Proposed Class

C Programming

Requirements

COSC 311

Why We Need This Class

Current computer science curriculum programming classes are based around high-level object orientated programming. Many students are not required to use a low-level non-object-oriented programming language such as C and are unprepared when the need presents itself. Our curriculum focuses on languages like Java and Python which do not require students to handle important concepts such as memory management, memory pooling, mixing assembly with other languages, build tools, and pointer arithmetic. A class is needed to help fill this void within our curriculum.

Class Purpose

The purpose of this class is to introduce students to C programming concepts, tools, libraries, and the importance of C in the realm of computers and computer science. Students will be able to read and develop C programs, understand how to use and implement libraries, have an understanding of program and memory structure, have the ability to implement common data structures and an understanding of pointer arithmetic. Three large projects will be used to teach C programming semantics and constructs in a non-object-oriented environment while targeting common C programming problems. Focus will be placed on efficient and reusable code, while fostering a deeper understanding of program memory and instruction flow.

Target Students

Students capable of implementing common data structures and algorithms in other languages but want to learn or get practice in C and low-level programming. Many students in the department ask me for advice and tutoring in C and C++; several have expressed interest in a class such as the one I am proposing. I wish to make this class available to undergraduate and graduate students.

What I Want To Get Out Of This Class

I want to continue to build my experience and knowledge at teaching others concepts and applications in computer science. I have been programming in C over the last seven years, and I want to pass on that knowledge to others so that they have a more rounded education and experience in real-world tools.

Grading

Grades are based on three categories

60%	C projects
20%	Lab Assignments
20%	Final

Textbook

An excellent textbook on C programing was selected for this class. The book covers most of the major topics that will be covered. The book can be accessed for free by students on the Books 24x7 database.

C Programming
Salim Y. Amdani
Laxmi Publications ©2009 (414 pages)
ISBN:9788131805466

Lab Assignments

Lab assignments are given at the start of each week and are due Friday at midnight via e-mail. Lab assignments are small projects based on weekly lecture topics. Lab assignments are worth 10pts each. Assignments lose 1pt each week they are late and must be turned in no later then the last day of class.

Projects

Four large projects will be assigned during the semester. These projects can be worked on alone or in pairs. These projects combine the skills taught in class and practiced in lab assignments. Projects will be due two weeks after they are given. Late projects will lose 5pts each week they are late and must be turned in no later then the last day of class.

Groups

Students can work in groups of two or three on larger projects, but each member of a group must turn in a minimum one page write up of their contribution to the project.

Graduate Students

Graduate students are expected to complete all optional problems.

Class Topics Plan

List of topics and assignments planned for each week.

Week 1

Readings

Chapter 1

Lecture Topics

- History of C
- Why is C important
- GCC
- Makefiles
- CMake

Assignment

- Hello World - Makefile
- Good-Bye World - CMake

Week 2

Readings

Chapter 3, 4, 5

Lecture Topics

- Data Types
- Constructs
- Functions
- Storage Class
- Intro stdlib

Assignment

- print sizeof C types
- Conditional Constructs
- Looping
- Basic functions
- Pass by value
- Recursion

Week 3

Readings

Chapter 6

Lecture Topics

- Macros
- Header Files
- Problems with data types
- assert, enforce, and C errors
- static arrays
- stdlib

Assignment

- Write stdtype.h (U32, I32, ect...)
- Write error.h (assert, enforce, ...)
- Arrays(reading, writing, sorting)

Week 4

Readings

Chapter 7

Lecture Topics

- Intro to pointers
- Passing Pointers
- malloc, calloc, realloc, free
- Dynamic arrays

Assignment

- Functions that take pointers
- Dynamic Memory (primitives, arrays)
- Using malloc, calloc, realloc, and free

Week 5

Lecture Topics

- Intro to Strings
- String Conversions
- Pointer Arithmetic

Assignment

- Reading and writing strings
- Strings (Searching, sorting, case issues) string.h
- Strings(Searching, sorting) pointer arithmetic

Week 6

Lecture Topics

- Libraries
- Unit testing

Assignment

- Start String Library Project

Week 7

Lecture Topics

- Memory Leaks
- Dangling Pointers

Assignment

- String Library Project Due

Week 8

Readings

Chapter 8

Lecture Topics

- Structures
- Unions
- Anonymous Structures and Unions
- Stacks
- Linked Lists

Assignment

- Basic Struct and unions usage
- Convert infix to postfix

Week 9

Lecture Topics

- Timing code
- Memory Management
- Memory Pools

Assignment

- Start Memory Pool Library Project

Week 10

Lecture Topics

- Using Assembly with C

Assignment

- Memory Pool Library Project Due

Week 11

Readings

Chapter 9

Lecture Topics

- File I/O - Text Files
- String Parsing with C

Assignment

- Start INI Reader Library Project

Week 12

Lecture Topics

- File I/O - Binary Files
- Function Pointers

Assignment

- INI Reader Project Due

Week 13

Lecture Topics

- Command line arguments
- Scanning Directories and Files
- File Stats

Assignment

- Start File Packer Project

Week 14

Lecture Topics

- GUI - FLTK

Assignment

- File Packer Project Due

Week 15

Lecture Topics

- Review for final

Assignment

- All Projects and Assignments are due

Strings Library Project

Implement the following functions to work like the C standard library functions Each function should have its own header file and C source file. The header file should be well documented as to the functions purpose, input parameters, and output values. Each function should have a unit test. You need to use CMake to make a library and a unit test binary that runs each functions unit test and reports each unit tests result.

