***Summer 2012 Course Syllabus (Draft)***
CS/SE 3376.0I1 **– C/C++ Programming in a UNIX Environment**
Online Course delivered through elearning.utdallas.edu

**Instructor:** Dr. Jey Veerasamy
Office: ECSS 3.231 or Virtual
Office Phone: 972-883-4241 or online
E-Mail: jeyv@utdallas.edu

**Office Hours:** TBD

**TAs:**
Name: TBD
Office: TBD
Office hours: TBD
Email: TBD

**Course Pre-requisites, Co-requisites, and/or Other Restrictions**

*Computer Science I and II (CS 1337 and CS 2336 or equivalent courses)*
*Basic Object Oriented Programming Skills*

**Course Description**

CS 3376 C/C++ Programming in a UNIX Environment (3 semester hours) Advanced programming techniques utilizing procedural and object oriented programming in a UNIX environment. Topics include file input and output, implementation of strings, stacks, queues, lists, and trees, and dynamic memory allocation/management. Design and implementation of a comprehensive programming project is required. Prerequisite: ECS 2336 or equivalent. (3-0) S

**Student Learning Objectives/Outcomes**

1. Ability to create classes of abstract data consisting of variables and functions
2. Ability to utilize C++ constructors, copy constructors, and destructors
3. Ability to utilize C++ OOP features using static member data and member functions
4. Ability to utilize C++ File and Stream Input/Output Processes
5. Ability to generate reusable code using inheritance
6. Ability to use polymorphism and virtual member functions
7. Ability to generate reusable code using templates
8. Ability to create and utilize dynamic data structures such as linked lists
9. Ability to create and utilize recursive functions

**Required Textbooks and Materials**

Starting Out with C++: Early Objects (7th Edition) by *Tony Gaddis, Judy Walters, and Godfrey Muganda*
ISBN-13: 978-0-13-607774-9 ISBN-10: 0-13-607774-9 (6th edition will be ok too)

Beginning Linux Programming, 4th edition by *Neil Matthew, Richard Stones*
ISBN-10: 0470147628 ISBN-13: 978-0470147627 (3rd edition will be ok too)

**Suggested Course Materials**

C++ language tutorial http://www.cplusplus.com/files/tutorial.pdf

**Assignments & Academic Calendar** *(subject to change throughout the term)*

| **Dates** | **Topics (multimedia lectures in elearning)** | **Activities** |
|---|---|---|

| | | |
|---|---|---|
| Week 1 (May 30 - June 3) | Review of Syllabus<br>UNIX, Linux and C overview | Install XShell |
| Week 2 (June 4 - 10) | Working with Shell<br>C/C++ fundamentals | Read Chapters L1 & L2<br>Read Chapters C1-C6<br>Assignment 1 due |
| Week 3 (June 11 - 17) | Working with Files, Linux Environment<br>Classes and Objects, Arrays | Read Chapters L3 & L4<br>Read Chapters C7 & C8<br>Assignment 2 due |
| Week 4 (June 18 - 24) | Pointers & dynamic memory allocation<br>More Classes & OOP | Read Chapters C10 & C11<br>Assignment 3 due |
| Week 5 (June 25 - July 1) | Strings<br>Advanced File and I/O operations<br>Recursion | Read Chapters C12-C14 |
| Week 6 (July 2 - 8) | Inheritance & Virtual methods<br>Exceptions & Templates<br>Template classes | Read Chapters C15 & C16<br>Assignment 4 due<br>**Mid-term Exam:**<br>**Date, Time & Place TBD** |
| Week 7 (July 9 - 15) | Linked lists, Stacks & Queues,<br>Binary Trees<br>Development Tools - Makefiles, Debugging | Read Chapters C17-C19<br>Read Chapters L9 & L10<br>Assignment 5 due |
| Week 8 (July 16 - 22) | Processes and Signals<br>POSIX Threads, Semaphores | Read Chapter L11 & L12<br>Assignment 6 due |
| Week 9 (July 23 - 29) | IPC: Pipes | Read Chapter L13 |
| Week 10 (July 30 - August 5) | Shared Memory & Message Queues | Read Chapter L14<br>Assignment 7 due |
| Week 11 (August 6 - 12) | Namespace & other C++ topics | **Final Exam:**<br>**Date, Time & Place TBD** |

