Compiler Construction

Compiler Project: Specification Document

Team Members:

Huzefa Saifuddin, 22K-5125
Baasim Ahmed, 22K-5029

Section:

BCS-7E

Lexical Rules

letter	::=	a | b | ... | z | A | B | ... | Z
digit	::=	0 | 1 | ... | 9
id	::=	letter { letter | digit | _ }
intcon	::=	digit { digit }
realcon	::=	intcon.intcon
charcon	::=	'ch' | '\n' | '\0', where ch denotes any printable ASCII character, as specified by the C function isprint(), except for \ (backslash) and ' (apostrophe).
stringcon	::=	"{ch}", where ch denotes any printable ASCII character, as specified by the C function isprint(), except for " (quotes) and the newline character.
comment		Comments are like in C, i.e., a sequence of characters preceded by /* and followed by */, which contains no occurrence of */.

Syntactic Rules

Grammar Production Rules

```
prog        ::= { decl ';' | func }
decl        ::= type decl_var { ',' decl_var }
              | type id '(' param_types ')' '{' { type decl_var { ';' decl_var } }
              { cmd } '}'
              | void id '(' param_types ')' '{' { type decl_var { ';' decl_var } }
              { cmd } '}'
decl_var    ::= id [ '[' intcon ']' ]
type        ::= char
              | int
              | float
              | bool
param_types ::= type (id | &id | id '[' ']') { ',' type (id | &id | id '[' ']') }
func        ::= type id '(' param_types ')' '{' { type decl_var { ';' decl_var } }
              { cmd } '}'
              | void id '(' param_types ')' '{' { type decl_var { ';' decl_var } }
              { cmd } '}'
cmd         ::= if '(' expr ')' cmd [ else cmd ]
              | while '(' expr ')' cmd
              | for '(' [ atrib ] ';' [ expr ] ';' [ atrib ] ')' cmd
              | return [ expr ] ';'
              | atrib ';'
              | id '(' [ expr { ',' expr } ] ')' ';'
              | '{' { cmd } '}'
              | ';'
attrib      ::= id [ '[' expr ']' ] = expr
expr        ::= expr_simp [ op_rel expr_simp ]
expr_simp   ::= [+ | - ] termo { (+ | - ||) termo }
termo       ::= factor { (* | / | &&) factor }
factor      ::= id [ '[' expr ']' ] | intcon | realcon | charcon |
              id '(' [ expr { ',' expr } ] ')' | '(' expr ')' | '!' factor
op_rel      ::= ==
              | !=
              | <=
              | <
              | >=
              | >
```

Associativity and Operator Precedence

Operator	Associativity
!, – (unary)	right to left
*, /	left to right
+, – (binary)	left to right
<, <=, >, >=	left to right
==, !=	left to right
&&	left to right
||	left to right