



**2007-02-08**  
**Lecture #4**  
**(Material on Test 1)**

**Today's Outline**  
**C++ OOP**  
**Linked Lists**

## ADT – Abstract Data Types

**Primitive Data Types:** int, char, double, float, bool

**Abstract Data Types:** string, Date, Student, Car, DVD, etc.

An ADT is built using Primitive Data Types or other ADTs.

### Example:

**What is a String?**

- A collection of characters with an integer length

**What is a Date ?**

- A collection of integers (day, month, year)

**What is a student?**

- 3 Strings for F,L,M Name - 1 Date for Birthday
- 1 Integer for Student ID - 1 Integer for Year in school (1,2,3,4...)

## ADT parts

**1.)A Set of Data Values**

**2.)A Set of Operations (functions/methods) on those values**

**Example:**

**Student**

**Data**

- **F,L,M Name, Birthday, School Year, Student ID**
- **Current Semester, List of Classes for current semester**

**Operations**

- **RegisterForClass() // Adds to List of Classes**
- **PassCourse() // Removes from List of Classes**
- **FinishSemester() // Increase Semester / School Year**
- **PrintStudent() // Display Name, Bday, etc.**
- **PrintClassSchedule() // Display List of Classes**

## Benefits of ADTs

- 1.) **Modularity** – *Code is broken into small ... manageable parts*
- 2.) **Information Hiding** – *User just knows what it does, not how*
- 3.) **Implementation Independence** – *We can write a main program that uses the ADT ... modify an algorithm in the ADT ... and not have to rewrite the main algorithm*

*Example:*

```
main(){  
    Student john ;  
    john.RegisterForCourse(“cs251”) ;  
}
```

// what if how we register for courses changed? For example  
// we used to use pencil & paper, now we use Titan Web

- 4.) **Encapsulation** – *We combine related items into one ADT ... or related actions into one method/function*

## OOP

Object oriented programming uses these ADTs

Steps for us to use OOP

1.) Write a **Specification**

- Use Pseudo Code to outline the ADT
- Lays out “**what to do**” ... NOT “how to do it”

2.) **Implementation** - Code the Classes in C++

Example:

**Car Data**

- Make
- Model
- Year
- Color
- Mileage

**Car Operations**

- Paint( New Color )
- Drive ( Miles)

## Let's work with an already existing ADT

### The C++ string class (`#include<string>`)

#### Specification (what do do)

**string data values (i'm just guessing here)**

- the array of characters
- the amount of space (characters) allocated

**string operations (these are real)**

- append
- capacity
- clear
- length
- replace
- push\_back

## The C++ code ...

```
class MyString{
    private:
        char * TheString ;    // data
        int AllocatedSpace ; // data
    public:
        MyString() ;          // constructor
        MyString(char*) ;    // constructor
        ~MyString() ;        // destructor
        void append(char*) ; // operation
        int capacity( ) ;     // operation
        void clear ( ) ;      // operation
        int length( ) ;       // operation
        void replace(char, char) ; // operation
        void push_back(char) ; // operation
} ; // don't forget the semicolon
```

## **C++ Classes (constructors, destructors, pointers)**

```
int main( ... )  
{  
    MyString stack1 ; // on the stack, calls constructor  
    MyString stack2(“hello”) ; // on the stack, calls constructor  
    MyString stack3(stack2) ; // on the stack, calls copy constructor  
  
    return EXIT_SUCCESS ;  
} // calls the 3 destructors
```

## C++ Classes (constructors, destructors, pointers)

```
int main( ... )  
{  
    MyString * ptr1 ; // ptr on the stack, but no MyString yet  
    MyString * ptr2 ; // ptr on the stack, but no MyString yet  
    MyString * ptr3 ; // ptr on the stack, but no MyString yet  
    ptr1 = new MyString() ; // on heap, calls constructor  
    ptr2 = new MyString("hello") ; // on heap, constructor  
    ptr3 = new MyString(ptr2) ; // on heap, copy constructor  
  
    delete ptr1 ; // destructor  
    delete ptr2 ; // destructor  
    delete ptr3 ; // destructor  
  
    return EXIT_SUCCESS ;  
}
```

## C++ Classes (constructors, destructors, pointers)

```
int main( ... )  
{  
    MyString * ptr1 ; // ptr on the stack, but no MyString yet  
    MyString * ptr2 ; // ptr on the stack, but no MyString yet  
    ptr1 = new MyString(“hello”) ; // on heap, constructor  
    ptr2 = ptr1 ; // 2 pointers, 1 data  
  
    // NOTICE YOU ONLY NEED 1 DELETE  
    delete ptr1 ; // destructor, could've also said delete ptr2  
        // but DO NOT put them both!  
  
    return EXIT_SUCCESS ;  
}
```

