Tutorial on Project 1 and OOP*
[Supplements the Project Handout]

Advanced Programming Techniques
CS292

* Originally prepared by Nazia and Zubair in Winter 2003
Running the Skeleton Code

- Make a separate directory for your code ...
  $ mkdir proj1
- Now move the code to this directory ...
  mv <source> <destination>
- Compile many files together:
  bash-2.03$ g++ -o output
  Time_first_class.cc
  AppointmentBook.cc Date.cc
  SimpleInput.cc
bash-2.03$ g++ -o output Time_first_class.cc AppointmentBook.cc Date.cc SimpleIn
bash-2.03$ output

Inside AppointmentBook Constructor

*******************************

***** Welcome to the C++ ORGANIZER *****
**** The key to your organized future ****
*******************************

Enter Choice or ? for help: ?

Valid commands are:

P/Print print schedule for day
A/Add add a new event
C/Cancel cancel an already scheduled event
F/Find find events by phrase
Q/Quit quit
H/Help this help menu

Enter Choice or ? for help: bash-2.03$ g++ -o output Time_first_class.cc AppointmentBook.cc Date.cc SimpleIn

**** Invalid command, try again ****

Enter Choice or ? for help:
The rest of the Use Cases
Print Schedule for a day

• Press P (Print Schedule for a day)
• Program will ask you for date
• Program will print the events listed for that date.

Alternative 1
• In case of incorrect input program will loop until correct information is added

Alternative 2
• In case No event is scheduled for a day, user is informed so.
Add Use Case

- Press A (Add a new event)
- Program will ask you for a title of event
- Program will ask you for date of event
- Program will ask you for time of event
- Program will ask you for location of event
- Program will ask you for comments

Alternative 1
- In case of any incorrect input program will loop until correct information is added

Alternative 2
- In case Event overlaps with another event, program will report this and will not add this event.
Cancel Use case

- Press C (Cancel an Event)
- Program will ask you the date
- Program will list events on that day with serial numbers
- User enters the serial number of event he wants to delete
- Program will delete the event

Alternative 1
- In case of incorrect input program will loop until correct information is added

Alternative 2
- In case No such event exists, user is informed so and no event deleted.
Find Use case

- Press F (Find an Event)
- Program will ask you to enter a string
- Program will try to match that string in the entire database
- Program will list the events where the string matched with serial numbers
- User enters the serial number of event he wants to view
- Program will Print the event

Alternative 1
- In case of incorrect input program will loop until correct information is added

Alternative 2
- In case No such event exists, user is informed so
Other Use cases

• Help use case has already been implemented but you are welcome to add feature
• Quit use case is also trivial but may be you will like to add some checking
The Class Diagram

Program Classes

SimpleInput  AppointmentBook

Organizer

DailySchedule

TimeInterval  Event

Utility Classes

Date

Time

user
### SimpleInput

<table>
<thead>
<tr>
<th>Static Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>static ReadLine()</td>
</tr>
<tr>
<td>static ReadInteger()</td>
</tr>
<tr>
<td>static ReadDate()</td>
</tr>
<tr>
<td>static ReadTime()</td>
</tr>
</tbody>
</table>

**Responsibilities:**
- Take Input from cmd
- Validate input
**Responsibilities:**
- Records time in hours, minutes and seconds.

<table>
<thead>
<tr>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>int hours</td>
</tr>
<tr>
<td>int minutes</td>
</tr>
<tr>
<td>int seconds</td>
</tr>
<tr>
<td>void setTime()</td>
</tr>
<tr>
<td>void printTime()</td>
</tr>
</tbody>
</table>

For Time can we:
- Create?
- Message?
- Print?
- Compare?
- Shift?
<table>
<thead>
<tr>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What attributes?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>What methods this class should support?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Responsibilities:**
- Start by identifying its responsibilities
<table>
<thead>
<tr>
<th><strong>Event</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>string name</td>
</tr>
<tr>
<td>string location</td>
</tr>
<tr>
<td>string comment</td>
</tr>
<tr>
<td>TimeInterval eTime</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Responsibilities:</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>- Stores an Event</td>
</tr>
</tbody>
</table>
**TimeInterval**

<table>
<thead>
<tr>
<th>Time</th>
<th>sNeTime[2];</th>
</tr>
</thead>
<tbody>
<tr>
<td>Date</td>
<td>sNeDate[2];</td>
</tr>
</tbody>
</table>

- Who will use it and what functionality will they require
- Do we have to calculate something internally?