**Grading Policy**

Programming assignments, e-classroom participation and exams determine grades. The final grade will be composed as follows:

| | | |
|---|---|---|
| Assignments | 28% | (7 assignments * 4% each = 28%) |
| Participation | 22% | (11 weeks * 2% each = 22%) |
| Mid-term Exam | 25% | |
| Final Exam | 25% | |

Letter grades will be assigned as follows:

| | | | | | |
|---|---|---|---|---|---|
| 98-100 | A+ | 92-97 | A | 90-91 | A- |
| 88-89 | B+ | 82-87 | B | 80-81 | B- |
| 78-79 | C+ | 72-77 | C | 70-71 | C- |
| 68-69 | D+ | 62-67 | D | 60-61 | D- |
| Below 60 | F | | | | |

**Course & Instructor Policies**

This is offered as 100% online course, so it is possible to complete this course wherever you are! We will make max use of elearning features to run this course. Office hours will be conducted virtually as well. More details will be provided later. If you will be in Dallas area during Summer, you are expected to come to campus to take mid-term and final exams. Otherwise, work with the instructor to make alternate arrangements for proctoring the exams locally (local testing center, local university, etc. - more details will be coming soon).

**Warning:** Typical online course requires the students to be very self-motivated. Online courses move fast & cover lot of content every week. So, it is very important to keep up & avoid falling behind. Unlike a face-to-face classroom, instructor is NOT there in person to remind about the upcoming assignments & due dates. Students should plan their weeks in advance and spend their time wisely. If you have never taken an online course, do a web search on "how to succeed in online courses" to learn more.

Video lectures and discussion questions will be posted every week in elearning. Each student is expected to contribute at least 2 meaningful posts in online forums. That is how you will learn the participation grade for the week. You cannot catch up on participation once the week is over. A typical week runs from Monday to Sunday. Students are encouraged to login to elearning at least every 2 days once and catch up on valuable discussions.

Course credit is only given for work assigned in the course schedule.  No extra work will be assigned nor will extra credit be given for any extra work performed by a student. However, there is only one extra credit item in this course: Active participation in elearning (well beyond the minimum requirement of 2 posts/week). When computing the final grade towards the end of the course, instructor may assign upto 5% additional credit based on your active role in the classroom.

Both exams are closed book and closed notes. Exams will focus more on concepts and less on details. Necessary documentation will be provided to avoid the need for memorization as much as possible. All make-up exams are scheduled during the week following the actual exam date at the discretion of the instructor. Make-up exams are only given to those students who coordinate the missing of an exam prior to the originally scheduled exam date.

There will be 7 assignments in this course & 6 of them are programming assignments. Each assignment contributes 5% to the final grade. All these assignments should be done in Linux and you can hand-in your assignments directly in Linux. We will NOT use elearning to submit the assignments, however all the grades will still be in elearning. More details on submission steps will be given with assignment #1.

I encourage everyone to submit the assignments 1 or 2 days early. Do not wait until the last minute to submit it. But I do understand things happen and occasionally you may not be able to submit assignments on time. My policy is to assess 1% penalty for every 2 hours. For example, if you submit the assignment exactly 1 day later, 12% penalty will be assessed. Late assignments will be accepted up to 4 days. You won't be able to submit it after 4 days and your assignment grade will be set to 0. Only exception to this late policy is serious medical condition, for which you will need to submit proof of doctor certificate. Please do not send emails requesting for extension or penalty waiver. In case of cs1 server outage, you will be given grace period to submit it.

TA is responsible for grading the assignments. Instructor is responsible for grading the exams. So, contact the TA directly for any grading related discrepancies for assignments. If you cannot resolve it with TA, bring it to instructor's attention.

**Assignments (subject to change - official ones will be in elearning):**

**Assignment #1:** Linux commands to do complex jobs

In this assignment, you will come up with Linux commands to process the data files and arrive at the expected output.

