CSE450 Exam Cheat Sheet

Regular Expressions

`*` Matches the previous element zero or more times.
`+` Matches the previous element one or more times.
`?` Matches the previous element zero or one time.
`{n}` Matches the previous element exactly n times.
`{n,}` Matches the previous element at least n times.
`{n,m}` Matches the previous element at least n times, but no more than m times.
`[character_group]` Matches any single character in `character_group`. By default, the match is case-sensitive.
`[^character_group]` Negation: Matches any single character that is not in `character_group`. By default, characters in `character_group` are case-sensitive.
`[first-last]` Character range: Matches any single character in the range from *first* to *last*.
. Matches any single character in the Unicode general category or named block specified by name.
^ The match must start at the beginning of the string or line.
$ The match must occur at the end of the string or before `\n` at the end of the line or string.

Project 5 solution lex

```
VAL_LITERAL          r'((\d+)(\.\d+)?)|(\.\d+)'
CHAR_LITERAL     r"'([^\\']|\\n|\\t|\\'|\\\\)'"
STRING_LITERAL  r'"([^\\"]|\\n|\\t|\\"|\\\\)*"'
ID                    r'[a-zA-Z_][a-zA-Z_0-9]*'
ASSIGN_ADD     r'\+='  | ASSIGN_SUB       r'\-='
ASSIGN_MULT    r'\*='  | ASSIGN_DIV        r'/='
COMP_EQU        r'=='  | COMP_NEQU         r'!='
COMP_LTE        r'<='  | COMP_GTE          r'>='
COMP_LESS        r'<'  | COMP_GTR           r'>'
BOOL_AND        r'&&'  | BOOL_OR          r'\|\|'
WHITESPACE   r'[ \t]'  | COMMENT     r'\#[^\n]*'
newline        r'\n+'  |
```

Context Free Grammars

CFGs Consist of 4 components (Backus-Naur Form or BNF):
Terminal Symbols = token or ϵ
Non-terminal Symbols = syntactic variables
Start Symbol S = special non-terminal
Production Rules of the form LHS $\rightarrow$ RHS

- LHS = A single non-terminal
- RHS = A string of terminals and non-terminals
- Specify how non-terminals may be expanded
- By default, the LHS of the first production rule is the Start Symbol

$S \rightarrow aSa$
$S \rightarrow T$
$T \rightarrow bSb$
$T\epsilon$

Shorthand - vertical bar '|' to combine multiple productions
$S \rightarrow aSa|T$
$T \rightarrow bTb|\epsilon$

project 5 CFG

```
program : statements

statements :

statements : statements statement

 statement    : expression ';'
              | print_statement ';'
              | declaration ';'
              | block
              | if_statement
              | while_statement

statement : ';'

statement : FLOW_BREAK ';'

if_statement :
FLOW_IF '(' expression ')' statement %prec IFX

if_statement :
FLOW_IF '(' expression ')' statement FLOW_ELSE statement

while_statement :
FLOW_WHILE '(' expression ')' statement

block : '{' new_scope statements '}'

"new_scope :"

print_statement :
COMMAND_PRINT '(' non_empty_comma_sep_expr ')'

non_empty_comma_sep_expr : expression

non_empty_comma_sep_expr :
non_empty_comma_sep_expr ',' expressi\usepackage{tikz}
\usetikzlibrary{shapes}on

expression : var_usage '=' expression

expression : expression '+' expression
| expression '-' expression
| expression '*' expression
| expression '/' expression

expression : '-' expression %prec UMINUS

expression : '!' expression

expression : var_usage ASSIGN_ADD expression
| var_usage ASSIGN_SUB expression
| var_usage ASSIGN_DIV expression
| var_usage ASSIGN_MULT expression

expression : expression COMP_EQU expression
| expression COMP_NEQU expression
| expression COMP_LTE expression
| expression COMP_LESS expression
| expression COMP_GTR expression
| expression COMP_GTE expression


expression :
expression BOOL_AND expression
| expression BOOL_OR expression


simple_declaration : type ID


assign_declaration : simple_declaration '=' expression


expression : ID '.' ID '(' ')'


statement : ID '.' ID '(' expression ')'


declaration : simple_declaration
| assign_declaration


var_usage : ID


expression : var_usage


expression : STRING_LITERAL


expression : CHAR_LITERAL


expression : '(' expression ')'


type : ARRAY_KEYWORD '(' TYPE ')'


var_usage : ID '[' expression ']'


type : STRING_KEYWORD


expression : COMMAND_RANDOM '(' expression ')'
```

Tube IC

Scaler ones:

`val_copy` s1 s2	s2 = s1
`add` s1 s2 s3	s3 = s1 + s2
`sub` s1 s2 s3	s3 = s1 - s2
`mult` s1 s2 s3	s3 = s1 * s2
`div` s1 s2 s3	s3 = s1 / s2
`test_less` s1 s2 s3	If (s1 < s2) set s3 to 1, else set s3 to 0.
`test_gtr` s1 s2 s3	If (s1 > s2) set s3 to 1, else set s3 to 0.
`test_equ` s1 s2 s3	If (s1 == s2) set s3 to 1, else set s3 to 0.
`test_nequ` s1 s2 s3	If (s1 ! = s2) set s3 to 1, else set s3 to 0.
`test_gte` s1 s2 s3	If (s1 >= s2) set s3 to 1, else set s3 to 0.
`test_lte` s1 s2 s3	If (s1 <= s2) set s3 to 1, else set s3 to 0.
`jump` Lable	jump to the lable
`jump_if_0` s1 Lable	If s1 == 0, jump to Lable.
`jump_if_n0` s1 Lable	If s1 != 0, jump to Lable.
`random` s1 s2	s2 = a random integer x, where 0 <= x < s1.
`out_val` s1	Write a floating-point value of s1 to standard out.
`out_char` s1	Write s1 as charto standard out.

array ones:

`ar_get_idx` a1 s2 s3	In a1, find value at index s2, and put into s1.
`ar_set_idx` a1 s2 s3	In a1, set value at index s2 to the value s3
`ar_get_size` a1 s2	Calculate the size of a1 and put into s2.
`ar_set_size` a1 s2	Resize a1 to have s2 entries.
`ar_copy` a1 a2	Duplicate all values within a1 into a2.

Tube AC

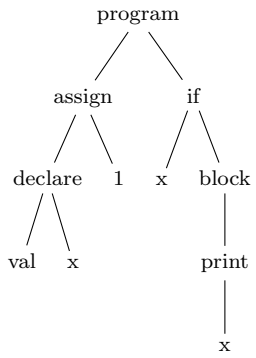
- There are no scalar or array variables.
- There are eight registers called regA, regB, regC, regD, regE, regF, regG, and regH. These are identical to scalar variables, but you have a limited number of them.
- There are no array-based instructions so you must find replacements for the array instructions.

Flow Control exampls

using them jumps

IF example

```
val x = 1;
if(x) {
    print(x);
}
```

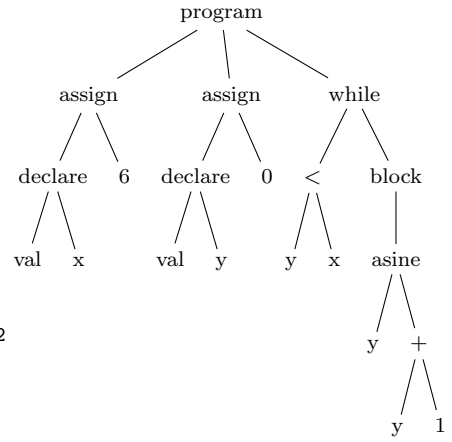


IC:

```
val_copy 1 s2
val_copy s2 s1
jump_if_0 s1 if_1
out_val s1
out_char '\n'
if_1:
```

WHILE

```
val x = 6;
val y = 0;
while(y < x) {
    y += 1;
}
```



IC:

```
val_copy 6 s2
val_copy s2 s1
val_copy 0 s4
val_copy s4 s3
start_1:
test_less s3 s1 s5
jump_if_0 s5 end_2
val_copy 1 s6
add s3 s6 s7
val_copy s7 s3
jump start_1
end_2:
```

Assembly inst

arry IC to AC code

IC:`val_copy` 5 s1 **AC:**`val_copy` 5 regA
`store` regA 1

IC:`ar_get_size` 1 s1 s6
AC:`load` 2 regA
`load` regA regB
`store` regB 6

IC:`ar_get_idx` s1 s6 s2
AC:`load` 2 regA
`add` 1 regA regA
`add` regA regB regA
`mem_copy` regA 1

IC:`ar_set_idx` s1 s7
AC:`load` 2 regA
`val_copy` 0 regB
`load` 4 regC
`add` 1 regA regA
`add` regA regB regA
`store` regC regA

IC:`ar_set_size` s1 s7
AC:`load` 2 regA
`val_copy` 3 regB
`load` 0 regC
`store` regB regC
`add` 1 regC regD
`add` regD regB regD
`store` regD 0
`store` regC 2
`load` regA regE
`test_less` regB regE regF
`jump_if_0` regF `bigger_1`
`val_copy` regB regF
`jump end_size_2`
`bigger_1: val_copy` regE regF
`end_size_2:`
`while_st_3:`
`add` 1 regA regA
`add` 1 regC regC
`mem_copy` regA 1
`sub` regF 1 regF
`jump_if_0` regF `while_end_4`
`jump while_st_3`
`while_end_4:`

memory

L	S	F
0	106	111
1	4	
2	100	107
3	105	
4	3	
5	103	
6		
7		
8		
9		
⋮	⋮	
100	0	
101	1	
102	'a'	
103	2	
104	'b'	
105	'c'	
106	0	3
107		'a'
108		'b'
109		
111		
112		
113		