**Responsibilities:**
- can record and compare time intervals
<table>
<thead>
<tr>
<th><strong>DailySchedule</strong></th>
<th><strong>Organizer</strong></th>
</tr>
</thead>
</table>

**Responsibilities:**
- Manage a list of event schedules on a particular day

**Responsibilities:**
- Tracks the schedule of all various days

<table>
<thead>
<tr>
<th><strong>AppointmentBook</strong></th>
</tr>
</thead>
</table>

**Responsibilities:**
- Interactive portion of the program
Object Oriented Analysis and Design

- Programming = Art + Science
- How to manage complexity?
- Classes and Data Abstraction
- Choosing the right abstractions
  - What data is required to represent the objects of this class?
  - What behavior will I need from that object?
- Refining the abstractions
  - Highly Cohesive
  - Loosely coupled
A Good Class\textsuperscript{1} (notice grading highly agrees with this)

- Provides a crisp abstraction of something drawn from the vocabulary of the problem domain or the solution domain.
- Embodies a small, well defined set of responsibilities and carries them all out very well.
- Provides a clear separation of the abstraction’s specification and its implementation.
- Is understandable and simple yet extensible and adaptable.

\textsuperscript{1} Grady Booch, 1994, OOAD with Applications, Addison-Wesley
Relationships of Classes

- There are 3 kinds:
  - Dependency (Using)
  - Generalization (Inheritance)
  - Association (+aggregation or containment)

- We are mainly using Association.
Testing your classes

• Good Specification will allow you to know what to test even before you write code

• Unit Testing: Make sure that class is working properly in isolation.
  – For correct input we get correct output
  – Decision constructs and loops are working properly especially on limit.

• Regression Testing: Classes work properly when working in a group.
How to Comment

• Use the Sample code as guideline
• “Comments are deodorants”, ideally code should be self explanatory.
• Tip of the Day: Any programmer can write code that computers understand, good programmers write code humans can understand.
• Abstraction Comments (AC) are a must.
• AC is a contract between user (of the function) and coder.
• AC can be less formal (as in skeleton code or formal)
• Now a days AC is used to generate documentation (JAVADOC).
Directives

/*  EXAMPLE.H - Example header file  */
#if !defined( EXAMPLE_H )
#define EXAMPLE_H

class Example
{
    ...
};
#endif // !defined( EXAMPLE_H )

• To handle problem with nested commenting
    #if 0
        .... //code with many c style comments
    #endif
C++ Vectors

- Vectors contain contiguous elements stored as an array. Accessing members of a vector or appending elements can be done in constant time, whereas locating a specific value or inserting elements into the vector takes linear time.
- Constructors: methods to create vectors
- Operators: assign and compare vectors
- `assign()`: assign elements to the vector
- `at()`: returns an element at a specific location
• **back()** returns the last element
• **begin()** returns an iterator to the first element
• **capacity()** the number of elements the vector can hold
• **clear()** removes all elements
• **empty()** true if the vector is empty
• **end()** returns an iterator to the last element
• **erase()** removes elements
• **front()** returns the first element
• **get_allocator()** returns the vector's allocator
• **max_size()** returns the maximum number of elements that the vector can hold
- `insert()` insert elements into the vector
- `pop_back()` removes the last element
- `push_back()` adds an element to the end of the vector
- `rbegin()` returns a reverse iterator to the end of the vector
- `rend()` returns a reverse iterator to the beginning of the vector
- `reserve()` sets the minimum capacity of the vector
- `resize()` change the size of the vector
- `size()` returns the size of the vector
- `swap()` exchange two vectors
Read The Handout Carefully
&
Strictly Follow the Submission Guidelines
Questions?