

Compiler Design, Handout No.2, Assignment No.2

Tracing Grammars and constructing FA for some instructions in C++

Recall: To find a grammar of a language, the best is to construct its FA and then use FA to write its grammar.

Examples. Write the grammar of the following FAs.

FA(Finite Automata)	Grammar or CFG(Context Free Grammar)
<p>i) $\Sigma = \{a, b\}$, $L = aba^*$</p> <pre> graph LR A((A)) -- a --> B((B)) B -- b --> C((C)) C -- a --> C style A fill:none,stroke:none style B fill:none,stroke:none style C fill:none,stroke:none </pre>	<p> $A \rightarrow aB$ $B \rightarrow bC \mid \lambda$ $C \rightarrow aC$ $C \rightarrow \lambda$ Terminals or alphabets = $\{a, b\}$, Non-terminals or states = $\{A, B, C\}$ </p>
<p>ii) Given $L = a^* + b^*$, find its CFG 1st, construct FA for the language. 2nd use the FA to write the CFG of the language</p> <pre> graph LR X((X)) -- b --> B((B)) B -- a --> A((A)) B -- b --> B A -- a --> A style X fill:none,stroke:none style B fill:none,stroke:none style A fill:none,stroke:none </pre> <p> X is final: $L1 = \lambda$ A is final: $L2 = aa^*$, B is final: $L3 = bb^*$ $L = L1 + L2 + L3 = \lambda + aa^* + bb^*$, write $\lambda = \lambda + \lambda$ $= (\lambda + aa^*) + (\lambda + bb^*) = a^* + b^*$ </p>	<p> $X \rightarrow aA \mid bB \mid \lambda$ $A \rightarrow aA \mid \lambda$ $B \rightarrow bB \mid \lambda$ </p>
<p>iii) Given $L = (a+b)^*$, write the CFG of L</p> <pre> graph LR X((X)) -- a --> X X -- b --> X style X fill:none,stroke:none </pre>	<p>$X \rightarrow aX \mid bX \mid \lambda$</p>

Tracing grammars: To determine whether a given word is accepted or rejected by a grammar, we use one of the following two methods:

- Parse tree
- Left-most-derivation

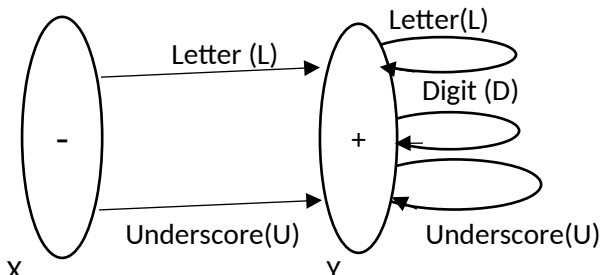
Example. Trace the following CFG to determine whether word w is accepted or rejected:

CFG: $A \rightarrow aB$, $B \rightarrow bC$, $C \rightarrow aC \mid \lambda$. Terminals= $\{a,b,c\}$, Non-terminals= $\{A,B,C\}$

