CENTER FOR TEXTUAL STUDIES AND
DIGITAL HUMANITIES

DIGH 401 - Introduction to Computing

Fall Semester 2013

Week 11

Today's Class

- Leap years

- Data Structures
    - array consideration
    - stacks
    - queues
    - collections and dictionaries
    - hash tables
    - graphs

Conceptual Design Specification

- use your outline for a conceptual project (DIGH 400) as the basis
- this project focuses on the conceptual implementation rather than using a proposal model
- think about how to plan the construction of the programming or output of the project
- choose a 'software engineering model' to help plan and outline your project

eg: you might choose to base the process on the waterfall model using the steps for analysis, design, implementation, and testing

- detail each step of your model for your project
- implementation and testing phases can be conceptual
- explain in an introduction or overview why you chose your particular software model
- models can also be used, such as flowcharts or UML EXAMPLE

Project Presentation 2

4th December 2013 - ~10 mins per paper & questions from the class

- conference style short paper
- outline your design specification etc
- use template for guidance
- UML or flowchart outlines

Leap Years

```
<?php

for ($i=1; $i<=2012; $i++) {

	if ( (($i % 4 == 0) && !($i % 100 == 0)) || ($i % 400 == 0)) {
		echo 'Year = '.$i.' & Number of days = 29. Leap year!<br />';
	}
	else {
		echo 'Year = '.$i.' & Number of days = 28<br />';
	}

}

?>
```

Example

Data

- every computer program needs to store data
- dump data into a variable
- need a separate variable for each chunk of data
- we can also store data together, for example
    - arrays
    - structures

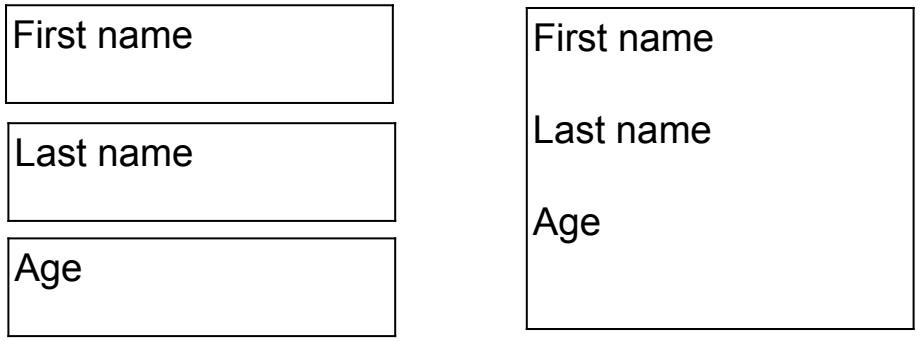
Data Structures

- group separate variables together
- store multiple variables within another variable
- user-defined data type
- cannot use a structure until you declare a variable to represent the structure

```
int[] anArray;
```

```
$students = array();
```

Data Structures



Data Structures - storing & retrieving data

- identify the variable that represents that structure
- identify the specific variable inside the structure to use

Create array: $students = array();

Insert a value: $students[0] = "Emma"

Output a value: echo $students[0];

Example

Data Structures - storing & retrieving data

```
<?php

$students = array();

$students[0] = 'Emma';

echo $students[0];

?>
```

Data Structures - Stacks

- data structure providing two basic methods
    - push
    - pop
- buffering a stream of objects
- last in is first out (LIFO)
- data items for processing, instructions for processing, instructions pending execution...

A stack of plates...

```
array_push($students, 'Catherine');

array_pop($students);
```

NB: deletes last element of the array

Example

Data Structures - Stacks

```
<?php

$students = array();

$students[0] = 'Emma';

echo $students[0].'<br />';

array_push($students, 'Catherine');

echo $students[1];

?>
```

Data Structures - Stack implementation

plates again...

1. first plate begins the pile
2. next plate is placed on top and so on
3. a plate may only be removed from the top
4. order of pushing or popping plates is arbitrary
5. there will always be a certain number of plates on the pile
6. pushing a plate increase the pile by one, popping a plate decreases the pile by one
7. you cannot pop a plate off an empty pile
8. you cannot push a plate onto a full pile, assuming there is a maximum number for the pile

7 & 8 will need to be monitored.

Data Structures - Stack implementation

plates using a stack

1. We can create an array to hold the plates.

2. we then maintain a pointer to the top of the array, which acts as a marker to tell us the index in the array where the next plate is pushed and stored

3. we'd probably also set a maximum for the stack, and set the first top pointer to zero.

4. then we would need to be able to monitor the status of the stack to check total number of plates, and the current top pointer index.

