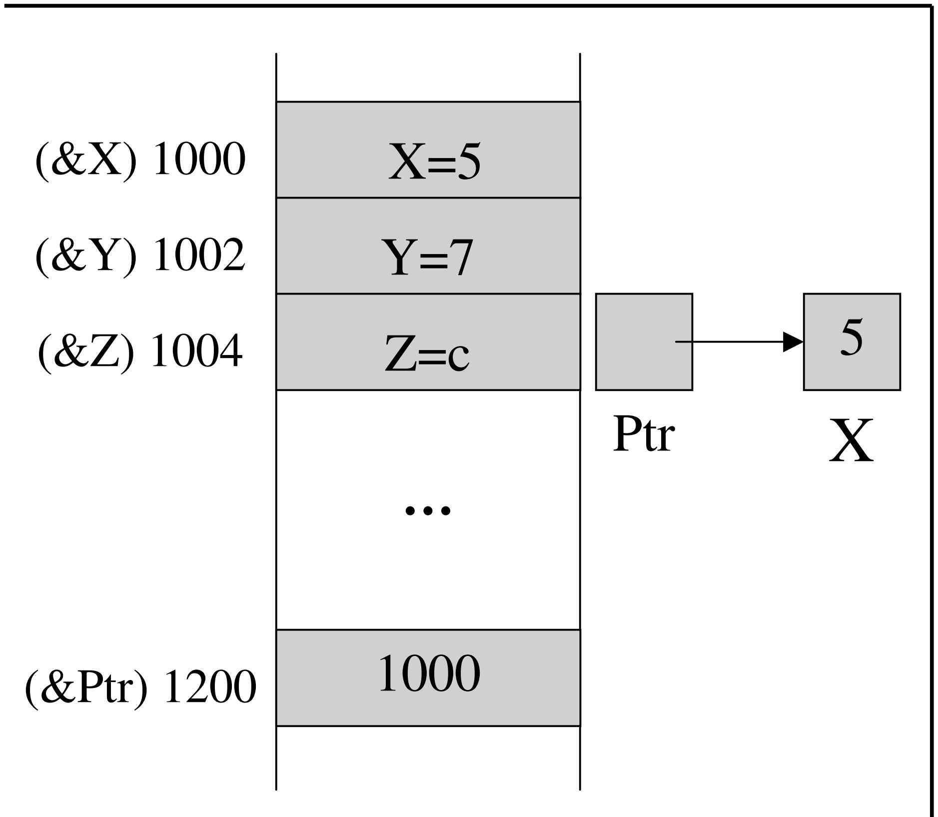
Strings

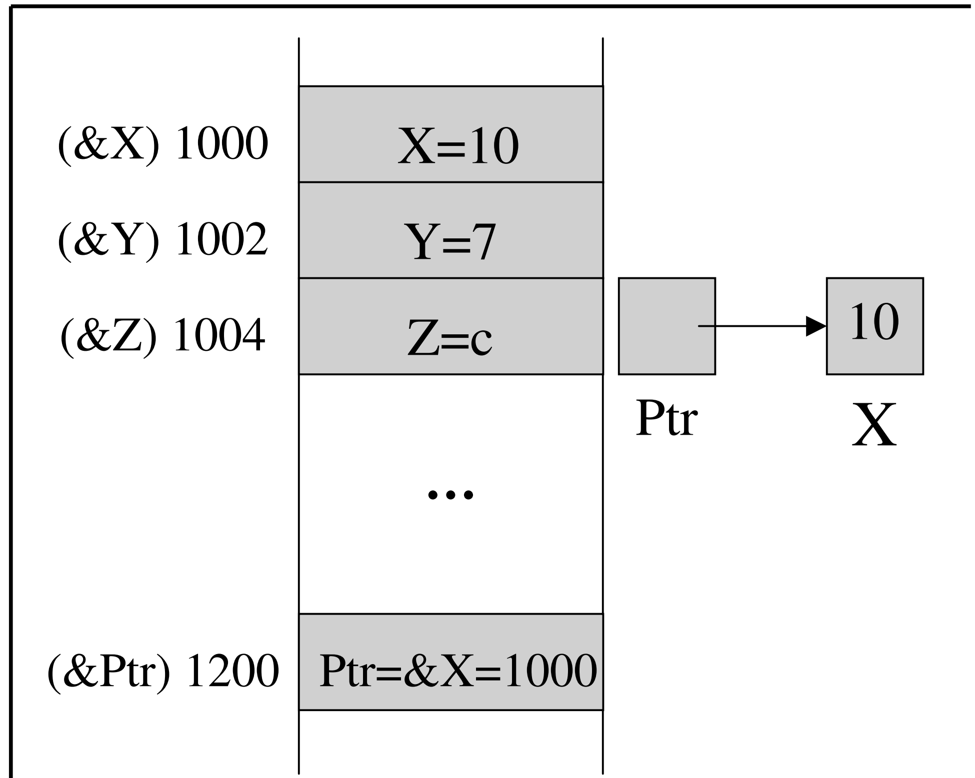
```
# include <string>

string a = “hello”;
string b = “all”;
string c;
c = a + “ “ ;
c += b;
cout << c[ 3 ];
```

