Computer Programming

Dr. Deepak B Phatak
Dr. Supratik Chakraborty
Department of Computer Science and Engineering
IIT Bombay

Session: Introduction to Object-Oriented Programming -- Structures

Quick Recap of Relevant Topics

IIT Bombay

- Basic ingredients for programming in C++
  - Data types, variables, constants, arrays
  - Sequential and conditional statements
  - Iterative constructs
  - Functions

Overview of This Lecture

- Basic idea of object-oriented programming
- Introduction to structures in C++

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**

Designing A Complex Program [Ref. AGRBook]

- We want to design a book check out/return/claim management system of a small library
- How does the system work?
  - Every patron has a unique 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

    No late charges!

Designing a Complex Program

IIT Bombay

Should we jump into writing code?

- Keep adding variables/arrays and functions as and when needed

For each book:

- Title and authors (char arrays)
- Price in Indian Rupees (double)
- Accession number (int)
- Check out status (bool)
- Claimant's id (int), if any

For each patron:

- Name and address (char arrays)
- Unique id (int)
- Number of books checked out (int)
- Claimed book's accession number (int), if any

Designing a Complex Program

IIT Bombay

Should we jump into writing code?

- Keep adding variables and function as and when needed

For each book:

- Title and authors (char arrays)
- Price in Indian Rupees (double
- Accession number (int)
- Check out status (bool)
- Claimant's id (int), if any

For each patron:

- Name and address (char array)
- Unique id (int)

1000 books implies 1000-sized arrays of

- char array (book title)
- char array (authors' names)
- double (price)
- int (accession number)
- bool (check out status)
- int (claimant id)

Perhaps manageable, but not natural to split information about same book in six different arrays

Designing a Complex Program

IIT Bombay

Must store information about a book at same index in all arrays

title[5], authors[5], price[5], accNum[5], checkOutStatus[5], claimantId[5] must refer to the same book

1000-sized arrays of
- char array (book title)
- char array (authors' names)
- double (price)
- int (accession number)
- bool (check out status)
- int (claimant id)

How do we find claimant for the book with acc. number 123456?

Find "index" such that accNum[index] is 123456
Then access claimantId[index]

Unnecessarily convoluted !!!

Is There A Better Way?

IIT Bombay

- Wouldn't it be nice if we could do the following?
  - Group all information about a book into a "book entity"
  - Have an array of these "book entities"

Once we search this array and find an entity whose accession number is 123456, we can read off the claimant information for this array element.

- Similarly, we could
  - Group all information about a patron into a "patron entity"
  - Have an array of these "patron entities"

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

C++ keyword

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

Structure type name (can be used like other data type names such as int, char)

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

Members of structure Book

Members of structure Book

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

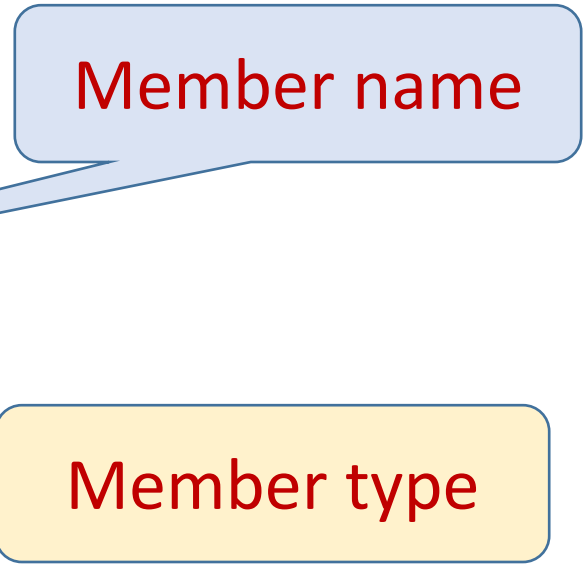
Members can be arrays or variables of other data types (including **other** structures)

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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



Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

Declaring variables of type **Book**

Book myChoice, yourChoice;

Note similarity with declaring

int myInt, yourInt;

Declaring an array of type **Book**

Book libraryShelf[1000];

Note similarity with declaring

int rollNumbers[1000];

Structures in C++

IIT Bombay

- C++ provides **structures** to group a set of variables of possibly different data types together

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

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

Overall Design Philosophy

IIT Bombay

- Identify **entities** (physical or conceptual) involved in the working of the system
  - E.g., Books and Patrons
  - Entities also called **objects**
- Think of system functionality in terms of **operations on and interactions between objects**
  - E.g., Patron checking out a book, patron claiming a book
- **Abstract** away (hide) details not necessary for an operation
- Implement system modularly by focusing on entities, their interfaces and their interactions

OBJECT-ORIENTED PROGRAMMING (in one slide)

Entity or Object

IIT Bombay

- Contains information specific to the object
  - "Fixed" information – usually doesn't change as objects interact

    Name of patron, title of book, authors of book
  - "State" information – can change as objects interact

    Check out status of book, number of books checked out by patron
- Unambiguous, well-defined boundaries
  - Clear specification of what information is part of an object

    When a patron claims a book, where is the information stored?

    Claimant's id is in "Book" object

    Claimed book's accession number is in "Patron" object

Interactions between Objects

IIT Bombay

- Ideally, every interaction between two objects should happen through well-defined interfaces of objects
  - Allows hiding unnecessary details of object from programmer

    When a patron checks out a book by accession number, do we need to access book's title, author's name or price?

  - Abstraction, data encapsulation: To be covered later in course

Summary

- Motivation for object-oriented programming
- Introduction to structures as objects in C++

Designing a Complex Program

IIT Bombay

Should we jump into writing code?

- Keep adding variables and function as and when needed

For each book:

- Title and authors (char arrays)
- Price in Indian Rupees (double)
- Accession number (int)
- Check out status (bool)
- Claimant's id (int), if any

For each patron:

- Name and address (char array)
- Unique id (int)
- Number of books checked out (int)
- Claimed book's accession number (int), if any

1000 books implies

- 2000 character arrays (title, authors)
- 1000 double variables (price)
- 1000 int variables (acc. numbers)
- 1000 bool variables (check out status)
- 1000 int variables (claimant ids)

Not quite manageable
Recipe for disaster!