Course Overview

Unix & C

15-123

Systems Skills in C and Unix

Effective Programming in C and
UNIX

All Semesters: 9 units

- This course is designed to provide a substantial exposure to the C programming language and the Unix programming environment for students with some prior programming experience but minimal exposure to C.
- Features of the C language that are emphasized
  - Arrays, strings and unions, dynamic memory allocation (malloc and free), pointers, pointer arithmetic, and casting.
- Data structures that are emphasized
  - Dynamic lists and hash tables.
- Algorithmic efficiency is emphasized
  - Space and time complexity.
- Students will develop a sense of proper programming style in the C idiom
  - be exposed to cross-platform portability issues.
  - learn to use tools such as emacs/vi, make, gdb to assist them in the design, testing and debugging programs. learn about regular expressions and will be able to use scripting languages such as Perl and Shell scripting
  - This course serves as the prerequisite for 15-213.

Prerequisites: 15-100

Course Components

- 8 programming labs – 40%
- skills labs – 7%
- Quizzes or Salons – 10%
- Written midterm – 10%
- C programming midterm – 7%
- Script programming midterm – 5%
- Final Exam – 20%
- TA points – 1%

Course Objectives

- At the end of this course
  - You should be able to write fairly sophisticated C programs
  - You should have a good understanding of program verification, debugging (tools and process)
  - You should have a good understanding of machine memory model and how programs work
  - You should be able to write useful scripts using languages such as perl and bash
  - You will have some understanding of how assembler s work
  - You should be prepared to go into 15-213

Course material

Primary Course Text Books:
All course textbooks are optional. Lecture notes are available from
(1) http://www.cs.cmu.edu/~guna/15-213s10/lectures
(2) C Programming Language (and Edition) by Brian W. Kernighan (Author), Dennis Ritchie (Author)

Other Recommended Text Books are:
(3) "C for Java Programmers" by Thomas M. Muldner ISBN: 0-201-70279-7 - Addison Wesley Longman 2000
(4) ANSI C on UNIX by Paul Wang http://www.sofpower.com/pub_bk01.html
(5) Learning Perl, Fourth Edition by Randal L. Schwartz, Tom Phoenix, brian d foy
(6) The UNIX Programming Environment by Kernighan and Pike

About the course
Course Staff

- Professor Guna (http://www.cs.cmu.edu/~guna)
- Gates 6005, office hrs – T, TR 10:30-12:00 or by appointment, or anytime my door is open

Course Assistants

- Section A
  - TBA
- Section E
  - Emily Grove
- Section F
  - Ke Young Lee
- Section G
  - Sylvia Han

How your time should be divided

- This is how you should spend your time on any week (9 units)
  - Attending lecture
    - 3 hours
  - Recitation
    - 1 hour
  - Homework and Coding
    - 5 hours
  - Disclaimer
    - It is hard to predict how long it will take you to finish your programming assignment
    - Talk to the course staff, if it is taking an unusually long time (20 hour /week)
    - We will be tracking this time as part of the assignment

Important

- Start assignments early – C programming can be very time consuming
- Assignments are individual, do not ask others to write code or copy others code w/o permission
- Sample code given in class can be used in any assignment
- Read notes and annotated notes
- Do homework
  - Not graded
- Attend lectures and recitations
  - DO NOT use laptops other than to take notes in class or write code
  - Any other activity is prohibited
  - Seek help early and often

Testing your prior knowledge

What is a function?

- A mathematical notion is that a function takes a given set of inputs and produces one and only one output
  - Hence for the same set of inputs it must always produce the same output
  - Functions can be used in programming to
    - Divide and conquer
    - Promote modularity
    - Unit testing
    - proof of correctness of the algorithm
  - Functions have overhead
    - Change in execution path
    - Runtime stack use

What is the purpose of the following function?

```c
int f(int n) {
    int i = 0, k = 0;
    while (k <= n) {
        k = i*i + 1;
        i++;
    }
    return i-1;
}
```

The purpose of the function is to find the floor of the square root of n. We also note that throughout the loop, the relationship

\[ k = i^2 + 1 \]

remains true

We assume that n >= 0

Also check to make sure if n is a perfect square, the code still produce the correct output.
A programming construct that allows one to repeat a task.

What are the types of loops you know? When do you use them?

- For loops (when number of iterations are predetermined) and while loops (otherwise).

Does a loop always end? Give an example where a loop does not end.

A loop does not always have to end. Here is one example:

```java
for (int i=0; i<10; i++)
...
```

Does a loop always execute once? Give an example where a loop may never execute.

No, if the initial condition is false, loop may never be entered.

---

What is a Loop?

- A loop is an iterative construct.
- It allows repeating a block of code repeatedly.

**for loop syntax (revisited)**

```java
for (initializations; exit condition; change)
{
    /* loop body */
}
```

Order of execution:
1. Initialize once.
2. Check the condition, if true enter the loop body.
3. Change the loop control variable.
4. Return to step 2.

**while loop syntax (new)**

```java
while (condition(s))
{
    /* loop body */
}
```

Order of execution:
1. Initialize once outside the loop.
2. Check the condition, if true enter the loop body.
3. Change the loop control variable.
4. Return to step 2.

---

When loops go wrong

```java
int pdt = 1;
for (int i=0; i<=32; i++)
pdt *= 2;
System.out.println(pdt);
```

The problem here is that loop runs 33 times. Each time, pdt gets multiplied by 2. So pdt is multiplied 33 times or have the value $2^{33}$.

But in a 31 bit integer world highest value is $2^{31} - 1$.