**Description**: http://noosphere.princeton.edu/data/basket_csv.html
**Data**: ~veerasam/students/data_files/eggdatareq.csv in cs1 server.

1. How many data records with record type "13" are present in data file?
2. Are there any 2 times (lines) in which series of data is exactly same?
3. Each data point seems to have a value within the range 90 and 100, few values are outside this range too. Find the distribution of data points for the whole data file.

**Description**: http://www.cyberciti.biz/faq/understanding-etcpasswd-file-format/
**Data**: ~veerasam/students/data_files/passwd.txt in cs1 server.

4. Find the distribution of various shells used by all the users.
5. Find all the users with group id 300 (output the full names of those users).
6. Find the distribution of first letter of last names of all users.
7. Find out how many users have duplicate last names (it is ok if you don't compute grand total).
8. Check whether each user really has a unique username and userid (Basically, write 2 separate commands to find out any duplicate usernames and userids)
9. Find out how many different base points are used for home directories? For example, /home is different from /home1, similarly /home2 is different too.
10. Find out the average # of users per group id (multiple commands are ok if needed)

Before you work on these problems, it is a good idea to copy the files into your directory.

```
cp ~veerasam/students/data_files/*  .
```

will do it.

Capture the commands in a text file called hw1.txt in Linux

1. …
2. ...

Then, when you are ready to submit, issue the following commands:

```
cp ~veerasam/.bashrc .
source .bashrc
Now, cd to appropriate directory
turnin hw1 hw1.txt
```

**Assignment #2:** Simple C program: We will implement a guessing game in C language.

Here is how we play this game:

- Think of a number between 1 and 100 in your mind. Then, your program should ask you minimal # of questions and determine your number based on your answers. Program's question format will be Is it NN (<, =, >)?
  You can respond to that question in 3 ways: < indicates that your number is less than computer's guess, = means that program guessed your number right, and > means that your number is greater than computer's guess.
- Allow the user to play this game any number of times.
- When the user is ready to quit, output average # of guesses / game.
- Implementation guidelines:
  - Use a separate function to play the game.
  - Invoke that function from main() repeatedly as needed.

When you are ready to submit, issue the following command:

```
turnin hw2 guess.c
```

If turnin command does not work, issue the following commands, then run turnin command again.

```
cp ~veerasam/.bashrc .
source .bashrc
Now, cd to appropriate directory
```

Do not utilize C++ stuff in this assignment. Avoid using arrays as well.

**Assignment #3:** C++ program to do grocery checkout.

Write a C++ program to perform grocery check-out procedure for a simple store with max 100 products. Design & use a C++ product class in your program.

When the program starts, it should read the product information file (inventory.txt - see attachment and it is also available as http://www.utdallas.edu/~veerasam/students/data_files/inventory.txt in Linux) - it contains product information (PLU code, product name, product sales type, price per pound or price per unit & current inventory level) - one product in each line.

```
4101 BRAEBURN_REG      1 0.99 101.5
4021 DELICIOUS_GDN_REG 1 0.89 94.2
4020 DELICIOUS_GLDN_LG 1 1.09 84.2
4015 DELICIOUS_RED_REG 1 1.19 75.3
4016 DELICIOUS_RED_LG  1 1.29 45.6
4167 DELICIOUS_RED_SM  1 0.89 35.4
4124 EMPIRE 1 1.14 145.2
4129 FUJI_REG 1 1.05 154.5
4131 FUJI_X-LGE 1 1.25 164.1
4135 GALA_LGE 1 1.35 187.7
4133 GALA_REG 1 1.45 145.2
4139 GRANNY_SMITH_REG 1 1.39 198.2
4017 GRANNY_SMITH_LGE 1 1.49 176.5
3115 PEACHES 1 2.09 145.5
4011 BANANAS 1 0.49 123.2
4383 MINNEOLAS 1 0.79 187.3
3144 TANGERINES 1 1.19 135.5
4028 STRAWBERRIES_PINT 0 0.99 104
4252 STRAWBERRIES_HALF_CASE 0 3.99 53
4249 STRAWBERRIES_FULL_CASE 0 7.49 67
4011 BANANAS 1 0.69 156.7
94011 ORGANIC_BANANAS 1 0.99 56.3
```

Then, program should repeatedly invoke customer check-out functionality until the store associate (user) decides to quit. As part of checkout functionality, prompt for PLU code, validate it, then the user to input weight for each product if it is sold by weight, or # of units if sold by unit. Compute the price of the item and keep up the subtotal.

