Oregon State University

CS 352 - TRANSLATORS

Winter 2015

Milestone 1: Introduction to *gforth*

Author:
Drake Bridgewater

Professor:
Dr. Jennifer Parham-Mocello

DUE 01/16/15 (11:59pm)
January 21, 2015

Contents

1 Assignment

1.1 Introduction

The output of our compiler will be the input to the "machine." In C, the compiler generates an object file, the familiar .o file, that is then linked into the a.out file. To actually run the program you must still cause the operating system to load a.out file into memory for the machine to execute.

For our project, we will instead use the program gforth. gforth is almost completely opposite of Lisp. The syntactic format for gforth is postorder. While Lisp has some syntax, gforth only has spelling rules. As an example,

our 1+2 expression in C would be entered as

1 2 +

in gforth.

1.2 Objective

- Objective 1 is to introduce you to gforth. Gforth will be the machine that we code to.
  - reversing list: 1 2 3 rot rot swap .s
- Objective 2 is to emphasize the crucial role of generalized trees and generalized postorder traversal.
- Objective 3 is to get you to formulate a generalized expression tree data structure.

Professional Methods and Values

The professional will learn to use many programming languages and paradigms throughout her/his professional career. The hallmark of the professional in this venue is the ability to quickly master a new programming language or paradigm and to relate the new to the old.

Assignment

The below exercises are simple "Hello World" type exercises. These short program segments or small programs are designed to get you to understand how to learn a new language: you pretty much just sit down and try some standard things that you already know must work.

1.3 Performance Objectives

In this milestone, you have several clear performance objectives.

1. Learn to run gforth on either a Departmental machine or how to install and use gforth on your own machine.
2. Learn the simple programming style of gforth. Documentation/Download information is given on the links page.
3. **Translate a the infix style expressions in the Initial Forth Exercises 1 - 11 to an expression tree. The output here is a drawn tree. You do not need to include the printf statements as part of your tree. The printf is just to make sure you know to print the answer, not just calculate it.**
4. **Do a postorder traversal of the expression tree to generate the gforth input. The output of this step is gforth code.**
5. Produce running gforth code that evaluates the programs equivalent to initial exercises. The output here is the running of the gforth code.

1.4 Milestone Report

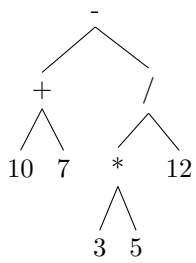
- Your milestone report will include hand written answers to 3 and 4 above.
- In addition, your milestone report must **include a data structure for an n-ary tree and a pseudo-code recursive algorithm to translate an arbitrary instance of these trees into postorder.**
- Use handin on the TEACH website to submit the gforth input file, makefile, and milestone report. We will generate the output file.
- Click on these links to see a template for the Milestone Report and Makefile we will use in this class.

2 Answers

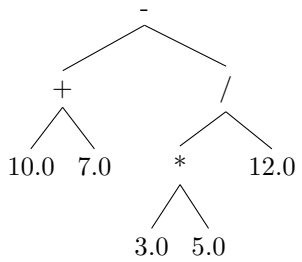
2.1 Forth Exercises

1. printf("Hello World n");
   Command for gforth:
   ." Hello World!"

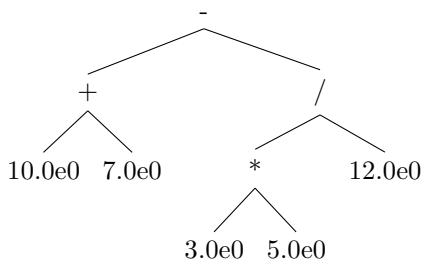
2. $10 + 7 - 3 * 5/12$
   Command for gforth:
   10 7 + 3 5 * 12 / - .s



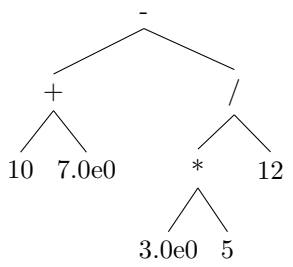
3. $10.0 + 7.0 - 3.0 * 5.0/12.0$
   Command for gforth:
   10.0e0 7.0e0 F+ 3.0e0 5.0e0 F* 12.0e0 F/ F- f.s



4. $10.0e0 + 7.0e0 - 3.0e0 * 5.0e0/12.0e0$
   Command for gforth:
   10.0e0 7.0e0 F+ 3.0e0 5.0e0 F* 12.0e0 F/ F- f.s



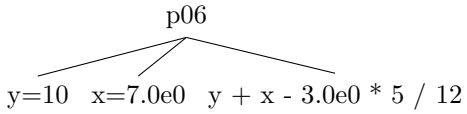
5. $10 + 7.0e0 - 3.0e0 * 5/12$
   Command for gforth:
   10 7.0e0 s>f F+ 3.0e0 5.0e0 F* 12 s>f F/ F- f.s