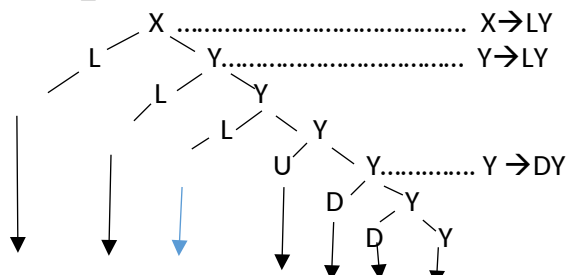
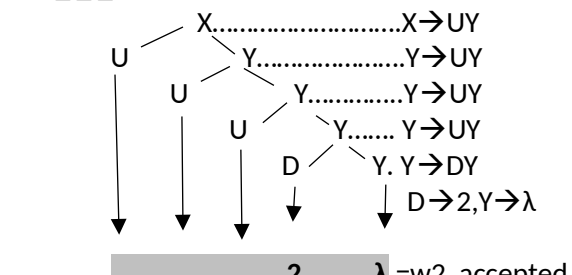
Parse tree method	Left-most-derivation method
<p>i) $w_1 = abaaa$: Start with the first rule: $A \rightarrow aB$. Since "a" is terminal we stop, but "B" is nonterminal, we expand it by using rule: $B \rightarrow bC$. Stop when there are no more non-terminal letters exist.</p> <p>rule used</p> <p>$A \rightarrow aB$</p> <p>$B \rightarrow bC$</p> <p>$C \rightarrow aC$</p> <p>$C \rightarrow aC$</p> <p>$C \rightarrow aC$</p> <p>$C \rightarrow \lambda$</p> <p>$\lambda = w_1$</p>	<p>i) $w_1 = abaaa$: Start with the first rule $A \rightarrow aB$, go from left-to-right, a is terminal but B is non-terminal, replace B with $B \rightarrow bC$. Skip all terminals until you see a non-terminal, expand the non-terminal. Do this until there are no more non-terminals left in the grammar, the word is accepted if there are no more non-terminals left and reject otherwise.</p> <p>$A \rightarrow aB$, replace B by bc $\Rightarrow abC$, skip a,b and replace C with aC $\Rightarrow abaC$, replace C with aC $\Rightarrow abaaC$, replace C with aC $\Rightarrow abaaaC$, replace C with λ $\Rightarrow abaaa\lambda = w_1$ stop, w_1 is accepted</p> <p>Note-1. In left-most-derivation you must only do one substitution for each non-terminal from left-to-right. For example: $A \rightarrow aB$, you can replace B with bC $\Rightarrow abC$, with only one substitution</p> <p>But, use both $B \rightarrow aC$ and $C \rightarrow aC$ to get $A \rightarrow aB$ $\Rightarrow abaC$ is illegal, two substitutions at once!</p> <p>Note-2: In left-most-derivation, only the first line is the actual given grammar and we use $A \rightarrow aB$, using single arrow, after that is all substitutions and we must use $A \Rightarrow abC$, arrow with two lines \Rightarrow</p> <p>ii) Trace $w_2 = abaab$</p> <p>$A \rightarrow aB$ $\Rightarrow abC$ $\Rightarrow abaC$ $\Rightarrow abaaC$ $\Rightarrow abaa$(does not generate b)</p> <p>CFG rejects w_2</p>

Now, let's see the applications of what we have covered so far.

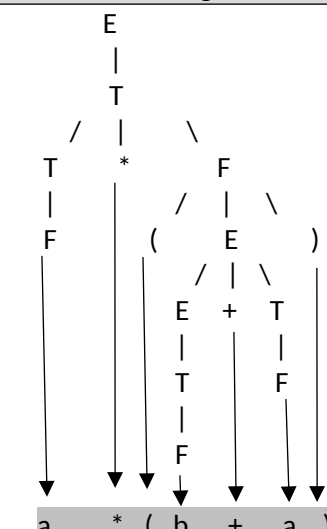
Example. An Identifiers in C++ could be string of letters, digits, or underscores. Identifiers must begin with a letter or underscore. Construct a grammar for identifiers in C++

FA representing the grammar of identifiers in C++	CFG
 <p>Must begin with a letter or underscore, followed by more letters, digits, or underscores</p>	$X \rightarrow LY \mid UY$ $Y \rightarrow LY \mid DY \mid UY \mid \lambda$ <p> $D \rightarrow 0 \mid 1 \mid 2 \mid \dots \mid 9$, digits are 0,1,...,9 $U \rightarrow _$, underscore is $_$ $L \rightarrow 'A' \mid 'B' \mid \dots \mid 'Z' \mid 'a' \mid 'b' \mid \dots \mid 'z'$ Letters are 'A'..'Z', 'a'..'z' </p> <p> Terminals = {a..z, A..Z, 0..9, $_$} Non-terminals = {X, U, L, D, Y} </p>

Example: Trace the above CFG to determine whether each identifier w_i is accepted or not:

<p>$w_1 = \text{Tax_23}$</p>  <p>T a x _ 2 3 λ =accepted</p>	<p>$w_2 = _ _ _ 2$</p>  <p>_ _ _ 2 λ =w2 accepted</p>
--	--

Example. Use the given CFG. Trace the grammar to determine whether $w = a^*(b+a)$ is accepted or not.

