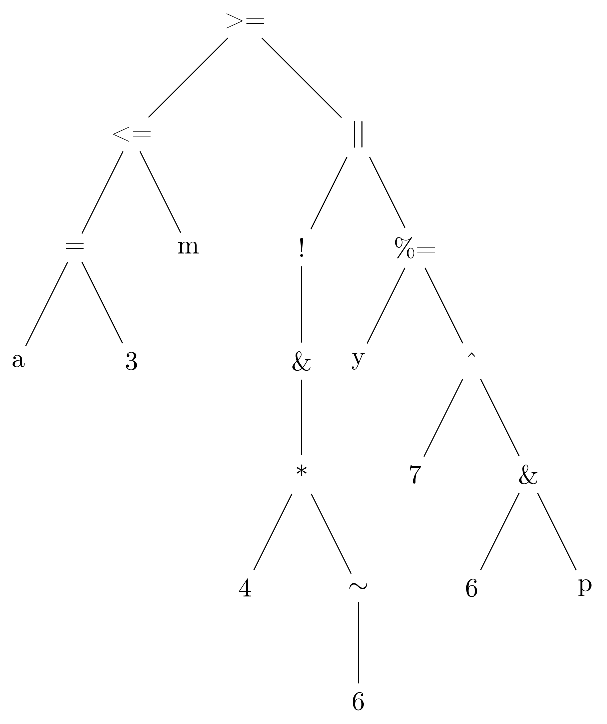
CMSI 386 Homework #4

Zane Kansil & Edward Bramanti

November 12, 2013

1. Abstract syntax tree for expression:

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



2. Non-intuitive Javascript semicolon ambiguities

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

The case is ambiguous because Javascript will return whatever follows the return statement. JS can either return the result of an empty statement (undefined) or it can assume the users intent and return the otherwise dangling object {x: 5}. In this case JS returns undefined.

In Python the closest translation would be:

```
def f():
    return
        {"x": 5}
```

The ambiguity is solved because Python throws an unexpected indent error on line 3.

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

In this case the programmer has omitted the semicolons to delimit the initialization of b and a. This causes JS to try to call `b(4 + 5).toString(16)`, or `b("9")`. This is of course an error because numbers are primitive; they cannot be called. In Python the closest translation would be:

```
b = 8
a = b + b
('0x9')
```

Python has no issues in this case because assignment (and other statements) are delimited by the newline character; no function call will be made. Simulating the exact scenario of the JS throws a similar `int` object not callable error.

```
b = 8
a = b + b('0x9')    //throws 'int' not callable
```

(c)
```
var place = "mundo"
["Hola", "Ciao"].forEach(function (command) {
  alert(command + ", " + place)
})
```

This case is ambiguous because `"mundo"` is being indexed with the array `["Hola","Ciao"]`. The array of `["Hola","Ciao"]` can not be casted to an integer. Indexing a string with a non-integer, or an integer that is out of bounds, results in the string being evaluated to `undefined`. Therefore, the case is calling `forEach()` on `undefined`. Python solves this issue because statements end with `NEWLINE`.

(d)
```
var sayHello = function () {
  alert("Hello")
}
```

```
(function() {
  alert("Goodbye")
}())
```

This case is ambiguous because it seems as though `sayHello()` is a declaration, and the second function looks like it is being called. But, since there is no semicolon to break the first function definition, the parentheses around the second function act as a function call. It does not pass a value; instead, it calls a `void` function, which prints `"Goodbye"`. The script will print `"Goodbye"` first, and then `"Hello"`. Python does not run into this problem, because a function call has to be on the same line. The `NEWLINE` character will prevent anything that looks like a potential function call written on separate lines.

