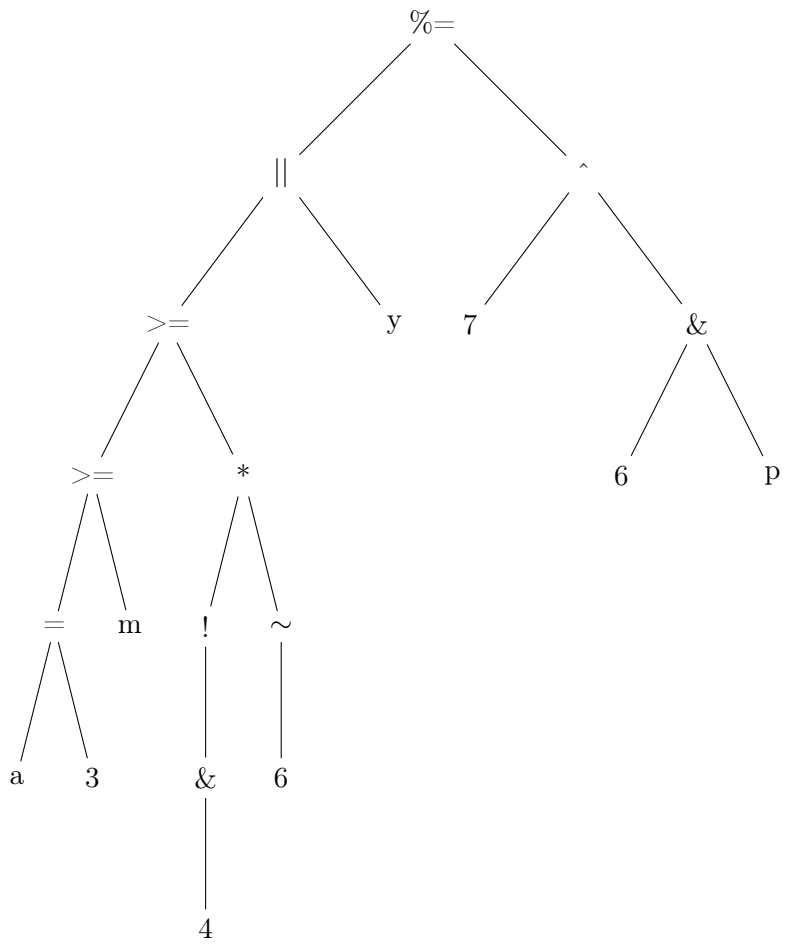
Homework 4

Andrew Kowalczyk

October 31, 2013

1 Abstract Syntax Tree

(a = 3) >= m >= ! & 4 * ~6 || y %= 7 ^ 6 & p



2 Javascript semicolon questions

2.a

When trying to call the function f(), it returns undefined. Javascript inserts semicolons into the program as so:

```
function f() {
    return;
    {x: 5};
}
```

Python would remedy this because one would receive an indentation error trying to run the Python equivalent of code.

2.b

When trying to evaluate this expression, we receive this error TypeError: Property 'b' of object #<Object> is not a function. Javascript inserts semicolons into the program and interprets it as so:

```
var b = 8;
var a = b + b(4 + 5).toString(16);
```

Python would remedy this because the lexical definition of statements says that logical lines end in with a newline.

2.c

After trying to evaluate the program, we receive this error, TypeError: 'undefined' is not an object (evaluating '"mundo" ["Hola", "Ciao"].forEach'). This is because it is attempting to use the ["Hola", "Ciao"] as indexer operator to "mundo". Again, Python would remedy this because the lexical definition of statements say that logical lines end in with a newline.

2.d

When trying to evaluate this, we are alerted with "Goodbye" then "Hello". This is because sayHello is called with the anonymous function below due to being in a closure. Javascript interprets the code as this:

```
var sayHello = function () {
    console.log("Hello")
}

(function() {
    console.log("Goodbye")
}(sayHello))
```

3 What if local variables were allocated in static storage?

Consider this code:

```c
#include <stdio.h>

int f(int x) {
    int a = 0;
    if (x == 0) {
        a = 1;
    } else {
        return f(x - x) + a;
    }
    return 0;
}

int main() {
    printf("%d\n", f(5));
    return 0;
}
```

The program prints out 0. If local variables were allowed in static storage, the program would print out 1.

