Portland State University

Maseeh College of Engineering and Computer Science

Proficiency Examination Process

2016-2017

PSU Expectations of Student Competencies

Students that apply to PSU's Computer Science department for entry into the upper division program must pass a proficiency examination. This examination has students demonstrate their abilities to program, implement data structure algorithms and use recursion. Applicants will program independently while being observed by PSU Faculty, without any assistance. PSU requires that applicants program in C++ using GNU's C++ compiler using C++ standard 98 on the linux command line, without the assistance of IDEs or other auxiliary (or online) materials. PSU expects from the current coursework taken that students should be able to proficiently implement recursive solutions to data structure problems.

The Process

The process of taking the proficiency exam starts with an orientation that includes a linux tutorial and practice session. During the practice session, applicants will experiment with the environment used for the proficiency exam and work through sample problems.

The actual proficiency exam will take place for all applicants per a schedule provided by the CS Department. Students will receive a randomly assigned question at that time and be required to demonstrate programming in C++ implementing a recursive data structures' algorithm with no additional external support.

Students who receive an "In Progress" score will be invited to retake the proficiency demo. **Students must receive a passing score for their applications to be further processed by PSU.**

Compiler and Editor Requirements

During the proficiency exam, students must use the GNU GCC C++ compiler (g++) in the default –ansi mode. This mode meets the **C++ standard 98** guidelines (which can be obtained through these flags: –std=c++98, -std=gnu++98, or -ansi).

Students should compile with the –Wall flag to enable warnings about constructions that should be avoided. Students may compile with the –g flag to support the use of gdb. Use of any other flags is not allowed during the exam. All programming will be done using one of the following approved editors: **vi, vim, emacs, nano or pico**. The use of IDEs is not allowed.

Data Structure Expectations

It is expected that students applying for upper division entry should be proficient in programming **recursive** data structures algorithms. Students will be expected to demonstrate programming proficiency with linear linked lists, circular linked lists, doubly linked lists, arrays of linked lists, and binary search trees. For the proficiency demonstrations, each of these data structures has already been created and the applicants are asked to implement an algorithm to traverse and/or modify the data structure recursive.

Expected Core Competencies

PSU expects that students applying for entrance into the upper division be able to write complete programs in C++, using multiple files (.cpp and .h files) implementing pointer based data structure solutions, without the use of external libraries to assist with the data structures. Students should be competent using the linux environment without the assistance of an IDE, the internet, or external library support (beyond iostream, string and/or cstring libraries). Students should be fluent with the use of arguments (both pass by value and pass by reference) without using global variables.

When examining fluency, students should be able to develop code that is free from segmentation faults, memory leaks, and properly use **recursion** in the solution. The use of debuggers, such as gdb, is highly encouraged. **Correct syntax is expected**.

PSU expects students to show fluency with **C++ pointer based data structure solutions**. This includes recursive solutions for linear linked lists, circular linked lists, doubly linked lists, and arrays of linked lists and binary search trees; students are expected by this time to be fluent with linux editor functionality (search, replace, navigation) and gdb, implementing recursive solutions for problems similar to these (for example):

- Linear Linked list: Determine if there are duplicate data within an existing linked lists.
- Circular Linked list: Remove every occurrence of a particular item
- Doubly Linked list: Swap every other node
- Array of linked lists: Remove the last item in each linear linked list
- Binary Search Tree: Return the largest item in a binary search tree

The only libraries available during the proficiency exam are:

- iostream
- fstream
- cctype
- cstring

Background Competencies

Once applicants have taken CS162, PSU expects proficiency with the fundamental syntax of C++. Students should be proficient programming using pointers, dynamic memory, and linear linked lists. PSU expects that students passing CS162 are able to show fluency with **pointer based linear linked list** solutions. Students passing CS162 demonstrate fluency at problems such as (for example):

- Adding a node at the end of a linear linked list
- Counting the number of nodes in a linear linked list
- Finding how many times a matching item appears in a linear linked list
- Removing an item from a linear linked list

Once applicants have taken CS163 (Data Structures), PSU expects proficiency applying recursion to data structure algorithms. This includes being fluent using functions, including the returned value of the function and returning values when appropriate. PSU expects students passing CS163 are able to show fluency with **recursive pointer based data structure** solutions. Students passing CS163 are able to show fluency using recursive solutions for linked data structures (linear, circular and doubly linked), and pointer based binary search trees. Students should be fluent at problems such as (for example):

- Remove every leaf from a binary search tree
- Return the largest item in a binary search tree
- Calculate the height of a binary search tree
- Finding how many times a matching item appears in a binary search tree

Once applicants have taken the equivalent of CS202, PSU expects fluency programming in C++ using classes, pointers and dynamic memory (new and delete), functions (using returned values, passing by value and passing by reference), and recursive solutions to pointer based linked data structures. Students should be able to demonstrate proficiency at applying recursive solutions to insert, traverse or remove from pointer based linked data structures of: linear linked lists, circular linked lists, doubly linked lists, arrays of linked lists, and binary search trees.