CFG	Parsing tree	Left-most-derivation
$E \rightarrow E+T$ $E \rightarrow E-T$ $E \rightarrow T$ $T \rightarrow T^*F$ $T \rightarrow T/F$ $T \rightarrow F$ $F \rightarrow (E)$ $F \rightarrow a$ $F \rightarrow b$	 <p>a * (b + a)</p>	$E \rightarrow T$ $\Rightarrow T^* F, T \rightarrow T^* F$ $\Rightarrow F^* F, T \rightarrow F$ $\Rightarrow a^*, F \rightarrow a$ $\Rightarrow a^* (E), F \rightarrow (E)$ $\Rightarrow a^* (E+T), E \rightarrow E+T$ $\Rightarrow a^* (T+T), E \rightarrow T$ $\Rightarrow a^* (F+T), T \rightarrow F$ $\Rightarrow a^* (b+T), F \rightarrow b$ $\Rightarrow a^* (b+F), T \rightarrow F$ $\Rightarrow a^* (b+a), F \rightarrow a$ <p>Note: ONLY one substitution at each step</p>

Example. Given the following FA with $\Sigma=\{a, b, c\}$. We want to write a program to determine whether a given word $w=abcc\$$ ($\$$ marks the end of string) is accepted or rejected by an FA.

1. Given the following FA

First, we must complete the FA, means at each state what will be happened if the input is a, b, or c. If an input letter it is not defined, we will add a new NULL state, and dump that input into the NULL state.

2. Complete the given FA. Add a new state **s5**, and dump any input a, b, or c which are not defined at existing states into the new state s5. Now, since s5 is part of this FA, complete that state as well by a loop with labels a, b, and c.

3. Construct a transition table for this new FA

Input States	a=0	b =1	c = 2
0	1	2	2
1	1	3	3
2	4	4	2
3	5	5	5
4	4	4	4
5	4	4	4

Initial state=s0 or just 0

Final states=s2 and s5 or just 2 , 5

At state o, input a takes us to state 1, input b to state 2 and input c to also state 2, The same for other states.

introduce the transition table to your program using a 2-dimensional array: Table[6][3]

<pre>int Table[6][3]={ {1,2,2}, {1,3,3},{4,4,2}, {5,5,5},{4,4,4},{4,4,4} }; int state=0, i=0, col=0; // start at state 0 string w="abcc\$";//input sting to test while (i < w.length()) { switch(w[i])//find col for each input { case 'a' : col=0; break; case 'b' : col=1; break;</pre>	<pre> case '\$': if (state==2 state==5) cout<<w<<" is accepted\n"; else cout<<w<<" is rejected\n"; break; } //end of switch state= Table[state][col]; //go to the next state ++i; //go to the next input letter</pre>
--	---

case 'c' : col=2; break;	}// end of while loop
--------------------------	-----------------------

Trace the program segment for input: abbcc\$

Transition table				C++ code		Tracing the C++ codes for w=abbcc\$			
Input State(s)	a=0	b=1	c=2	<pre> int Table[6][3]={ {1,2,2}, { 1,3,3},{4,4,2}, {5,5,5}, { 4,4,4}, {4,4,4} }; int state=0, i=0, col=0; // start at state 0 string w="abbcc\$"; while (i < w.length()) { switch(w[i]) { case 'a' : col=0; break; case 'b' : col=1; break; case 'c' : col=2; break; case '\$': if (state==2 state==5) cout<<w<<" accepted"; else cout<<w<<" rejected"; break; } //end of switch //go to the next state state= Table[state][col]; //go to the next input letter ++i; } // end of while </pre>		state	W[i]	col	[state][col]
0	1	2	2			0	a	0	[0][0]=1
1	1	3	3			1	b	1	[1][1]=1
2	4	4	2			1	b	1	[1][1]=1
3	5	5	5			1	c	2	[1][2]=3
4	4	4	4			3	c	2	[3][2]=5
5	4	4	4			5	\$	State=5, the Input is accepted	

Example. Write a CFG for numbers of types int and float in C++

FA	CFG
<p>Unsigned int (only positive integers with no sign, such as 123)</p> <p>$D \rightarrow 0 1 2 ... 9$</p>	<p> $U \rightarrow DX$; one digit $X \rightarrow DX \lambda$; or more digits $D \rightarrow 0 1 2 3 ... 9$ </p>
<p>int (such as : +25, -25, or just 25)</p>	<p> $I \rightarrow SU$; sign or no sign followed by $U \rightarrow DX$; unsigned digit $X \rightarrow DX \lambda$; or more digits $D \rightarrow 0 1 2 ... 9$ $S \rightarrow + - \lambda$ </p>
<p>float (such as: +25., -25., 25.3, +.123)</p> <p> $F \rightarrow I. I.U I S.U$ $I \rightarrow SU; U \rightarrow DX; X \rightarrow DX \lambda;$ $D \rightarrow 0 1 2 3 ... 9$ $S \rightarrow + - \lambda$ </p> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <p>Integer followed by . 25. +25. -25.</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <p>Integer . unsigned integer +25.13, -25.13, 25.13</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <p>Signed . unsigned integer +.25, -.25, .25</p> </div> <div style="border: 1px solid black; padding: 5px; margin: 5px;"> <p>Integer numbers are also considering as floating numbers</p> </div>	

