

CSI333 Lecture 2-done 8/30/07

Goals-Know how to work things and how things work-Learn to Think productively, creatively, “out of the box”

1. Machine/Assembly and C Languages under Unix: Instruction Set Architecture is instructions & datatypes of computer **hardware**. C related to machine features, comprehensive view of C.
2. C/C++/Java hardware/software/system interface topics: type implementation, modularization with multiple source file programs, functions, pointers, strings, arguments, preprocessor, etc.
3. Computer performance (i.e., **speed** of a program's execution and it's observation).
4. Larger, more mature projects and software structures. Tools (make, debugger, binary editors and object analyzers, version control repositories).
5. See Patt/Patel (PP, BLUE hardcover :(book) and Bryant/O'Hallaron) and writings of top SW developers for detailed motivation.

Project 1

Your job is to understand the algorithm based on these ideas and then implement it (for Boolean, not numeric expressions) by a C++ or Java program. HOMEWORK: practice it on paper.

Observations:

1. The value or operator accessed or removed from the corresponding “box” is ALWAYS the LAST ONE put in the box but not yet removed. Those boxes can be *stacks*, i.e., pushdown or Last-In-First-Out (LIFO) stores.
2. The input is processed left-to-right.
3. The decision to PUSH, or REDUCE, or REMOVE a “(” is based only on the current input token and the top of the Operator stack.
4. When two operators affect the PUSH or REDUCE decision, the decision is based on their relative *precedence* (and associativity).

Associativity

Associativity where it really matters to the IRS and bosses:

$$\text{Salary} - \text{Deduction1} - \text{Deduction2}$$

means

$$(\text{Salary} - \text{Deduction1}) - \text{Deduction2}$$

NOT

$$\begin{aligned} &\text{Salary} - (\text{Deduction1} - \text{Deduction2}) \\ &= (\text{Salary} - \text{Deduction1}) + \text{Deduction2} \end{aligned}$$

(The IRS receives deductions and bosses don't pay them to you!)
Subtraction denoted by $-$ is a *binary (Here, 2 operand) LEFT* associative operator.

In Project 1, operators $\&$ and $|$ are left-associative.

Precedence

$\text{Price1} + \text{Price2} * \text{DiscountFactor2}$

$\text{BoolOne} | \text{BoolTwo} \& \text{BoolThree}$

mean

$\text{Price1} + (\text{Price2} * \text{DiscountFactor2})$

$\text{BoolOne} | (\text{BoolTwo} \& \text{BoolThree})$

NOT

$(\text{Price}-1 + \text{Price}-2) * \text{Discount}-2$

$(\text{Bool}-1 | (\text{Bool}-2) \& \text{Bool}-3$

It has been a social convention among nerds for hundreds of years that multiplication and division beat addition and subtraction. Precedence and associativity are needed to decide between PUSH or REDUCE when there are 2 operators involved. You figure it out and debug it.

It is really a CS1-CS2 Review:

1. Implement and use two stacks.
(Stacks recur over and over in this course...how function linkage works is a topic.)
2. Organize the program so the BoolExpression class holds the method to do the evaluation. (Computer hardware is built of gates that do logical, i.e., *Boolean* operations on bits.)
3. Implement and use a “Dictionary” –data structure that stores
(1) some of the letters from A-Z and (2) for each letter stored, a Boolean (truth) value “true” or “false”. Also, implement operations to
 - 3.1 Print the row of 0’s and 1’s for the letters in alphabetical order.
 - 3.2 Calculate the next row of 0’s and 1’s, for example:
ABC
000
001
010
011 etc
 - 3.3 Find the current Boolean value for a given letter whenever evaluate() needs it.

Getting started.. C++ choice

Class definition in BoolExpression.h

```
#ifndef BoolExpression_h_included
#define BoolExpression_h_included
#include <string>
#include "charStack.h" //YOU code these .h files
#include "boolStack.h" //YOU code these .h files
//#include "symbolTable.h" //Procrastinate here!
//Handle expressions with 0's and 1's BUT NO LETTERS first
class BoolExpression
{
public:
    //public means the body of ANY KIND of function
    // can access or call these members.
    ... to be continued
```

public members of class BoolExpression

```
public:
    BoolExpression();
        //Default constructor inputs expression.
//Be simple (gift): exit or throw exception on failure.
//Constr. might initialize stacks and the symbol table.

//Extendable: OVERLOAD with another constructor
// which has a string parameter.
// Good idea???.

void printTruthTable(); //Achieve application goal.
```

To be continued...

