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	$S \to aSa$
Non-terminal Symbols = syntactic variables	$S \to T$
Start Symbol S = special non-terminal	$T \to bSb$
Production Rules of the form LHS → RHS	$T\epsilon$

- 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

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

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 ’,’ expression

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

`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.

Flow Control exampls

using them jumps