Spring 2019 Programming Languages Qualifying Exam

CODE___________________________________

This is a closed book test.

Correct, clear and precise answers receive full marks

Please start a new page for each question.

There are five (5) questions, each 20 points

1) (20pts) Object Oriented languages

a) Define/describe Object Oriented paradigm.

The object oriented paradigm is where the main entity in programming is an object. The objects "contain" data and "methods" which allow external routines to access and mutate the data. The object is in control of the underlying data structures and external objects can only interact with the object with the method (API). Objects usually are associated with a CLASS. Object instances all use the same methods. Classes can inherit features from other classes and some languages allow for inheritance from multiple classes, which can cause a diamond problem

b) Java has non-object primitives like int and char, while Python treats everything as an Object. Identify the strengths and weaknesses of both approaches and propose an explanation on why these languages handle primitives differently.

Python uses everything is an object to make accessing anything in Python the same. Creating a STACK class does not disallow the use of primitives, for example. Java does distinguish between primitives and object. One advantage is better performance when dealing with simple data structures, especially arrays of primitites. Java does not need to do dynamic type checking and indirect referencing via the heap.

2) (20pts) Memory management

Consider the following C program. Identify the primary memory segments, provide their purpose and discuss how memory is maintained in each segment. Show the specific memory settings when function **f()** is reaches the base case. What is the output of the print statement?

```
char *s;

void f(int i)
{  static int k=0;
    k = k + I;
    if ( i <= 0) printf("k is %d string is %s\n", k, s);
   else f( i – 1);
}

int main()
{  char *s;
    s = malloc(20);
    strdup(s,"hello");
    f(2);
}
```

STACK, HEAP, DATA, CODE segments

The Code segment holds the object code of the program. It can also hold all string literals (like "hello"
The data segment holds the char *s, and the static int k from f()
The heap is used to allocate 20 bytes for s to point to. It holds the string "hello", a copy from hello out of the code segment.
Runtime stack should have main → f(2), f(1), f(0). Print "k is 3, string is hello"

3) (15pts) What does BNF stand for? In what context is it used? What is BNF similar to as it relates to Automata?

Backus Nauer Form. BNF is used to describe the syntax of a language using a recursive meta-language. It is similar to Context Free Grammars.

4) (15pts) Given the symbols {a,b}, Create a *Deterministic Finite Automata* which accepts all strings that contain no more than two consecutive **b**'s. For example **abbabbaba** is acceptable, where as **aabbababaabbba** is not (because the string has 3 or more consecutive b's).

Requires reader to ensure the FA is deterministic and accepts properly

5) (15pts) Compare and contrast dynamic and static type checking.

Type checking ensures reliability of a language. For example, assigning a string to an integer variable can be problematic. Compiled languages usually perform most of their type checking during compilation – this is called static type checking. Dynamic type checking is done during run time. Many interpreted language implement dynamic type checking. This means that for each operation, additional instructions must be made to ensure that operation is proper for the operands. Dynamic type checking can occur for compiled languages.

6) (15pts) Write the function *nth* in a functional language (LISP preferred), which takes a positive integer *N*, and a list *L*, and returns the element at the *N*th position in the list. You may assume that 1 <= *N* <= length of the list. In other words, don't worry about an out-of-bounds index. Your function must be recursive. For example: *(nth 2 '(a b c)) == 'b*

```
(define nth (N L)
  (if (eq? N 1) (car L) (nth (- N 1) (cdr L)))
```