

# Computer Programming

Dr. Deepak B Phatak

Dr. Supratik Chakraborty

Department of Computer Science and Engineering  
IIT Bombay

**Session: Programming using structures – Part 2**

# Quick Recap of Relevant Topics

---

- Brief introduction to object-oriented programming
- Defining structures in C++
- Accessing members of structures
- Initializing and copying structures
- Programming using structures
  - Checking out a book in a library information management system

# Overview of This Lecture

---

- More programming using structures
  - Implementing other functionalities in library information management system

# Acknowledgment

---

- Some examples in this lecture are from  
**An Introduction to Programming Through C++**  
by Abhiram G. Ranade  
**McGraw Hill Education 2014**
- All such examples indicated in slides with the citation  
**AGRBook**

# Recall: Library Information Management System

## [Ref. AGRBook]

---



- Every patron has a numerical id
- Every book has an accession number
- **Check out:** A patron can check out upto 3 books at any time
- **Claim:** If X has not already checked out 3 books, she can claim a book checked out by Y

When Y returns the book, it is held for X and cannot be lent to others

- **Return:** A patron can return a book checked out by her at any time

# Recall: Relevant Structures and Arrays

```
struct Book {  
    char title[50];  
    char authors[500];  
    double price;  
    int accNum;  
    bool checkOutStatus;  
    int claimantId;  
};
```

**Book** libraryShelf[1000]

# Recall: Relevant Structures and Arrays

```
struct Book {  
    char title[50];  
    char authors[500];  
    double price;  
    int accNum;  
    bool checkOutStatus;  
    int claimantId;  
};
```

**Assume checkOutStatus and claimantId of all elements of array libraryShelf initialized to "false" and "-1" respectively**

**Book** libraryShelf[1000]

# Recall: Relevant Structures and Arrays

---



```
struct Patron {  
    char name[50];  
    char address[100];  
    int uniqueId;  
    int numBooksChkdOut;  
    int claimdBookAccNum;  
};
```

**Patron** libraryPatrons[200]



# Recall: Relevant Structures and Arrays

**Assume numBooksChkdOut and claimdBookAccNum for all elements of array libraryPatrons initialized to “0” and “-1” respectively**

```
struct Patron {  
    char name[50];  
    char address[100];  
    int uniqueId;  
    int numBooksChkdOut;  
    int claimdBookAccNum;  
};
```

**Patron** libraryPatrons[200]

# C++ Function for Claiming a Book

`currPatron` (of type **Patron**) to claim `currBook` (of type **Book**)

- Check if `currPatron` has already checked out 3 books or already has a pending claim
  - If so, print appropriate message and return
  - Otherwise,
    - \* If `currBook` is not checked out by anybody, print appropriate message and return
    - \* Otherwise, if `currBook` not already claimed by a different patron
      - Store `currBook.accNum` in `currPatron.claimdBookAccNum`
      - Store `currPatron.uniqueId` in `currBook.claimantId`

# C++ Function for Claiming a Book

```
// PRECONDITION: Members of currBook and currPatron are  
// properly initialized – no garbage values
```

```
void claimBook(Patron &currPatron, Book &currBook)  
{
```

**... Code for claiming a book ...**

```
}
```

```
// POSTCONDITION: Register a claim of currPatron on currBook, if  
// allowed. Update members of currPatron, currBook appropriate
```

## C++ Function for Claiming a Book

```
void claimBook(Patron &currPatron, Book &currBook)
{
    if ((currPatron.numBooksChkdOut == 3) ||
        (currPatron.claimdBookAccNum != -1)) {
        cout << "Sorry! Patron " << currPatron.name;
        cout << " no longer allowed to claim any book." << endl;
        return;
    }
}
```

**... Code for claiming a book (part 1) ...**

}

## C++ Function for Claiming a Book

```
if (currBook.checkOutStatus == false) {  
    cout << "Book " << currBook.title;  
    cout << " (Acc. # " << currBook.accNum << ") ";  
    cout << "not yet checked out. No need for a claim." << endl;  
    return;  
}  
if ((currBook.claimantId != -1) &&  
    (currBook.claimantId != currPatron.uniqueId)) {  
    cout << "Sorry! Book already claimed by a patron." << endl;  
    return;  
}
```