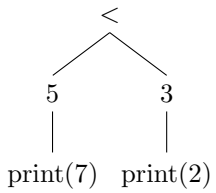
6. `y = 10;`
   `x = 7.0e0;`
   `y + x - 3.0e0 * 5 / 12`

   Command for gforth:
   : p06 7.0e0 10 { y F: x } y s>f x 3.0e0 5 s>f 12 s>f f/ f* f- f+ f. ;
   Explanation:

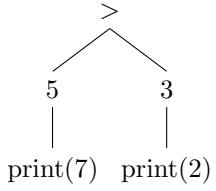
   - Put value on stack for variables
   - Redefine x and y
   - Set values and convert if necessary
   - Calculate value as necessary



7. if $5 < 3$ then 7 else 2
   Command for gforth:
   : TEST 5 4 < ." 7" IF CR ." 2" THEN ;

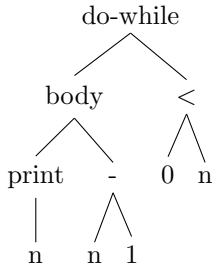


8. if $5 > 3$ then 7 else 2
   Command for gforth:
   : TEST 5 4 > ." 7" IF CR ." 2" THEN ;



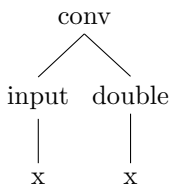
9.
```
for ( i = 0; i <= 5; i++ )
    printf("%d ", i); $
```

Command for gforth:
: TEST ( ) 5 BEGIN DUP . 1 - DUP 0 = UNTIL DROP ;



10.
```
double convertint(int x)
   { return ((double)x); }
```

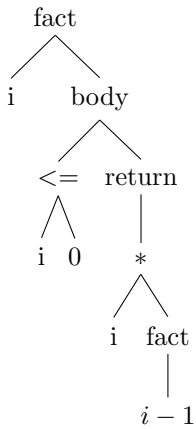
Command for gforth:
: TEST (n – r) s>f CR CR .s CR f.s ;



11.
```
int fact(int i)
{
    if (i <= 0 ) return 1;
    else return i*fact(i-1);
}
```

Command for gforth:

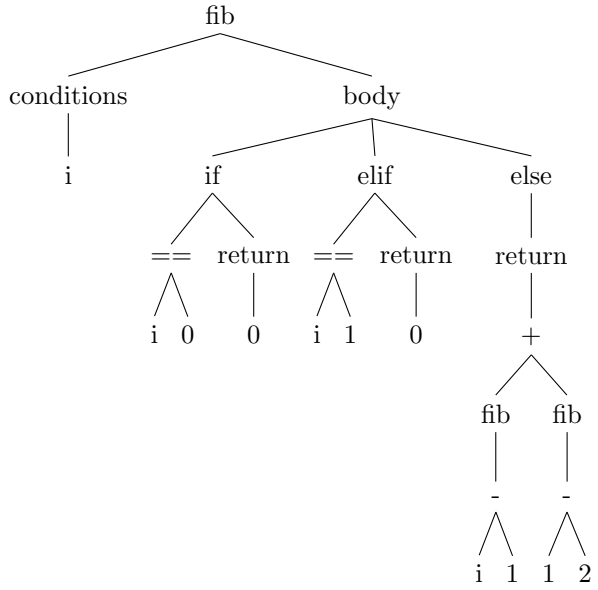
```
: p11 ( ) dup 0 <= if
        drop 1 else
        dup 1 - recurse *
        endif ;
```



12.
```
int fib(int i)
{
    if(i == 0) return 0;
    else if(i == 1) return 1;
    else return fib(i-1) + fib(i-2);
}
```

Command for gforth:

```
: p12 ( fib ) dup 0 = if
                drop 1 else
                dup 1 = if
                        drop 1 else
                         1 - dup 1 - recurse swap recurse +
                         endif
                endif .s ;
```



2.2 Report

This milestone is designed to get us thinking about the options of storing and accessing data. This could be seen from the request for our pseudo-code for a n-ary tree. As for how I accomplished the forth exercises over the course of a few days and it can be seen that my understanding in increased, but it should also be noted that the questions get progressively more difficult. In the end I needed to verify that the programs were producing the correct output. Therefore the functions that were implemented I used multiple values and compared with the actual. As for the mathmaticall problems I plug in the numbers in to python and I also plugged them in to a computational knowledge engine called WolframAlpha

2.3 pseudo-code for n-ary tree

An n-ary tree similar to a binary tree with the addition of more child nodes. This type of tree still contains only one root node and each node can contain many different properties allowing it to be great for accessing data. What makes it great for parsing lines of code is that the output language for this class that is desired is a post-fix language. Therefore, traversing a tree as post order will produce the correct semantics. At which point the grammar can be considered.