4 Scoping

4.a Static Scoping

It would print out 1122 since the second setX() call changes the global variable to 2.

4.b Dynamic Scoping

It would print out 1121 since the second setX() call only changes the local variable x.

5 Binding within dynamic scoping

Consider this python code:

```python
a = 4
b = 0
c = 7
d = 7

def f(n):
    global a
    a = n
    return a

```

```
print (a - f(b) - c * d)
```

If Python were to evaluate the the function first which would assign the global a to 0, then the expression could yield different results. If the lookup for a were to happen first, then a would stay as 4.

6 Explain the C declarations

6.a double *a[n];

This is an array of n pointers to doubles.

6.b double (*b)[n];

This is a pointer to an array of n doubles.

6.c double (*c[n])();

This is an array of n pointers pointing to functions returning a double.

6.d double (*d())[n];

This is a function returning a pointer to an array of n doubles.

7 Rewrite problem 6 in Go

7.a var a [n]*float64

7.b var b (*)[n]float64

7.c var c [n] *func()float64

7.d var d func() (*[n]float64)

8 Convert from infix (-b + sqrt(4 × a × c)) / (2 × a)

8.a Postfix

0 b - c 4 a ×× sqrt + 2 a × /

8.b Prefix

/ × 2 a + - 0 b sqrt ×× 4 a c

8.c Do you need a special symbol for unary negation? Why or why not?

There are two options. If you make parentheses required, you must assign another symbol for unary negation like ~. If you don't make them required, you can simply subtract from 0.

9 Interleave using C-Style arrays

Code available at `http://codepad.org/Fw4xASrh` and on my github.

```
#include <iostream>
using namespace std;

void interleave(int a[], int lena, int b[], int lenb, int c[]) {
    int aindex = 0;
    int bindex = 0;

    for (int i = 0; i < (lena + lenb); ) {
        if (aindex < lena) {
            c[i] = a[aindex];
            aindex++;
            i++;
        }

        if (bindex < lenb) {
            c[i] = b[bindex];
            bindex++;
            i++;
        }
    }
}

int main () {

    int a[50], b[50], c[100], lena, lenb, lenc, i;

    cout << "Input number of elements in first array" << endl;
    scanf("%d", &lena);

    cout << "Input " << lena << " integers" << endl;
    for (i = 0; i < lena; i++) {
        scanf("%d", &a[i]);
    }

    cout << "Input number of elements in second array" << endl;
    scanf("%d", &lenb);

    cout << "Input " << lenb << " integers" << endl;
    for (i = 0; i < lenb; i++) {
```

```
        scanf("%d", &b[i]);
    }

    interleave(a, lena, b, lenb, c);
    lenc = lena + lenb;

    for (int i = 0; i < lenc; i++) {
        cout << c[i] << " ";
    }
    cout << endl;

    return 0;
}
```

10 Interleave using C++ vectors

Code available at `http://codepad.org/P6KKMBqY` and on my github.

```
#include <iostream>
#include <vector>
using namespace std;

template <typename T>
vector <T> zip(const vector <T> & a, const vector <T> & b, size_t len) {
    vector <T> result;
    int aindex = 0;
    int bindex = 0;

    for (size_t i = 0; i < len; ) {
        if (aindex < a.size()) {
            result.push_back(a[aindex]);
            aindex++;
            i++;
        }

        if (bindex < b.size()) {
            result.push_back(b[bindex]);
            bindex++;
            i++;
        }
    }
    return result;
}

int main() {
    vector <int> a = {};
    vector <int> b = {};
    size_t lenc;
    int i, n, m, lena, lenb;
```

```
32
33      cout << "Input number of elements in first array" << endl;
34      scanf("%d", &lena);
35
36      cout << "Input " << lena << " integers" << endl;
37      for (i = 0; i < lena; i++) {
38          scanf("%d", &n);
39          a.push_back(n);
40      }
41
42      cout << "Input number of elements in second array" << endl;
43      scanf("%d", &lenb);
44
45      cout << "Input " << lenb << " integers" << endl;
46      for (i = 0; i < lenb; i++) {
47          scanf("%d", &m);
48          b.push_back(m);
49      }
50
51      len = (a.size() + b.size());
52      vector <int> c = zip(a, b, lenc);
53      for(size_t i = 0; i < c.size(); i++) {
54          cout << c[i] << " ";
55      }
56      cout << endl;
57  }
```