**... Code for claiming a book (part 2) ...**

# C++ Function for Claiming a Book

```
currPatron.claimdBookAccNum = currBook.accNum;  
currBook.claimantId = currPatron.uniqueId;  
return;
```

## A strange scenario:

**What happens if a patron tries to claim the only book she has already checked out? Why does this happen?**

# C++ Function for Returning a Book

```
// PRECONDITION: Members of currBook and currPatron proper  
//               initialized.  
//               currPatron had indeed checked out currBook  
void returnBook(Patron &currPatron, Book &currBook)  
{
```

**... Code for returning a book ....**

```
}
```

```
// POSTCONDITION: Update members of currPatron, currBook  
//               to register return of currBook by currPatron
```

# C++ Function for Returning a Book

```
void returnBook(Patron &currPatron, Book &currBook)
{
    if (currPatron.numBooksChkdOut > 0) {
        currPatron.numBooksChkdOut--;
        currBook.checkOutStatus = false;
    }
    else { cout << "Something strange! ";
           cout << "Returning somebody else's book?" << endl;}
    return;
}
```



# C++ Function for Returning a Book

```
void returnBook(Patron &currPatron, Book &currBook)
{
    if (currPatron.numBooksChkdOut > 0) {
        currPatron.numBooksChkdOut--;
        currBook.checkOutStatus = false;
    }
}
```

**A second strange scenario:**

**What happens if a patron tries to return a book checked out by somebody else? Why does this happen?**

```
}
```

# Did We Keep Track of Everything?

---

- Observations

When a patron checks out a book, the unique id of the patron is not recorded with the book.

Neither are the accession numbers of books checked out by a patron recorded with the patron

If we kept track of the above information, can we detect if

- a patron is indeed returning a book she borrowed?
- a patron is trying to claim a book she has checked out?

## A Desirable (Late) Change

---

- Add another member to struct Book  
    int borrowerId;
- Add another member to struct Patron  
    int borrowedAccNum[3];

**How do we change things now, after having implemented  
checkOutBook, claimBook and returnBook?**

**Object-oriented modular programming to our rescue!**

# Incrementally Changing Structure Definitions

```
struct Book {  
    char title[50];  
    char authors[50];  
    double price;  
    int accNum;  
    bool checkOutStatus;  
    int claimantId;  
    int borrowerId;  
};
```

**All initialized  
to -1**

```
struct Patron {  
    char name[50];  
    char address[100];  
    int uniqueId;  
    int numBooksChkdOut;  
    int claimdBookAccNum;  
    int borrowedAccNum[3];  
};
```

**Patron** libraryPatrons[200]

**Book** libraryShelf[1000]

# Incrementally Modifying checkOutBook

... After ascertaining that currPatron can indeed check out currBook ...

```
for (int i = 0; i < 3; i++) {
```

```
    if (currPatron.borrowedAccNum[i] == -1)
```

```
        {currPatron.borrowedAccNum[i] = currBook.accNum; break;}
```

```
}
```

```
currBook.borrowerId = currPatron.uniqueId;
```

```
currBook.checkOutStatus = true; currPatron.numBooksChkdOut ++;
```

```
if (currBook.claimantId == currPatron.uniqueId)
```

```
{currPatron.claimdBookAccNum = -1; currBook.claimantId = -1;}
```

# Incrementally Modifying claimBook

---

... After ascertaining that currPatron can claim currBook ...

```
if (currBook.borrowerId == currPatron.uniqueId) {  
    cout << "Claimed book checked out by same patron." << endl;  
}  
else {  
    currPatron.claimdBookAccNum = currBook.accNum;  
    currBook.claimantId = currPatron.uniqueId;  
}
```

# Incrementally Modifying returnBook

```
void returnBook(Patron &currPatron, Book &currBook)
{
    if (currBook.borrowerId == currPatron.uniqueId) {
        currPatron.numBooksChkdOut--; currBook.checkOutStatus = false;
        for (int i = 0; i < 3; i++) {
            if (currPatron.borrowedAccNum[i] == currBook.accNum) {
                currPatron.borrowedAccNum[i] = -1; break;
            }
        }
        currBook.borrowerId = -1;
    }
    return ;}
}
```

# Summary

---

- More on programming using structures
  - claimBook and returnBook in library information management system
- Incrementally changing structure definitions and functions using them
  - Modular, incremental software development