Is it a good idea to overload the default constructor with a constructor which has a string parameter? Please form groups to answer this question and explain your answer.

Overloading: When class definition has

```
class BoolExpression
{
    ....
    BoolExpression();           //default constructor
    BoolExpression(String x); //overloaded constructor
    ....};
```

When we execute

```
BoolExpression MB;
```

a BoolExpression object is instantiated and the default constructor body is called.

When we execute

```
BoolExpression MB("1+1&0");
```

a BoolExpression object is instantiated and the OTHER constructor body is called.

Results

Yes 9	Undecided 1	No 1
Explanation included "if" 5		

This kind of question doesn't have one right answer.
It is unlike the question " $3 + 4? = 9$ "

BoolExpression.h Continued

```
private: //means ONLY BoolExpression's methods can
        //access or call these.

std::string expr; //need std:: since using namespace
                //is BAD in header files.
//Hold the expression.
// i-th char is expr[i]. 0 <= i <= expr.length()-1
// end char is '\0'
//use getline(cin, expr) to read it.

bool evaluate( ); //get value for ONE TRUTH TABLE LINE
// body has the code for the algorithm
// outlined in the assignment.
```

Private part of class BoolExpression cont'd ...

```
charStack *pOpStack; //use new to allocate the
boolStack *pValStack; //stack objects we point to
//symbolTable *pletterValueTable;
//helpers...
bool valAtom( char x);
bool isAtom( char x);
void reduce( ); //The famous reduce step..
// pop 1 operator, 1-2 values, calculate and push result
bool figure_out_precedence( char x, char y);
// void fill_symbol_table(); //scan expr and put the let
// ??? whatever else you need.
}; //REMEMBER THE ; at the end of a class definition
#endif //match the #ifndef include guard
```

Put the main function in BoolExpression.cpp

```
// file BoolExpression.cpp
#include "BoolExpression.h"
#include <iostream> //need for cin, cout, getline, etc.
#include <string>
using namespace std; //use usings in .cpp files only, not in .h files
int main(int argc, char *argv[]) //command line args NOT USED
{
    BoolExpression myBoolExpression; //constructor reads it in
    // alternative, Java-style:
    // BoolExpression *pBE = new BoolExpression;
    myBoolExpression.printTruthTable();
    // alternative:
    // pBE->printTruthTable();
    return 0; //success return code.
}
```

How BoolExpression.cpp reads input:

```
BoolExpression::BoolExpression() //default constructor
{
    getline(cin, expr);
    //Your choice: initialize symbol table here or in printTru
    //I feel doing it later is more elegant.
}
// code the other methods of BoolExpression below.
```

Starting, Inputting a String in Java

```
class BoolExpression {
    public static void main(String[] args) {
        (new BoolExpression).printTruthTable();
        return; }
    private String expr;
    BooleanExpression()
    {
        System.out.print("Enter a Boolean Expression:");
        java.util.Scanner myScanner =
            new java.util.Scanner( System.in );
        expr = myScanner.nextLine();
    }
}
```

Starting with Java– cont'd

```
private boolean evaluate()  
{  
    if( expr.charAt( 0 ) == '0' ) return false;  
    if( expr.charAt( 1 ) == '1' ) return true;  
    System.out.println("evaluate is just a stub now.");  
    return false;  
}
```

Starting with Java– cont'd

```
private void printTruthTable() //also a stub
{
    System.out.println(" " + expr); //reprint expression.
    System.out.print(" "); //print leading space
    boolean value = evaluate();
    if(value) System.out.println("1");
    else System.out.println("0");
    return;
    //Java prints boolean values as "true" or "false"
    //NOT "0" or "1" as required, and as in C++
}
} //no ; needed after Java class definitions!
```