3. Give an example of a program in C that would not work correctly if local variables were allocated in static storage as opposed to the stack. For the purposes of this question, local variables do not include parameters. *Local variables init in static storage causes malfunctions*

```c
#include <stdio.h>

int f(int x) {
    /*static*/ int a = 99;
    if (x == 0) {
        a = 100;
        return f(1);
    }
    else if (x == 1) {
        return a;
    }
    return -1;
}

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

4. Consider the following pseudocode:

```
var x = 100;
function setX(n) {x = n;}
function printX() {console.log(x);}
function first() {setX(1); printX();}
function second() {var x; setX(2); printX();}
setX(0);
first();
printX();
second();
printX();
```

What does this program print if the language uses static scoping? What does it print with dynamic scoping? Why?

- In static scoping, this results in an output of `1,1,2,2`. The second execution of `setX()` changes the variable `x` to 2. Therefore, the last `printX()` in the script prints 2.
- Dynamic scoping results in an output of `1,1,2,1`. The second `setX()` call changes the local `x` to 2, leaving the global `x` unaffected.

5. The expression `a - f(b) - c*d` can produce different values depending on how a compiler decides to order, or even parallelize operations. Give a small program in the language of your choice (or even one of your own design) that would produce differentvalues for this expression for different evaluation orders.

   An example program (parallel.js) is listed below.

```
var a = 1;
var b = 2;
var c = 3;
var d = 4;

function f(n) {
    c = 10;
    return c;
}

alert(a - f(b) - c * d);
```

   If the statement is evaluated from left to right, it will evaluate to: `1 - 10 - 10*4`, which equals `-49`. If the statement is evaluated in parallel, such that `f(b)` and `c*d` are evaluated independent of one another, it will be evaluated like this: `1 - 10 - 3*4`, which equals `-21`.

6. Explain the meaning of the following C declarations:

```
double *a[n];
double (*b)[n];
double (*c[n])();
double (*d())[n];
```

- `a` is array of `n` pointer to `double`.
- `b` is pointer to array of `n double`.
- `c` is array of pointer to function returning `double`.
- `d` is function returning pointer to array of `n double`.

7. Rewrite the four declarations in the previous problem in Go.

```
type doublef func() double

*[n]a double
[n]*b double
c := [n]doublef{...}
d := func() *[]double {...}
```

8. Translate the following expression: `(-b + sqrt(4 * a * c)) / (2 * a)`

(a) Postfix notation:
`b ∼ 4 a * c * sqrt + 2 a * /`

(b) Prefix notation:
`/ + ∼ b sqrt * * 4 a c * 2 a`

(*) Do you need a special symbol for unary negation? Why or why not?
Yes, we need a special symbol for unary negation (∼, for example). We need the symbol because we are not using parentheses in the postfix and prefix notations. You can't use the same symbol unless parentheses are allowed in the answer.

9. Interleave in C++ with C-style arrays and with Vectors

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

//  Problem #9
char* interleave(char a[], int aLength, char b[], int bLength) {
    unsigned length = aLength + bLength;
    char* result = new char[length + 1];
    result[length] = '\0';

    int i = 0;
    int j = 0;
    int location = 0;
    while (location < length) {
        if (i < aLength) {
            result[location] = a[i];
            i++;
            location++;
        }

        if (j < bLength) {
            result[location] = b[j];
            j++;
            location++;
        }
    }
    return result;
}

//  Problem #10
char* vectorInterleave(char a[], char b[]) {
    unsigned length = strlen(a) + strlen(b);
    vector<char> result;
    int i = 0;
    int j = 0;
    int location = 0;
    while (location < length) {
        if (i < strlen(a)) {
            result.push_back(a[i]);
            i++;
            location++;
        }

        if (j < strlen(b)) {
            result.push_back(b[j]);
            j++;
            location++;
        }
    }
    /* A vector guarantees that its elements occupy contiguous memory,
     * so data is at address of the first element.
     */
    return &result[0];
}

int main() {
    char a[5] = "Dude";
    char b[8] = "1234567";
    cout << interleave(a,4,b,7) << "\n";
    cout << vectorInterleave(a,b) << "\n";
    cout << interleave(b,7,a,4) << "\n";
    cout << vectorInterleave(b,a) << "\n";
    return 0;
}
```