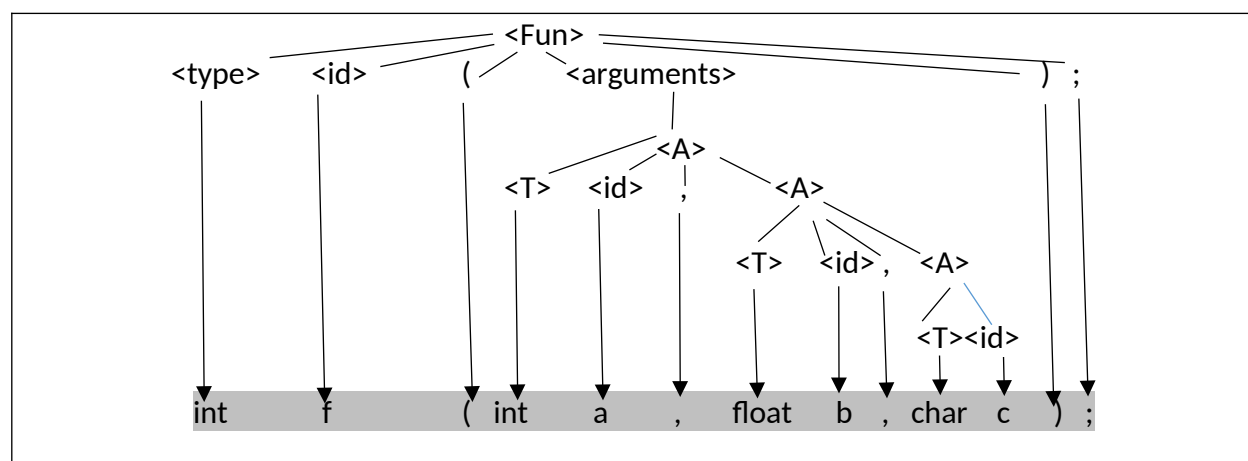
Example. Trace the above CFG for $n1=+123$, $n2= -.02$, $n3= -2.5$

<p>CFG: F is the initial state $F \rightarrow I. I.U S.U I U$ $I \rightarrow SU$ $U \rightarrow DX$ $X \rightarrow DX \lambda$ $D \rightarrow 0 1 2 ... 9$ $S \rightarrow + - \lambda$</p>	<p>$n1=+123$</p>
<p>$n2 = -.25$</p>	<p>$n3=-2.5$</p>

Example. Write a CFG for simple function's prototypes in C++

Examples of function's prototype in C++	CFG for function's prototype in C++
a. void f() b. void f(int a, int b, int c); c. int f(int a, int b, int c); 	$\langle \text{Fun} \rangle \rightarrow \langle T \rangle \langle \text{id} \rangle (\langle \text{arguments} \rangle) ;$ $\langle T \rangle \rightarrow \text{void} \mid \text{int} \mid \text{char} \mid \dots$ $\langle \text{id} \rangle \rightarrow \text{few examples before}$ $\langle \text{arguments} \rangle \rightarrow \langle A \rangle$ $\langle A \rangle \rightarrow \langle T \rangle \langle \text{id} \rangle, \langle A \rangle \mid \langle T \rangle \langle \text{id} \rangle \mid \lambda$