Applicants should have experience implementing OOP designs using concepts such as multiple classes, inheritance, dynamic binding, copy constructors, initialization lists, and operator overloading, by working at the command line and without an IDE. Applicants should have hands on implementation experience combining of data structures such as lists of trees, trees of trees, arrays of lists, balanced tree and graph implementations.

Applicants should also have been trained on the process of learning new programming languages that may be expected by the Professors of the upper division courses. Applicants should be prepared to demonstrate how to apply the concepts of inheritance, dynamic binding, and advanced data structures to other languages such as Java.

Proficiency Exams – General Information:

- Students will be demonstrating proficiency in programming in linux and data structures
    i. Every student must show proficiency in a high level programming language such as C++
    ii. The examination will have student implement various recursive data structure algorithms, using pointer based implementations.
    iii. All of the coding will be done using linux, with a linux editor (using either vi, vim, or emacs; pico/nano are allowed but not recommended)

- Every student will also be asked to demonstrate proficiency in using gdb and features of the linux editor that they are using
    i. Students will demonstrate navigation
    ii. Students will demonstrate search and replace
    iii. Students will demonstrate setting breakpoints in gdb

- The data structures algorithms implemented may include coding recursive traversal and/or manipulation algorithms for any of the following pointer based data structures:
    i. Linear linked lists
    ii. Doubly linked lists
    iii. Circular linked lists
    iv. Binary search trees
    v. Arrays of linear linked lists

- The data in the data structures may be an integer or a dynamically allocated array of characters.

- Students should be comfortable working within the class construct without the use of an IDE, working with .h file(s) and multiple .cpp files.

- Students are encouraged to practice prior to the proficiency exam.

Proficiency Exams – The Process:

1. Show Picture ID upon arrival proficiency examination
2. The proficiency examination is "closed book, closed notes"
    a. No electronic devices such as phones, smart watches may be used
    b. Online access to materials is prohibited
    c. Online access is only granted to login to an approved linux system using a terminal emulation program
3. A test question will be randomly assigned
4. Wait to be assigned a seat in the testing area
5. Once seated in the testing area,
    a. Students will be scored within 1 hour after starting the test; the student will be asked to submit their work when scored by the proctor
    b. Students are instructed to follow the instructions and wait for the proctor
    c. Students may use vi, vim, or emacs. In some situations pico or nano will be accepted
    d. Students may not use help, man, or any other tool to gain online assistance
    e. **We recommend** taking the time to **design** an approach before coding. Students are being scored on their comprehensive approach to problem solving and their ability to implement pointer based data structure algorithms.
    f. We recommend **commenting out code** rather than deleting it
    g. Students may write additional functions, but the prototype of the function given must be unchanged and called directly from main.
    h. We suggest writing more than one function if necessary. If more than a single function is written, main is still required to call the function assigned in the problem. However, students should minimize extra unnecessary traversals of the data structure.
    i. Students are encouraged to **compile and run the code multiple times.**
    j. Students are allowed to debug their programs using **gdb**.

Proficiency Exam - Rules:

- All problems will require a **recursive** solution
- **NO LOOPS may be used in the solution,** unless indicated in the problem statement
- Each problem will be given a specific prototype
- The prototype supplied may not be changed
- **Global variables** are not allowed
- The prototype supplied represents the function that should be called by main
- For questions that have students count, average, or traverse in some way, the **solution should NOT modify the data structure**
- For questions that ask students to insert or copy, the solution should **NOT delete items**
- **As changes are made to the code,** please comment out code rather than deleting it!
- Use of the cstring, cctype, and iostream libraries are allowed. No other libraries may be included or used.

Proficiency Exam – Sample Practice Questions:

A complete environment will be provided by PSU that already creates and populates the data structure with data. The data may be integer data or dynamically allocated arrays of characters. Students will be implementing specific functions to perform a task which will then be linked with code to build, display, and destroy the data structure assigned.

Linear linked lists, circular and doubly linked lists:

1. Write a recursive function in C++ to remove all nodes except for the first and last nodes.
2. Write a recursive function in C++ to remove the last two nodes
3. Write a recursive function in C++ to add a node to the end, if it is unique
4. Write a recursive function in C++ to move the last node to the beginning
5. Write a recursive function in C++ to copy the data structure

Binary search trees:

1. Write a recursive function in C++ to make a copy of a binary search tree
2. Write a recursive function in C++ to make a copy of a binary search tree and place it in a linear linked list, sorted
3. Write a recursive function in C++ to take a sorted array and insert it into a binary search tree balanced
5. Write a recursive function in C++ to add a node into a binary search tree
6. Write a recursive function in C++ to remove the largest item in a binary search tree
8. Write a recursive function in C++ to find the root's in-order successor