5. so, we now know that the stack is empty when the pointer is zero, and it will be full when the top pointer is equal to the size of the stack array, as defined by the user, for example

6. when we push a new plate on to the pile we simply check the current number of plates, the current index for the top pointer, and then increment the top pointer by 1 if there is space.

7. to pop a plate from the pile we need to check that a plate exists, and then decrease the index of the top pointer by one.

Data Structures - Queues

- similar to a stack except you join the queue at one end and leave at the other
- first in is first out (FIFO)
- buffering a stream of objects
- we can use an array for storing items in the queue
- two pointers in the queue
    - one to indicate where the next item should be stored
    - another to indicate the location of the next item to be retrieved

array_shift($students); NB: removes first array element & outputs

Example

Data Structures - Queues

```
<?php

$students = array();

$students[0] = 'Emma';

echo $students[0].'<br />';

array_push($students, 'Catherine');

echo $students[1].'<br />';

echo array_shift($students);

?>
```

Data Structures - Collections and Dictionaries

Conceptual Intro

- use a collection or dictionary to store different data types together
- a collection acts like a resizable array
- a collection holds different data types
- a collection identifies each chunk of data with a number
- a dictionary acts like a collection that identifies each chunk of data with a unique key

Data Structures - Using a Collection

- acts like a super array that can grow and expand...
- a collection can store different data types, or even other data structures such as an array
- not all programming languages offer direct support for the collection data structure

Python
C or Pascal
C#, .Net etc
Java

Data Structures - Adding data to a Collection

- create a collection by declaring a variable as a collection

    MyStuff as New Collection

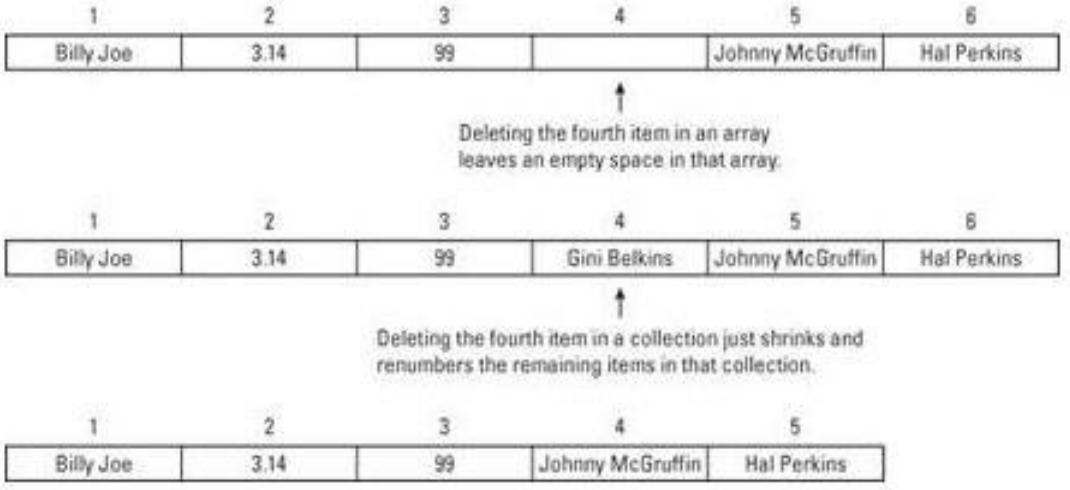
- when you create a collection it will contain zero items
- a collection will automatically expand when you add a new chunk of data
- each new element added to a collection is tacked onto the end
- every element in a collection gets numbered
- the first element is given a number of 1, then 2, and so on...
- you can also specify where to add new data in a collection using an index number

Data Structures - Deleting data from a Collection

- add data and delete data as necessary
- to delete data you must specify the location of that data by defining an index number

```
MyStuff.Remove(4);
```

- when you delete data from a collection, the collection automatically renumbers the rest of its data to avoid empty spaces



Example - unset()

Data Structures - Deleting data from a Collection

```
<?php

$students = array();

$students[0] = 'Emma';

echo $students[0].'<br />';

array_push($students, 'Catherine', 'Angela', 'Megan');

echo $students[1].'<br />';

unset($students[2]);

echo 'key 2 has now been unset from the array<br />';

print_r($students);

echo '<br />';

$i = 0;
$newStudents = array();
foreach ($students as $key => $value) {
      $newStudents[$i] = $value;
      $i++;
}

print_r($newStudents);

?>
```

Data Structures - Identification with keys

- collections identify data by their position in the collection
- if you don't know the specific location you will need to search the entire collection
- collections also give you the option to identify data with a descriptive string or 'key'
- allows you to use a descriptive string to help identify the data
- identify data by its given location or by a key
- criteria must be met when adding a key to data
    - the key must be added at the same time as the data
    - every key must be a string
    - every key must be unique

