Project Pre-Proposal

1) Who are the members of your team?
Michael Tang, Ross Blassingame

2) What basic problem will your project try to solve?
We wish to extend the Python language with our implementation of Python pointers, "pyPointers." With pyPointers, we wish implement and experiment with certain data structures such as linked-lists, binary trees, stacks and queues. We also want to utilize pyPointers to explicitly give users control of passing variables to functions of whether these variables should follow value or reference semantics.

3) Define the problem that you will solve as concretely as possible. Provide a scope of expected and potential results. Give a few example programs that exhibit the problem that you are trying to solve.

One application would be to be able to write a python program with explicit syntax that determines whether a certain variable should have value or reference semantics. Our scope is to extend our compiler to handle reference semantics with primitive variables, and extend lists and dictionaries with value semantics though explicit syntax which we call pyPointers.

```
def addOne(l):
    l[0] = l[0]+1
    print l
myList = [1,2,3]
addOne(myList)
print myList

Expected Output:
[2,2,3]
[1,2,3]
```

Another application would be to be able to implement different data structures that require pointers, such as linked lists, binary trees and queues. To the right is an example program that showcases the potential functionality regarding linked lists.

```
Class Node:
    def __init__(self, _value):
        self.value = _value
        self.next = None

n1 = Node(1)
n2 = Node(2)
n1.next = n2
print n1.value, n1.next.value

Expected Output:
1 2
```

To the right is an example program regarding the potential functionality regarding binary trees.

```
Class Node:
    def __init__(self, _value):
        self.value = _value
        self.left = None
        self.right = None

root = Node(5)
l = Node(4)
r = Node(6)
root.left = l
root.right = r
print root.value, root.left.value, root.fight.value

Expected Output:
5 4 6
```

4) What is the general approach that you intend to use to solve the problem?
Implementing pyPointers in our compiler is both feasible and realistic to attain in the timeframe specified. We will have to add new, custom AST classes that correspond with the syntax we choose to use to represent pointers, and have recursion take care of adding them into the AST. Once we successfully get the new AST classes regarding pointers working, the rest should, in theory, be a relatively straight-forward process. As shown in the python program in the box to the right, variables don't point to the location in memory – they simply copy the value at the time of the call.

```
>>> a = 1
>>> b = a
>>> a = 2
>>> b
1
```

We will use the AST classes to generate x86 assembly code to have variables point to the location in memory rather than simply copy the value of the variable. Here is how the same program run will turn out once pyPointers in implemented:

```
>>> a = 1
>>> *b = a
>>> a = 2
>>> b
2
```

5) Why do you think that approach will solve the problem? What resources (papers, book chapters, etc.) do you plan to base your solution on? Is there one in particular that you plan to follow? What about your solution will be similar? What will be different?
This will solve the problem because x86 allows us to set and access memory locations, so by reading the pyPointer syntax from our input programs, which will be converted into a custom AST, we can write the relevant x86 instructions that allows us to accomplish the tasks we listed above.

6) How do you plan to demonstrate your idea? Will you use your course compiler. If so, what specific changes do you plan to make to it (e.g., what passes need to be changed or added)?

Yes, we will use the course compiler we've built over the course of the semester. We will extend it to be able to use pointers by accessing memory locations so we can use new data structures, like linked lists, queues, and binary trees. We will need to add new AST classes that read our new pyPointer syntax. Here is an ordered list of what passes we will need to update:

1. Lex and parse
2. Uniquify
3. Explicate
4. Heapify
5. Closure convention
6. Flatten
7. Select instructions
8. Register allocation

We certainly will have our work cut out for us, and will need to be efficient in the ways we use our time if we want to have pyPointers completed by the project's due date.

7) How will you evaluate your idea? What will be the measurement for success?

If we can get the linked lists, binary trees, queues, and memory references working, we will count that as a successful project. We can use another language that makes use of memory and pointers to judge the effectiveness of our project. For example, we can make a program in C++ using pointers to implement a linked list, make the equivalent program in python, compile it using our compiler, and if they have the same functionality we consider that a success.