Once all purchased products are rung for a customer, output the total purchase amount. If the total purchase exceeds $50, apply 5% discount to the total.

We need to keep track of inventories automatically as well. So, keep updating the inventory data along with checkout operations. When the store closes every day, product information file should be saved in output.txt in the same format as input inventory file.

Implement this assignment in 3 files: product.h, product.cpp and store.cpp. When you are ready to submit your source files, issue the following commands in Linux prompt (no need to submit anything in elearning):

```
turnin  hw3 *.h *.cpp
```

If turnin command does not work, issue the following commands, then run turnin command again.

```
cp ~veerasam/.bashrc .
source .bashrc
Now, cd to appropriate directory
```

**Assignment #4 (Team assignment with max 2 members?):** Activities in a typical restaurant without using templates

This assignments gets into dynamic memory allocation big time! Unlike all previous assignments, this is a big one. That is why it is not due for a few weeks. I hope you will start on right away and enjoy the experience!

This assignment mimics the configuration and activities happen at a typical restaurant. Input is provided in 2 files: config.txt and activity.txt. Configuration file contains how many tables, table - waiter relationship & full menu list. Activity file mimics the actual activities that happen in restaurant. After setting up the necessary objects using the configuration file, you can read the activity file and process them. Then, you should be able to answer various queries. You should not use vector or any other template in this assignment.

We will use the following classes to complete this assignment. You can access the latest version of these files & data files at ~veerasam/students/proj4 in Linux. Feel free to add more variables if needed. Avoid making drastic changes to existing variables. You need to implement all the .cpp files including class implementation and overall application functionality.

Table : status, # of max seats, # of guests, order, waiter
menu_item: item_code, name, price
Menu : list of menu items
Order : a list of menu items ordered at a table
Payment: table #, # of guests, waiter, order, total

Sample configuration file (config.txt)

```
Tables: table #, max seats
1 2
2 4
3 2
4 2
5 2
6 4
7 6
8 10
9 2
10 4
11 4
12 4
13 4
14 2
15 2
16 2
17 2
```

```
18 2

Waiters: first name followed by table list
John 1,2,5,9,11,15
Maria 3,4,6,7,17,18
Mike 8,10,12,13,14,26

Menu:     listing of the full menu: item code, name, price
A1 Bruschetta 5.29
A2 Caprese_Flatbread 6.10
A3 Artichoke-Spinach_Dip 3.99
A4 Lasagna_Fritta 4.99
A5 Mozzarella_Fonduta 5.99
E1 Lasagna_Classico 6.99
E2 Capellini_Pomodoro 7.99
E3 Eggplant_Parmigiana 8.99
E4 Fettuccine_Alfredo 7.49
E5 Tour_of_Italy 14.99
D1 Tiramisu 2.99
D2 Zeppoli 2.49
D3 Dolcini 3.49
S1 Soda 1.99
S2 Bella_Limonata 0.99
S3 Berry_Acqua_Fresca 2.88
```

Sample activity file (comments are not part of the file):

```
T1 P2   // Party of 2 is assigned to Table 1
T2 P4
T4 P2
T1 O A1 A1 E1 E2   // Party at table 1 orders these items
T8 P10
T1 S // Food arrives at table 1
T3 P2
T1 C // T1 gets the check, pays and leaves & table is cleaned too.
T5 P2
T1 P1 // Party of 1 is assigned to table 1
...
```

Queries: You can display these questions to the user. Once the user selects a question, ask the user for additional input, if needed.

1. How many tables are occupied right now?
2. How many open orders right now?
3. How many orders have been processed so far today?
4. What is the average occupancy of particular table today?
5. What are the top 3 popular entries today?