a.length( ) is 4

a	h	e	l	l	o				
b	a	l	l						
c	h	e	l	l	o		a	l	l

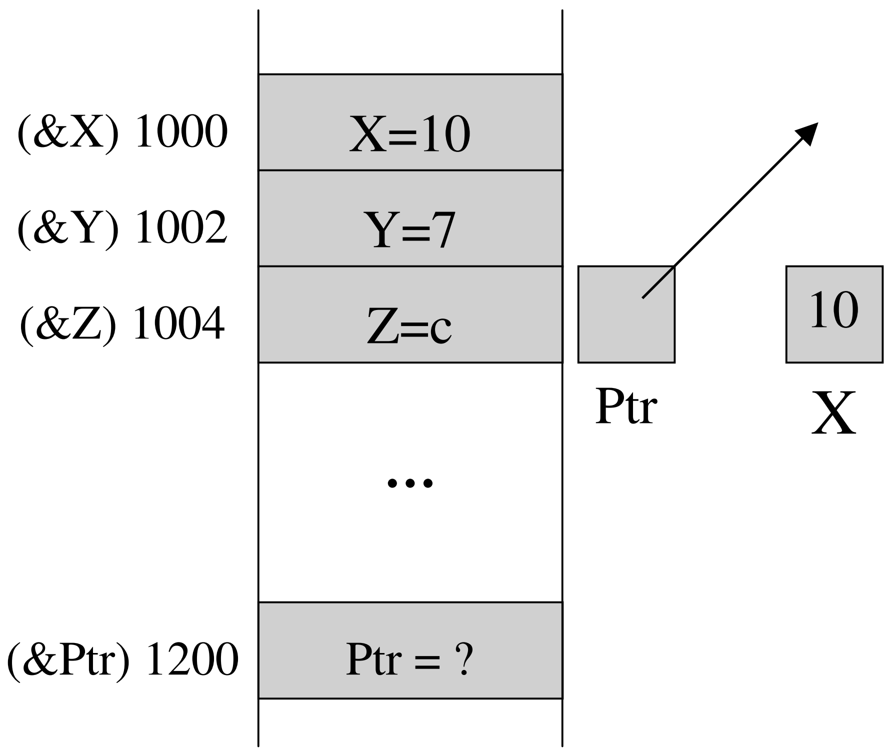
Pointers





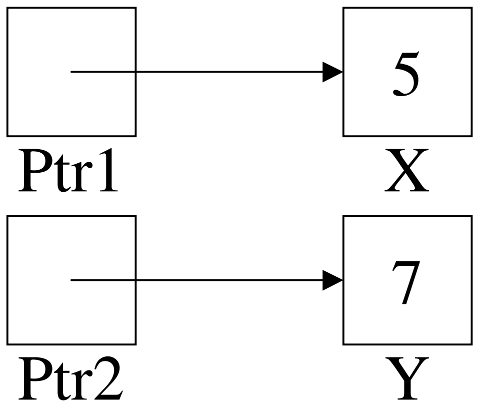
- What is the result of *Ptr = 10?