## C++ Classes (pointers)

```
MyString stack1("hello");  
MyString ptr1 = new MyString("goodbye");
```

```
stack1.append(" world");  
ptr1->append(" mister");
```

```
stack1.push_back("!");  
ptr1->push_back("!");
```

**NOTICE: if it's a pointer ... use the -> to call members functions  
(minus sign, greater than sign)**

**if it's a normal object on the stack ... just use a .  
(period)**

## Let's discuss the C++ syntax

```
class NAME_OF_OBJECT {  
    private:  
        // information hidden to outside world  
    public:  
        // information available to outside world  
}; // don't forget the semicolon
```

**Private** – In this string class above ... we don't want the user just reaching in and messing with the character string or the amount of space allocated ... so that information is private

**Public** – We will allow the user to work with the string, but in a controlled environment ... we write the append function, we write the push back function, etc.

## The Implementation of the operations ...

```
MyString::MyString(){ // constructor  
    AllocatedSpace = 100 ; // default is 100 characters  
    TheString = new char[AllocatedSpace] ;  
} // how to use: MyString * name = new MyString() ;
```

```
MyString::MyString(char * NewString){ // constructor  
    AllocatedSpace = strlen(NewString) * 2 ; // double space  
    TheString = new char[AllocatedSpace] ;  
    strncpy(TheString, NewString, strlen(NewString)) ;  
} // how to use: MyString * name = new MyString(“Justin Miller”) ;
```

```
MyString::~~MyString(){ // destructor  
    AllocatedSpace = 0 ;  
    delete [] TheString ;  
} // how to use: delete name ;
```

## **Definition of the operations**

**I have code written for the MyString class that we can look at now**

## Stack ADT

Let's write a C++ Stack Class with this specification ...

### Stack Data

- Integer Array
- Max Size of Stack

### Stack Operations

- Push
- Pop
- IsEmpty
- IsFull
- Clear

While we're writing it ... ask, why is the data private?

Why wouldn't we want somebody touching the Max Size variable?

## Standard Operations for most Objects

Typically our object stores data

But we make that data PRIVATE ... why?

Because we want to “**control**” how they access the data ...

Example:

```
class eBayItem{
    private:
        double CostOfShipping ;
        double InsuranceCost ;
    public:
        double GetCostOfShipping( ) ;
        void SetCostOfShipping(double, double) ;
};
// maybe we want the cost of shipping to include cost & insurance
// maybe we don't allow them to change the shipping after auction start
```

## **Pointer Implementation of a Linked List**

### **Advantages**

- 1.) No Max size, and it doesn't need to be resized (like MyString)**
- 2.) Avoid Linear cost of insertion / delete**  
Imagine inserting into a sorted array  
We'd have to copy all the items up / down a spot

### **Disadvantages**

- 1.) Extra space needed for pointers (not just storing the data)**
- 2.) Extra time needed for pointer access**

## Pointer Implementation of a Linked Lists

### Details

- A **Linked List** is a series of structures which are **NOT** necessarily in consecutive memory locations (where as an array is)
- Each structure has a pointer to the **NEXT** structure
- The pointer in the last cell points to **NULL**
- **Empty Linked List** is defined by the **Head Ptr = NULL**

```
struct LinkedListNode{  
    double data ;  
    LinkedListNode * nextPtr ;  
};
```

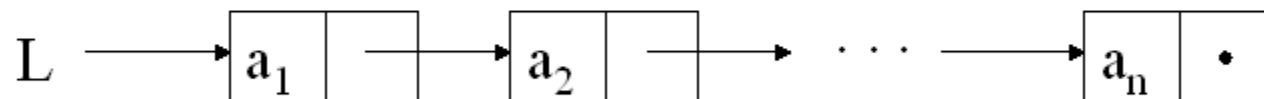
**NOTE:** struct is same as C++ class with all things public

## Pointer Implementation of a Linked List

# The Linked-List Data Structure

---

- List,  $L = (a_1, a_2, \dots, a_n)$
- A linked list representing  $L$



## **Pointer Implementation of Linked List**

### **Specification**

#### **Linked List Data**

- **Head Pointer to first Linked List Node**

#### **Linked List Operations**

- **Insert(double)**
- **Remove(double)**
- **Find(double)**
- **IsEmpty()**
- **Size()**
- **PrintList()**

**Note: Assume we want this list sorted ASC**

**Note: There is no IsFull**

**Note: How do we figure out the Size?**



## **Homework 2 (Involves writing C++ Linked Lists – actually a Multi-List which we'll talk about next lecture)**