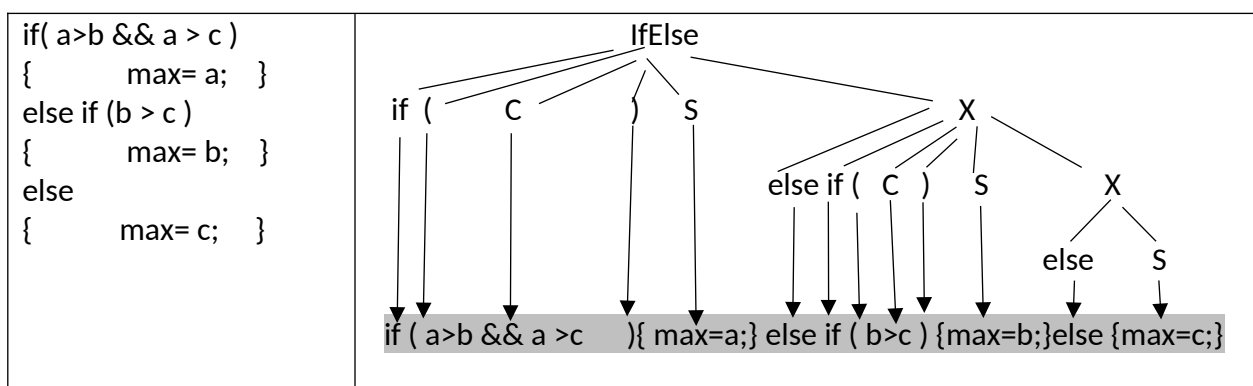
Example. Trace the following function's prototype: int f(int a, float b, char c);



Example. Write a CFG for if-else statements in C++

Examples of if-else statement in C++	CFG for if-else statements in c++
i. if(a) cout<<a; ii. if(a > b) { Max=a; cout<<Max<<endl; } else { <statements> } iii. if(<condition>) { <statements> } else if (<condition>) { <statements> } else if (<condition>) { <statement> } else { <statements> }	$\langle \text{if-else} \rangle \rightarrow \text{if} (\langle \text{condition} \rangle) \{ \langle \text{statements} \rangle \}$ $\rightarrow \text{if} (\langle \text{condition} \rangle) \{ \langle \text{statements} \rangle \} \text{else} \{ \langle \text{statements} \rangle \}$ $\rightarrow \text{if} (\langle \text{condition} \rangle) \{ \langle \text{statements} \rangle \}$ else if (<condition>) { <statement> } else if (<condition>) { <statement> } else { <statements> } or IfElse $\rightarrow \text{if} (C) SX$ X $\rightarrow \lambda \mid \text{else } S \mid \text{else if} (C) SX$ C $\rightarrow \text{condition}$ S $\rightarrow \text{statement}$

Example. Trace the following :



(CFG of some statements in C++)

50 points

1. (8 points) Consider the following grammar:

$S \rightarrow aSbB \mid A \mid c$

$A \rightarrow cA \mid c$

$B \rightarrow d \mid A$

Trace the grammar to determine which of the following words are accepted or rejected?

- i. accbc (use parse tree) ii. acccdd (use left-most-derivation)

Accepted

Not Accepted

2. (8 points) Given the following CFG:

$S \rightarrow I = E$

$E \rightarrow E + T \mid E - T \mid T$

$T \rightarrow T * F \mid T / F \mid F$

$F \rightarrow (E) \mid I$

$I \rightarrow a \mid b$

Use parsing tree to trace the grammar and decide which of the following statements are accepted or rejected.

- i. $a = a * (b - a * a)$

Accepted

- ii. $b = a * b - b * (a + b)$

Accepted

3. (8 points) Find the language of the following grammars:

a. $S \rightarrow aS \mid bB \mid aA \mid \lambda$

$B \rightarrow bB \mid aA$

$A \rightarrow aA \mid bA \mid \lambda$

$L = a^* + a^*bb^*a(a+b)^* + a^*a(a+b)^*$

b. $S \rightarrow aS \mid bA \mid \lambda$

$A \rightarrow aA \mid bX \mid \lambda$

$X \rightarrow aX \mid bX \mid \lambda$

$L = a^* + a^*ba^* + a^*ba^*b(a+b)^*$

4. (9 points) Find a CFG for each of the following languages

$S \rightarrow aA \mid bB \mid \lambda$ (i) $L = a^* + b^*$ (ii) $L = a^*b^*c^*$ (iii) $L = ab^* + ba^* + c$ $S \rightarrow aA \mid bB \mid cC$

$A \rightarrow aA \mid \lambda$ $S \rightarrow aS \mid bB \mid \lambda$ $A \rightarrow bA \mid \lambda$

$B \rightarrow bB \mid \lambda$ $B \rightarrow bB \mid cA \mid \lambda$ $B \rightarrow aB \mid \lambda$

Programming assignment

$A \rightarrow cA \mid \lambda$

$C \rightarrow \lambda$

1. (17 points) Write a program to find the value of a postfix expression. Variables are one or more characters each. We might have some integer numbers as part of the expression. (Hint. Read each part of the expression as a token of type string, if the first character of the token is a letter that indicates the token is a variable name, push its value in stack. If the token is an integer number use the predefined function stoi (member of <string> library) to return its numeric value of the token.

