# Tut 1: VAL

### 1.a

***Question:***

int f(int a, int b) {

while ( a != b )

if ( a < b )

b -= a ;

else

a -= b ;

return a ;

}

***Answer:***

// assumption: eax stores the value

// of a, and ebx stores the value of b

void exec(){

loop: // the while loop

// termination condition

if(eax == ebx) goto end;

if(eax >= ebx) goto branch1; // else condition

// the if condition

ebx -= eax;

goto endBranch1;

branch1:

eax -= ebx;

endBranch1:{};

goto loop;

end:{}

}

### 1.b

***Question:***

extern int a[];

int f() {

return a[0] - a[1]\*a[2]\*(a[1]+a[2]) - a[3] << a[4] ;

}

***Answer:***

// assumption: M's position: 0,3; 4,7; 8,11; 12,15;

// stores the integer indexed 0,1,2,3 respective

// and the final result is stored in eax

void exec(){

eax = \*((int\*)&M[0]);

// calculate the claster for multiplication

ebx = \*(int\*)&M[4];

ebx \*= \*(int\*)&M[8];

ecx = \*(int\*)&M[4];

ecx += \*(int\*)&M[8];

ebx \*= ecx;

eax -= ebx;

eax -= \*(int\*)&M[12];

eax <<= \*(int\*)&M[16];

}

### 1.c

***Question:***

int f(int x) {

switch ( x ) {

case 0 :

return 13 ;

case 1 :

return 8 ;

case 2 :

return 5 ;

case 3 :

return 3 ;

default :

return -1 ;

}

}

***Ans:***

// assumption: eax stores the value of x

void exec(){

if(eax == 0)goto c0;

if(eax == 1)goto c1;

if(eax == 2)goto c2;

if(eax == 3)goto c3;

goto cDefault;

c0:

ebx = 13;

goto endSwitch;

c1:

ebx = 8;

goto endSwitch;

c2:

ebx = 5;

goto endSwitch;

c3:

ebx = 3;

goto endSwitch;

cDefault:

ebx = -1;

endSwitch:{};

end:{};

}

# Tut 2:

## Q1 - Ruby

***Q:*** Give 3 str to match given regular expression

***Ans:***

#1.a

puts "1.1: "

x = /(a[bcd]\*(aa)+){2,4}/

puts x

# matches

puts (x.match "aaaaaaaaaaaa")

puts (x.match "adaaaaadaaaa")

puts (x.match "abaaaaabaaaaaa")

# do not match

puts (x.match "aaabaababbaaa")

#1.2

puts "1.2: "

x= /(a|bc)(d?|e)\*/

puts (x.match "ade")

puts (x.match "bcdef")

puts (x.match "bce")

#do not match

puts (x.match "o")

#1.3

puts "1.2: "

x= /(((a|bc)d+)e)+/

puts (x.match "ade")

puts (x.match "bcddde")

puts (x.match "adddebcdde") # the '+' after parentheses means match only; not necessarily means to repeat

#do not match

puts (x.match "o")

puts "something"

gets

### Q2 – DFA

### Q3 – Regular Grammar

## Q4 – C procedure

# Tut 3:

1. Define a Prolog predicate that takes two list arguments, and succeeds when the second list is the reverse of the first.

rev([],[]):-!.

rev([H|T],B):-

rev(T,C),append(C,[H],B).

2. Write a Prolog program that implements the quicksort algorithm. Consider using the cut (!) to make your program more efficient.

part(\_X,[],[],[]):-!.

part(X,[H|T],HI,LO):- H<X,LO=[H|TL],part(X,T,HI,TL).

part(X,[H|T],HI,LO):- H>=X,HI=[H|TH],part(X,T,TH,LO).

qs([],[]).

qs([H|T],L):- part(H,T,HI,LO),qs(HI,SH),qs(LO,SL),append(SL,[H|SH],L).

3. A type of virtual machine that is often used as the target architecture of compilers is the stack machine. A simplified version of such a machine has the following features:

- A stack that can store integers, which can be accessed in a last-in-first-out fashion.

- Two types of instructions:

o Data transfer instructions: push C, where C is a constant; this instruction pushes the constant C onto the top of the stack.

o Arithmetic instructions: add, sub, mul, div; these instructions pop two integers from the stack, perform the operation indicated by the instruction’s name, and push the result back.

•A program is a sequence of instructions, separated by semicolon; the instructions are executed in the order in which they appear in the program.

•At the end of the program’s execution, the result of the program is the integer found at the top of the stack.

Write a Prolog predicate that simulates an execution of such a machine. Your predicate should take in as argument the program to be run, and should return the result of the

program.

Sample run of the program:

?- exec((push 2 ; push 3 ; push 4 ; add ; mul), Result).

Result = 14

The program pushes the values 2, 3, and 4 on the stack. Then, the add instruction pops 3 and 4, and pushes back 7. Further, the mul instruction pops 7 and 2, and pushes back 14. Since the program has terminated, the result is the integer at the top of the stack,

which is 14.

:- op(100,fx,push).

inst(push K,S,[K|S]):- !.

inst(add, [A,B|S],[C|S]) :- C is A+B.

inst(sub, [A,B|S],[C|S]) :- C is A-B.

inst(mul, [A,B|S],[C|S]) :- C is A\*B.

inst(div, [A,B|S],[C|S]) :- C is A/B.

exec(P,R) :- exec(P,[],[R]).

exec(S1;S2,I,O):-

inst(S1,I,AUX), exec(S2,AUX,O).

exec(S,I,O) :- inst(S,I,O).

4.

:- op(100,fx,push).

inst(push K,S,[K|S]):- !.

inst(add, [A,B|S],[C|S]) :- C is A+B.

inst(sub, [A,B|S],[C|S]) :- C is A-B.

inst(mul, [A,B|S],[C|S]) :- C is A\*B.

inst(div, [A,B|S],[C|S]) :- C is A/B.

exec(P,R) :- exec(P,[],[R]).

exec(S1;S2,I,O) :- inst(S1,I,AUX), exec(S2,AUX,O).

exec(S,I,O) :- inst(S,I,O).