Uninitialized Pointer

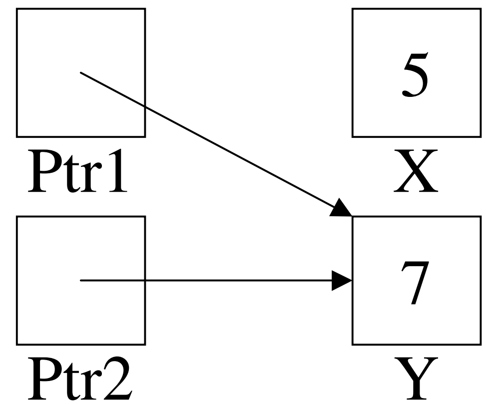


A Few Operations

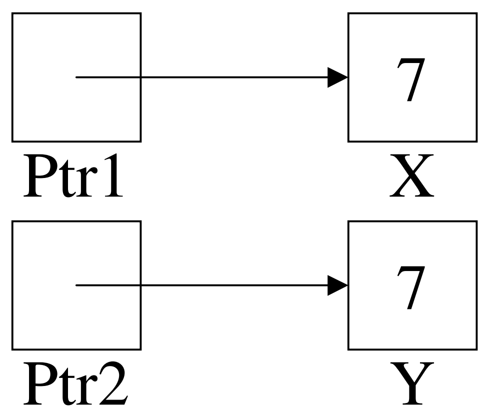
- Initial State



- Ptr1 = Ptr2



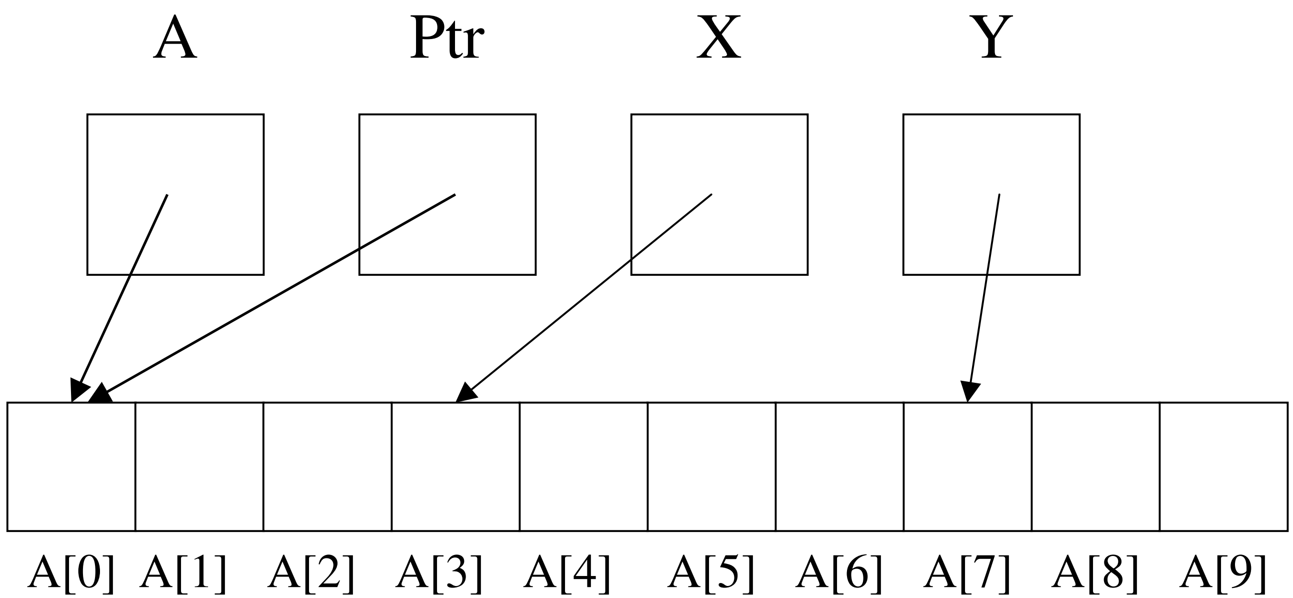
- *Ptr1 = *Ptr2



Pointer Arithmetic

X=&A [3] ;