Functions To Implement

```c
void * _memset( void *, I32, size_t );
void * _memcpy( void *, void *, size_t );
size_t _strlen( char * );
char * _strcat( char *, char * );
char * _strncat( char *, char * );
char * _strcpy( char *, char * );
char * _strncpy( char *, char * );
char * _strdup( char * );
char * _strndup( char * );
I32 _strcmp( char *, char * );
I32 _stricmp( char *, char * );
char * _substr( char *, size_t );
char * _trim( char * );
char * _strchr( char *, char );
char * _strrchr( char *, char );
I32 _atoi( char * );
char * _itoa( char *, I32 );
char * _strtok( char *, char * );
char * _strhex( char *dest, const I32 val ); // write val as a hex string in dest
char * _strbin( char *dest, const I32 val ); // write val as a binary string in dest
```

Optional Functions

```c
F32 _atof( char * );
char * _ftoa( char *, F32 );
char * _strstr( char *, char * );
char * _strfry( char * );
void * _memmove( void *, void *, size_t );
```

Memory Pool Library Project

Develop a library to create a stack based memory pool. The memory pool will be used in a time critical application that requires dynamic chunks of memory to be created and maintained during the algorithm. There will be one structure, the memory pool, and four functions.

API

```
struct MemoryPool // holds info about a memory pool

/*
     Creates a memory pool of size pool_size.  If pool is NULL,
     this function will return a pointer to a New MemoryPool,
     else returns pool;

     Example Usage:
     // create a 10K pool
     MemoryPool *pool = PoolCreate( NULL, 10240 );

*/
MemoryPool * PoolCreate( MemoryPool *pool, size_t pool_size );

/*
     Grab memory from a pool.  Returns NULL if pool is out of
     memory.

     Example Usage:
     // get 1024 bytes from the pool
     char *str = (char *)PoolMalloc( pool, 1024 );

*/
void * PoolMalloc( MemoryPool *pool, size_t block_size );

/*
    Releases all of the memory in the pool to be reused.
    Watch out for dangeling pointers!
*/
void PoolRelease( MemoryPool *pool );

/*
    Frees the memory pool back to the operating system.
    Pool is nolonger usable.

*/
void PoolFree( MemoryPool *pool );
```

Optional Functions

Implement the two-sided memory pool or the long/short term two sided memory pool.

For two-side memory pool:

```
// Replace PoolRelease with PoolSwap
void PoolSwap( MemoryPool * pool );
```

For long/short term two-side memory pool:

```
// Add parameter to PoolMalloc
void * PoolMalloc( MemoryPool * pool, size_t block_size, int long_term );

// Add parameter to PoolRelease
void Poolrelease( MemoryPool *pool, int long_term );
```

INI Reader Project

Develop a library that reads in a standard INI configuration file into memory, implement the give API functions to query the INI file, add new entries, and update entries. Each function should have its own header file and C source file. Each header file should be well documented as to its purpose, input parameters, and output values. You need to use CMake to produce a library and a unit test program that tests each function in the API.

API

```
struct Config // memeory struct to load the INI file into
I32 ConfigOpen( Config *, char * ); // read a file into memory
I32 ConfigClear( Config * );  // clear an config structs memory, no memory leaks!
char * ConfigGetString( Config *, char *, char * ); // get a string value from the config settings
I32 ConfigGetInt( Config *, char *, I32 ); // get an int value from the config settings
F32 ConfigGetFloat( Config *, char *, F32 ); // get a float value from the config settings
char * ConfigSetString( Config *, char *, char * ); // set or add a string
I32 ConfigSetInt( Config *, char *, I32 ); // set or add an int
F32 ConfigSetFloat( Config *, char *, F32 ); // set or add a float
```

Optional Functions

```
I32 ConfigSave( Config *, char * ); // save a config struct to a INI file
```

File Packer Project

Develop a file and directory packing utility. This program will pack one or more files and/or directories into a single file or unpack a file rebuilding the packed file structure. This program is controlled via command line arguments.

The program takes the following arguments.

```
pack -p [DIRECTORY TO PACK] [NAME OF PACK FILE]
pack -u [PACK FILE]
pack -h
```

Optional Arguments

```
pack -l [PACK FILE]
pack -a [PACK FILE] [FILE/DIR TO ADD TO PACK]
pack -d [PACK FILE] [FILE/DIR TO DELETE FROM PACK]
```