So this overflows and can display an incorrect value.

---

**Proving the Loop invariance**

```java
int fact(int n) {
    int i = n, k = n;
    while (k > 0) {
        k -= i * k;
        i+1;
    }
    return i;
}
```

Check the loop invariant:
- Is it true just before loop execution?
- Does it hold during the execution of the loop?
- Is it true just after the execution of the loop?
- What are pre and post conditions for this function?
What are Strings?
- String is an array of characters
- Characters come from ASCII (8-bit) or Unicode (16-bit) tables
- Memory is a big long String of bytes
- In Java
  - Strings are objects with their own attributes and operations (methods)
  - Strings are immutable
- Strings are very common in many applications
- In C Strings are not objects and is a byte array of characters ending with NULL character ‘\0’

What are boolean variables?
- Boolean variables only takes values TRUE or FALSE
- C does not have boolean as a type
- Use 0 for false and 1 for true
- Technically we can use a byte to store things
- The condition in an if statement is a boolean variable
- Boolean variables can be combined using
  - Logical AND (&&)
  - Logical OR (||)
  - Logical NOT (!)
- Properties
  - NOT (A and B) = NOT (A) or NOT(B)
  - NOT (A or B) = NOT (A) and NOT(B)
- Prove these identities

Logic Tables

<table>
<thead>
<tr>
<th></th>
<th>T</th>
<th>AND</th>
<th></th>
<th>T</th>
<th>OR</th>
<th></th>
<th>T</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A = True
B = False

<table>
<thead>
<tr>
<th></th>
<th>True</th>
<th>False</th>
</tr>
</thead>
<tbody>
<tr>
<td>T</td>
<td>False</td>
<td>True</td>
</tr>
<tr>
<td>F</td>
<td>True</td>
<td>False</td>
</tr>
</tbody>
</table>

Prove !(A && B) = !A || !B

Homework: prove
!(A || B) = !A && !B

Understanding UNIX
**Operating Systems**

- End user
  - Application programs
  - Utilities
  - Operating System
  - Computer Hardware

- Programmer

**Unix Operating System**

- Began at AT & T in 1970's
- Free source code for certain groups
  - Many versions of unix
- Linux version
  - Unix “like” system
  - Free and open source
  - Collaborative development
  - Small kernel

**Unix system shell**

- Read Command
  - SHELL
  - Execute command
  - Transform command

- Prompt

**Accessing unix**

- [http://www.cmu.edu/myandrew](http://www.cmu.edu/myandrew)
- Download and install SSH secure shell
- SSH
  - Provides access to unix.andrew.cmu.edu machines
  - Using a shell we can perform various tasks
    - `mkdir`, `cp`, `quota`, `mv`, ...
  - We develop and test our C and perl programs
  - We write shell scripts to make life easy

**What is C?**

- A general purpose programming language
- Developed in 1972 at AT & T for use with unix
- One of most popular programming languages
  - High level procedural programming
  - Direct Access to low memory
- C++ is the object oriented extension to C
  - Popular in industry
  - STL

**Why learn C?**

- Good
  - Flexibility
  - Efficiency
  - Low level access to memory
- Caution
  - Low level access to memory
    - Memory access violations (buffer overflows)
  - Hard to debug C code
    - Use a debugger such as gdb
  - Platform dependent
Life of a C program

```c
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("hello world\n");
    return 0;
}
```

How programs get executed

Program Development Process

- Editing
  - The process of creating the source code
- Compiling
  - The process of translating source code to object code
- Linking
  - The process of linking all libraries and other object codes to generate the executable code
- Executing
  - The process of running the program executable
- Testing/Debugging
  - The process of making sure program does what it is supposed to do
  - Consider all “edge” cases and make sure code does not break for some inputs

The C compiler – gcc

- GNU C compiler
  - Compiles, assembles, and produces executable code
- Also can compile
  - C++, Modula-3, FORTRAN, Objective-C, ...
- Examples
  - gcc hello.c → a.out
  - gcc -c hello.c → hello.o
  - gcc -S hello.c → hello.s
  - Using various flags
    - gcc -std=gnu99 hello.c
    - gcc -Wall -pedantic -ansi -O2 program.c

ANSI C

- Standard published by
  - American National Standards Institute for C language
- Some ANSI features
  - Do not mix data and code
  - Do not use functions that are not part of the standard libraries
Moving from Java to C
- From object oriented thinking to procedural thinking
- From classes and methods to functions/procedures
- From object oriented decomposition to procedural decomposition
- From a relatively "safe" high level language to fairly low level "unsafe" language
- From no direct access to memory (Java) to direct manipulation of memory.
- Automatic garbage collection to no garbage collection (clean up)

Code Examples

Data Representations

Data representations
- \text{int } x = 15;
  - Decimal representation of 15
- \text{int } x = 0x0F;
  - Hexadecimal (base-16) representation of 15
- \text{15 = 0000 ... 0000 1111}
  - Binary representation of 15
- Typically integers are 32-bits
  - Most significant bit is the sign bit (1-negative, 0-positive)
  - What is the largest signed integer that can be represented by 32-bits?
  - What is the largest unsigned int?
  - More about this in skills lab 1 and in lecture 02

Things to do before next class
- Take the background survey from Bb->course information
- Login to salon and complete the prior knowledge assignment
- After you complete, go back to assignment view mode and select up to 3 responses that you like from global questions
- Make your self familiar with course websites
  - Bb and \url{http://www.cs.cmu.edu/~gona/rc-rc-rc}
  - Go to recitation tomorrow

Next: more on Representation of data