Associative Arrays in PHP

```
$students['teacher'] = 'Megan';
```

Example

Data Structures - Identification with keys

```
<?php

$students = array();

$students[0] = 'Emma';

echo $students[0].'<br />';

array_push($students, 'Catherine', 'Angela');

$students['teacher'] = 'Megan';

print_r($students);

echo '<br />Teacher = '.$students['teacher'];

?>
```

Data Structures - Searching and retrieving data from a Collection

- two direct ways to search and retrieve data from a collection
    - use the index number of the data
    - use the key of the data
- you need to keep a check of index numbers to be able to use that search option
- using index numbers to search is easy, but also unreliable
- keys are a good idea when we add an element
- use the key to find the required data
- you can always retrieve data from a collection with either its key or index no.

```
$students['teacher'];
```

Data Structures - Using a Dictionary

- similar to a collection but with two potential advantages
    - searching for data is much faster
    - key can be any type, including strings and numbers
- dictionaries use a 'hash tables' data structure
- keys must be used in a dictionary
- declare a variable for a dictionary

```
MyBook as New Dictionary;
```

Data Structures - Adding data to a Dictionary

- add data to a dictionary by defining both the data and the key to associate

```
MyBook.put(key, value);
```

- each time you add data to a dictionary you must add a corresponding key

Data Structures - Searching and retrieving data from a Dictionary

- identify the dictionary variable and key required
- use the key to find specific values

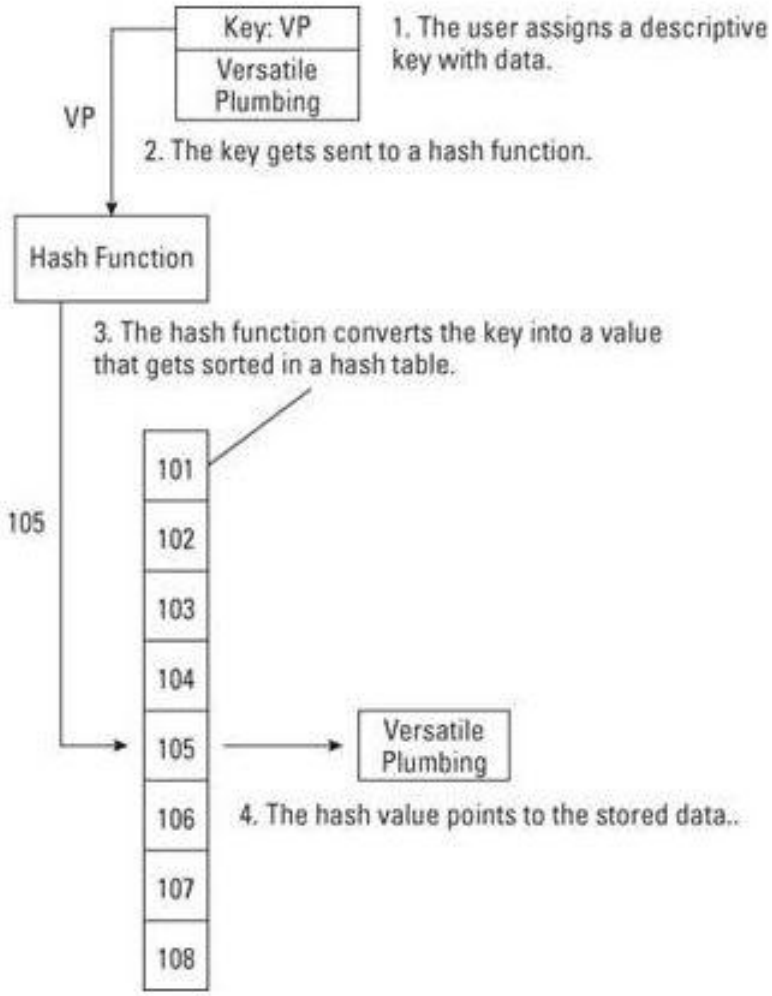
```
MyBook.get(key);
```

- searching a dictionary is more efficient because a computer does not need to search through the entire dictionary sequentially
- instead it searches through data using a hash table
- hash table makes searching faster by dividing data into distinct sections

Data Structures - Hash Tables

- dictionaries use hash tables to speed up searching
- a hash table takes the keys used to identify data and converts that key into a hash value
- hash value is stored in a sorted list

DIGH 401 - Introduction to Computing



Data Structures - Retrieving data from a Hash Tables

- to retrieve data you give the computer the key associated with the required data
- a hash function is used to convert the key to a value
- this value is compared to a list of stored values in the hash table
- when a match is found the data associated with the key is returned