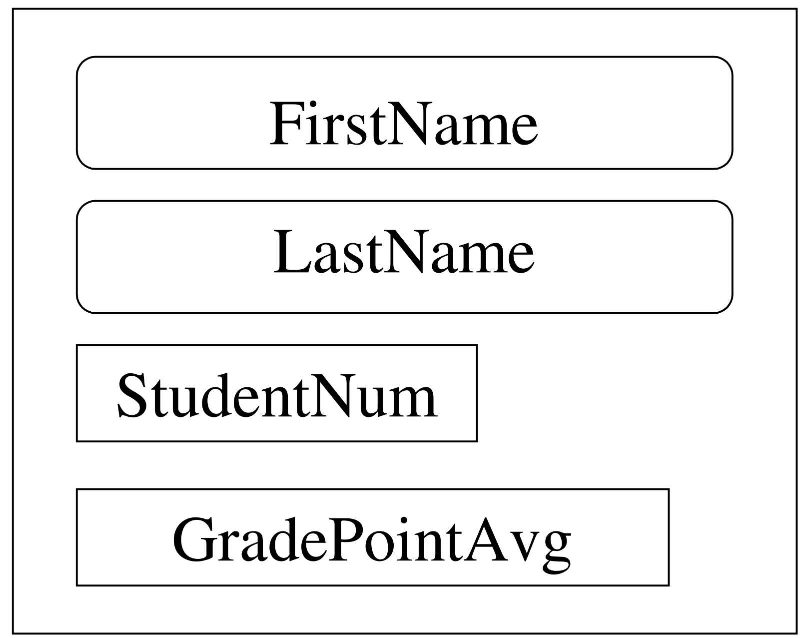
Y=X+4 ;



Structures

```
Structure Student
{
        char FirstName ;
        char LastName ;
        int StudentNum ;
        double GradePointAvg ;
};
```

- Student Structure



```
// Print the student information.
void printInfo ( const Student &s )
{
  cout <<  “ID is  “  << s.studentNum  << endl;
  cout <<  “Name is  “  << s.firstName  <<  “  “
                                  << s.lastName  << endl ;
  cout <<  “GPA is “   << s.gradePointAvg << endl;
}

// Simple main.
int main ( )
{
        Student mary;

        mary.lastName = “Smith” ;
        mary.firstName = “Mary” ;
        mary.gradePointAvg = 4.0 ;
        mary.studentNum = 123456789 ;

        printInfo ( mary ) ;

        return 0;
 }
```