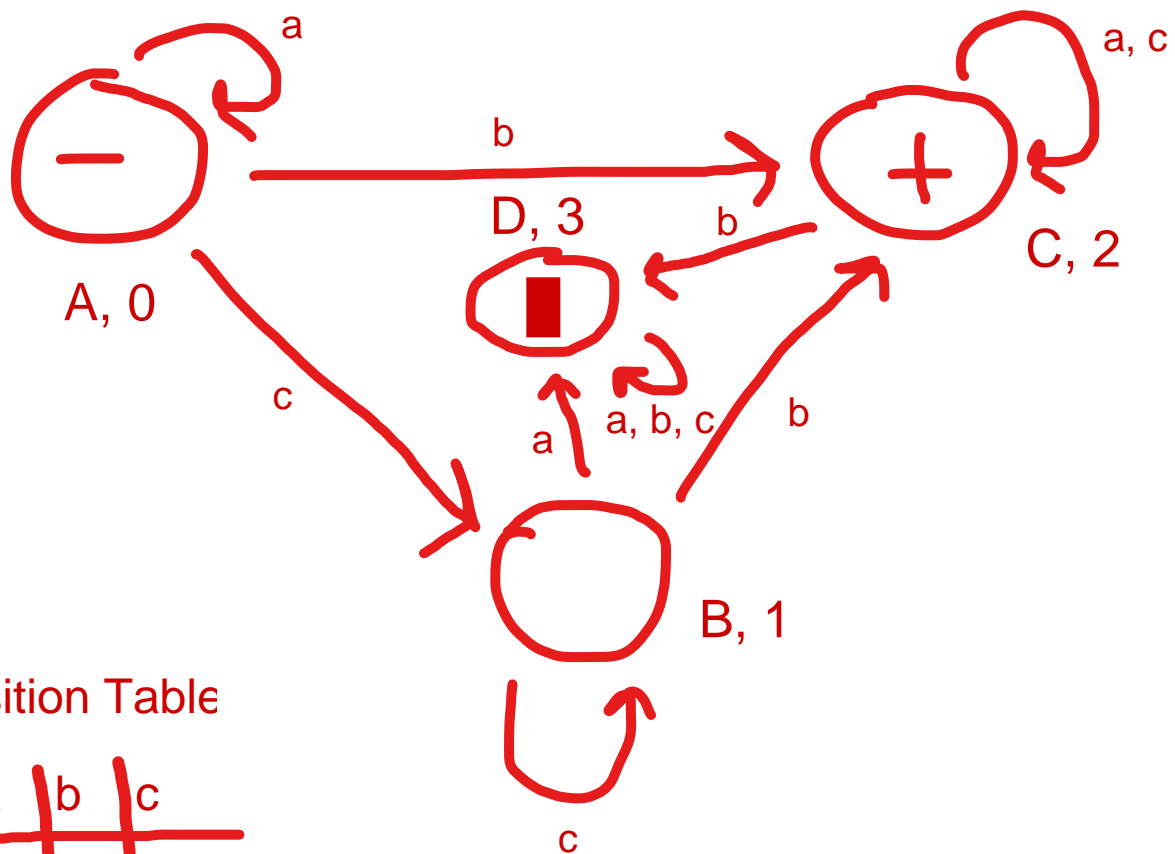
Sample I/O:

<p>Enter a postfix expression with a \$ at the end: 20 jerry 45 + tom - * \$</p> <p>Enter the value of jerry: 10</p> <p>Enter the value of tom: 5</p> <p>Expression's value is 1000</p> <p>CONTINUE(y/n)? y</p>	<p>Enter a postfix expression with a \$ at the end: myscore yourscore 45 + 100 + * \$</p> <p>Enter the value of myscore: 3</p> <p>Enter the value of yourscore: 5</p> <p>Expressions value is 450</p> <p>CONTINUE(y/n)? n</p>
---	---

As input, please use the given expressions and the given values.

2. Given the language $L = a^*c^*b(a+c)^*$ with alphabet = {a,b,c}. (1) Construct an FA to accept L. (2) complete the FA (3) construct a transition table for the new FA. (4) Write a program to decide whether the following words are members of L or not

W1 = bac W2 = aa b a³ W3 = baaccb



Transition Table

	a	b	c
0	0	2	1
1	3	2	1
2	2	3	2
3	3	3	3