Error checking:
- Do not allow orders from table with no party assigned to it.
- Do not allow assigning new party to a table when another party is already there.
- Do not allow assigning new party to a table for which waiter has not been assigned.
- Do not allow check-out from empty table.
- Do not allow delivery of food to an empty table!
- Do not assign any party to table with no waiter assignment.

- Reject any request that does not match with sample formats.

Since we are using lot of dynamic memory in this assignment, we will use a tool called valgrind to ensure that there are no memory leaks. If the executable file for Assignment 4 is hw4, we can run the following command. After hw4 ends, valgrind will display a summary.
valgrind hw4

Unlike Java, there is no bult-in String Tokenizer in C++. I have attached a zip archive that  Tokenizer functionality. This can be helpful to read and process the input lines from the files.

When you are ready to submit your source files, test your program using valgrind tool to find and fix any memory leaks. Then, issue the following commands in Linux prompt (no need to submit anything in elearning):

```
turnin  hw4 *.h *.cpp
```

If turnin command does not work, issue the following commands, then run turnin command again.

```
cp ~veerasam/.bashrc .
source .bashrc
Now, cd to appropriate directory
```

**Assignment #5:** Respin grocery checkout using template class

We are going to re-spin Project 3 using our own lookup table and handle bit different input format. If you want to look at a version of complted Project 3, look into ~veerasam/cs3376/proj3 in cs1 server.

1. Design and use a generic lookup table template class to speed up lookup operation. However, we can assume that lookup key is integer. We are mapping it to an user-specified class. Lookup key should support multiple ranges. You can assume a MAX limit of 10 ranges. Our usage will be specifically for PLU codes which means ranges are 0000 to 9999 and 90000-99999, and we will be mapping it to Product class pointer. Skeletion code & data file are available under ~veerasam/students/proj5 directory.
2. Take full advantage of Tokenizer code we introduced during Project 4 to read new format of input file inventory.csv. I have attached it for your reference. Test.cpp contains sample usage - do not include it in your project.
3. Output should be generated in a new file called output.csv in the same format. This operation need to be efficient. We can use brute-force approach and use loops to check the whole ranges. Optionally, if you want additional challenge, you are welcome to review http://www.cs.northwestern.edu/~riesbeck/programming/c++/stl-iterator-define.html and write your own iterator class for the lookup table template class. Then, you can iterate through the contents and generate output.

When you are ready to submit your source files, test your program using valgrind tool to find and fix any memory leaks. Then, issue the following commands in Linux prompt (no need to submit anything in elearning):

```
turnin  hw5 *.h *.cpp
```

**Assignment #6:** Respin restaurant using templates

Re-spin Assignment 4 using C++ template classes (primarily vector). Specifically, each array can be replaced by a vector. You can start with your assignment #4 solution and replace each array with a vector. All other requirements remain the same as Assignment #4.

**Assignment #7:** Semaphores: Producer Consumer problem

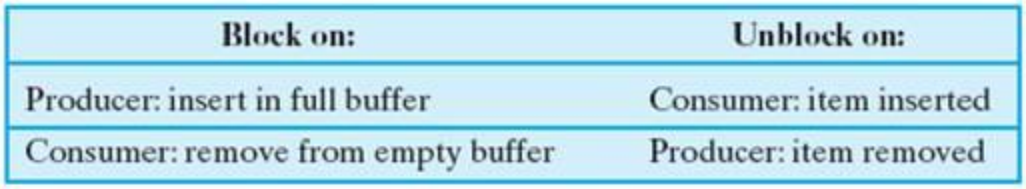
We are going to use 1 producer thread and 5 consume threads. They are going to run forever. There are two versions of the algorithm:

1. Without using semaphores (proj7v1.cpp)
2. With semaphores (proj7v2.cpp)

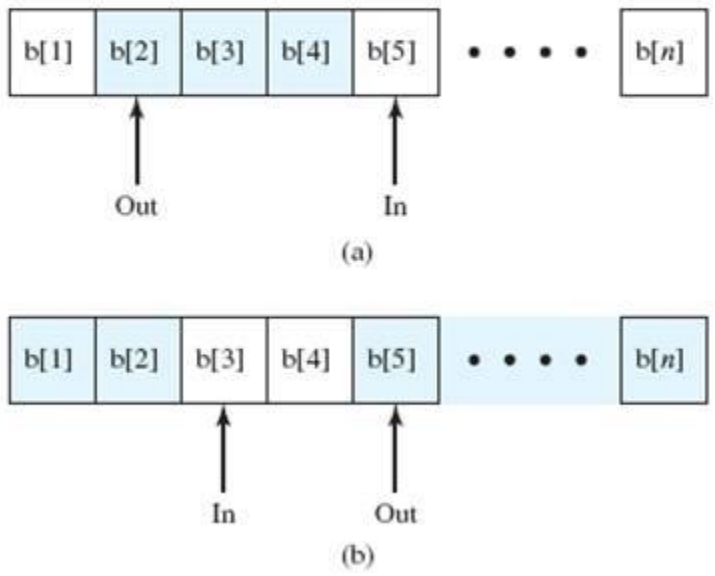
Submit both programs with sample outputs for both. Try to highlight the differences between both outputs. Feel free to insert sleep() in appropriate places to enable strange behavior. Try to keep those sleep() calls in both programs too.

Use the following diagrams and code to guide you through the project.

## Bounded Buffer



| Block on: | Unblock on: |
|---|---|
| Producer: insert in full buffer | Consumer: item inserted |
| Consumer: remove from empty buffer | Producer: item removed |



## Solution: No semaphores

| Producer | Consumer |
|---|---|
| while (true) { | while (true) { |
| /* produce item v */ | while (in == out) |
| while ((in + 1) % n == out) | /* do nothing */; |
| /* do nothing */; | w = take(); |
| append(); | /* consume item w */ |
| } | } |

## With semaphores

Semaphore n = 0, p = 1, c = 1, e = sizeofbuffer;

| Producer | Consumer |
|---|---|
| while (true) { | while (true) { |
| v = produce(); | semWait(n); |
| **semWait(e);** | semWait(c); |
| append(v); | w = take(); |
| semSignal(n); | semSignal(c); |
| } | **semSignal(e);** |
|  | consume(w); |
|  | } |

```
append():
    b[in] = v;
    in = (in + 1) % n;


take():
    w = b[out];
    out = (out + 1) % n;
```

Array b[] can be an array of 20 integers. produce() can be as simple as return the next integer in the sequence starting with integer value 1. consume(w) can simply print w out.

Once the assignment is completed, submit all the source files and 2 sample output files.

```
turnin hw7 proj7v1.cpp proj7v2.cpp buffer.h buffer.cpp output1.txt output2.txt
```

Programming assignments will be graded on a 100 point basis, utilizing the following criteria:

| | | **Max Score** |
|---|---|---|
| Source Code | Overall design | 40% |
| | Comments | 10% |
| | Indentation | 10% |
| | Readability | 10% |
| Execution | Nominal cases | 25% |
| | Special cases | 5% |
| **Total** | | **100%** |

**Policies and Procedures for Students**

The University of Texas at Dallas provides a number of policies and procedures designed to provide students with a safe and supportive learning environment. Brief summaries of the policies and procedures are provided for you at http://provost.utdallas.edu/home/index.php/syllabus-policies-and-procedures-text and include information about technical support, field trip policies, off-campus activities, student conduct and discipline, academic integrity, copyright infringement, email use, withdrawal from class, student grievance procedures, incomplete grades, access to Disability Services, and religious holy days. You may also seek further information at these websites:

- http://www.utdallas.edu/BusinessAffairs/Travel_Risk_Activities.htm
- http://www.utdallas.edu/judicialaffairs/UTDJudicialAffairs-HOPV.html
- http://www.utsystem.edu/ogc/intellectualproperty/copypol2.htm
- http://www.utdallas.edu/disability/documentation/index.html

***These descriptions and timelines are subject to change at the discretion of the Professor.***