2.3.1 Things to consider when designing a n-ary tree

- layout: I have put layout here primarily because Dr. Jennifer Parham-Mocello has mentioned that there is a top down approach and a bottom up approach. She has told us that we our implementation will be a top down but I want to still think about how it could work as a bottom up. The primary reason Dr. Jennifer Parham-Mocello has said this is because we don't have time to implement a bottom up as it is more difficult.
- Functions
  - Add node
  - Moving to a node
  - Removing a node
  - Print entire tree (pre, post and in-order traversal)
  - Printing single node

2.4 Source For tree.py

```
__author__ = 'Drake'


class Node():
    def __init__(self, value):
        self.children = []
        self.value = value
        self.child_count = 0

    def get_number_children(self):
        return self.child_count

    def get_value(self):
        return self.value

    def get_left_child(self):
        return self.children[min(self.children)]

    def get_right_child(self):
        return self.children[max(self.children)]

    #   returns the child at idx
    def get_child_at(self, idx):
        if idx <= self.children.count():
            return -1
        return self.children[idx]

    #   Takes a value and return a child
    def get_child(self, child_value):
        for child in self.children:
            if child_value == child.value:
                return child

        # return a error that it does not contain value searching for
        return -1

    def add_child(self, child):
        self.children.append(child)
        self.child_count += 1

    def print_children(self, indent):
        for child in self.children:
            print("\t"*indent + child.value)


class Tree():
    def __init__(self, rootValue):
        self.root = Node(rootValue)
        self.currentLocation = self.root
        # equal one because we just added one
        self.size = 1

    def go_home(self):
```

```
        self.currentLocation = self.root

    def add_node(self, value):
        print("adding node with value: " + value)
        newNode = Node(value)
        self.currentLocation.add_child(newNode)
        self.size += 1

    def go_to_child_node(self, child_value):
        print("trying to move to: " + child_value)
        nextMove = self.currentLocation.get_child(child_value)
        print("Moving to child with value: " + nextMove.get_value())
        if nextMove != -1:
            self.currentLocation = nextMove
        else:
            print("**error could not get child to move to")

    def get_size(self):
        return self.size

    def print_tree(self):
        print("-"*80 + "\n\t print tree called")
        print self.root.get_value()
        self.print_tree_helper(self.root)

    def print_postordered_tree(self):
        print("-"*80 + "\n\t print post ordered tree called")
        print self.root.get_value()
        self.post_order_tree_print(self.root)

    def print_tree_helper(self, node, indent=0):
        indent += 1
        for child in node.children:
            # if child.get_child_count() > 0:
            print("\t"*indent + child.get_value())
            self.print_tree_helper(child, indent)

    def post_order_tree_print(self, node):
        for child in node.children:
            self.post_order_tree_print(child)
            print(child.get_value())
```

tree.py

2.5 Source for gforth

```
: p01 ." Hello World" ;
: p02 10 7 + 3 5 * 12 / - . ;
: p03 10.0e0 7.0e0 F+ 3.0e0 5.0e0 F* 12.0e0 F/ F- f. ;
: p04 10.0e0 7.0e0 F+ 3.0e0 5.0e0 F* 12.0e0 F/ F- f. ;
: p05 10 7.0e0 s>f F+ 3.0e0 5.0e0 F* 12 s>f F/ F- f. ;
: p06 7.0e0 10 { y F: x } y s>f x 3.0e0 5 s>f 12 s>f f/ f* f- f+ f. ;
: p07 5 4 < ." 7" IF CR ." 2" THEN ;
: p08 5 4 > ." 7" IF CR ." 2" THEN ;
: p09 ( ) 5 BEGIN DUP . 1 - DUP 0 = UNTIL DROP  ;
: p10 ( n -- r ) s>f CR CR .s CR f.s ;
: p11 ( fact ) dup 0 <= if
    drop 1 else
    dup 1 - recurse *
    endif .s ;
: p12 ( fib ) dup 0 = if
    drop 1 else
    dup 1 = if
      drop 1 else
      1 - dup 1 - recurse swap recurse +
      endif
    endif .s ;


p01 CR CR clearstack
p02 CR CR clearstack
p03 CR CR clearstack
p04 CR CR clearstack
p05 CR CR clearstack
p06 CR CR clearstack
p07 CR CR clearstack
p08 CR CR clearstack
p09 CR CR clearstack
5 p10 CR CR clearstack
5 p11 CR CR clearstack
5 p12 CR CR clearstack
10 p12 CR CR clearstack

BYE
```

forthExercises.fs