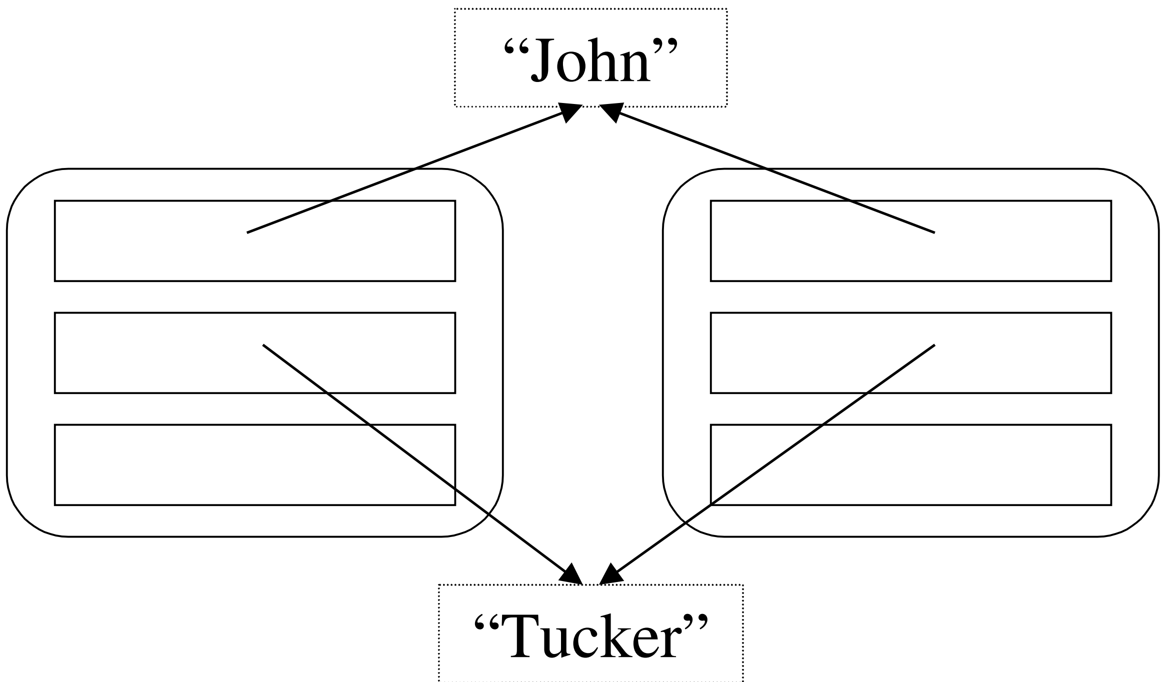
Program to illustrate the declaration of a structure, access of its data members, and parameter passing.

Data Copy

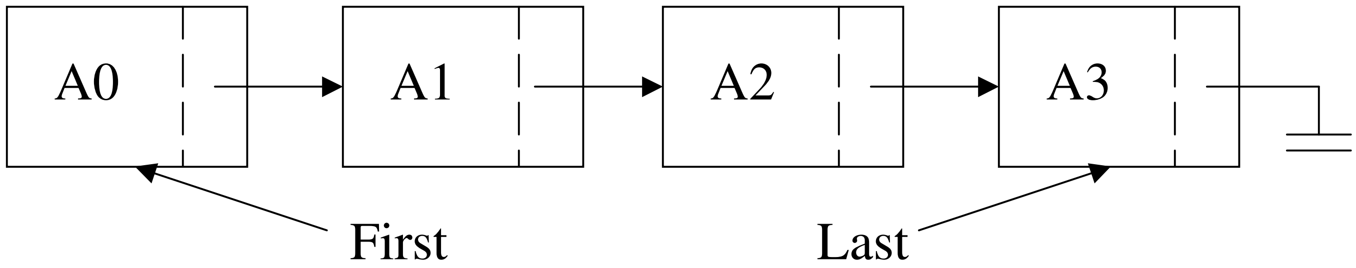
- **Indigenous data**: data contained by a structure
- **Exogenous data**: data not part of a structure, rather accessed by a pointer
- **Shallow copy**: is a copy of pointers rather than data being pointed at
- **Deep copy**: is a copy of the data being pointed at rather than the pointers

Data Copy (cont.)

- Example of shallow copy ➔ only pointers are copied



Linked List



The structure definition is

```
struct Node
{
    Object item;        //Some element
    Node *next;
};
```

At any point we can print the list by using the iteration

```
for( Node *p = first; p != NULL; p = p->next )
    printItem( p->item );
```

and at any point we can add a new last item x**, as in**

```
last->next = new Node;          // Attach a new Node
last = last->next;              // Adjust last
last->item = x;                 // Place x in the node
last->next = NULL;              // It's the last, so make next NULL
```

Homework

1. Carefully read chapter 1 from the textbook.
2. Problems:

   a) 1.4, page 36 of the textbook;

   b) 1.6, page 37 of the textbook.