Sonoma State University

Department of Computer Science

CS-460: Programming Languages

BNF Language Definition

A C-like programming language in Backus-Naur Form:

<CHARACTER> ::= | ! | # | $ | % | & | (|) | * | + | , | - | . | / | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | [|] | ^ | _ | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z | { | | | } | ~

<ESCAPED_CHARACTER> ::= \a | \b | \f | \n | \r | \t | \v | \\ | \? | \' | \" | \x<HEX_DIGIT> | \x<HEX_DIGIT><HEX_DIGIT>

<LETTER> ::= A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z

<DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<HEX_DIGIT> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F | a | b | c | d | e | f

<L_PAREN> ::= (

<R_PAREN> ::=)

<L_BRACKET> ::= [

<R_BRACKET> ::=]

<L_BRACE> ::= {

<R_BRACE> ::= }

<DOUBLE_QUOTE> ::= "

<SINGLE_QUOTE> ::= '

<SEMICOLON> ::= ;

<COMMA> ::= ,

<ASSIGNMENT_OPERATOR> ::= =

<PLUS> ::= +

<MINUS> ::= -

<ASTERISK> ::= *

<DIVIDE> ::= \

<MODULO> ::= %

<CARET> ::= ^

<LT> ::= <

<GT> ::= >

<LT_EQUAL> ::= <=

<GT_EQUAL> ::= >=

<BOOLEAN_AND> ::= &&

<BOOLEAN_OR> ::= ||

<BOOLEAN_NOT> ::= !

<BOOLEAN_EQUAL> ::= ==

<BOOLEAN_NOT_EQUAL> ::= !=

<BOOLEAN_TRUE> ::= TRUE

<BOOLEAN_FALSE> ::= FALSE

<STRING> ::= <CHARACTER | <ESCAPED_CHARACTER> | <CHARACTER> <STRING> | <ESCAPED_CHARACTER> <STRING>

<DOUBLE_QUOTED_STRING> ::= <DOUBLE_QUOTE> <STRING> <DOUBLE_QUOTE>

<SINGLE_QUOTED_STRING> ::= <SINGLE_QUOTE> <STRING> <SINGLE_QUOTE>

<LETTER_UNDERSCORE> ::= <LETTER> | _

<LETTER_DIGIT_UNDERSCORE> ::= <LETTER> | <DIGIT> | _

<WHOLE_NUMBER> ::= <DIGIT> | <DIGIT> <WHOLE_NUMBER>

<INTEGER> ::= <WHOLE_NUMBER> | <PLUS> <WHOLE_NUMBER> | <MINUS> <WHOLE_NUMBER>

<IDENTIFIER> ::= <LETTER_UNDERSCORE> | <LETTER_UNDERSCORE> <LETTER_DIGIT_UNDERSCORE> | <LETTER_UNDERSCORE> <LETTER_DIGIT_UNDERSCORE> <IDENTIFIER>

<IDENTIFIER_LIST> ::= <IDENTIFIER> | <IDENTIFIER> <COMMA> | <IDENTIFIER_LIST>

<IDENTIFIER_ARRAY_LIST> ::= <IDENTIFIER> <L_BRACKET> <WHOLE_NUMBER> <R_BRACKET> | <IDENTIFIER> <L_BRACKET> <WHOLE_NUMBER> <R_BRACKET> <COMMA> <IDENTIFIER_ARRAY_LIST>

<IDENTIFIER_AND_IDENTIFIER_ARRAY_LIST> ::= <IDENTIFIER_LIST> | <IDENTIFIER_ARRAY_LIST> | <IDENTIFIER_LIST> <IDENTIFIER_ARRAY_LIST> | <IDENTIFIER_ARRAY_LIST> <IDENTIFIER_LIST>

<DATATYPE_SPECIFIER> ::= char | bool | int

<NUMERICAL_OPERAND> ::= <IDENTIFIER> | <INTEGER> | <GETCHAR_FUNCTION> | <USER_DEFINED_FUNCTION> | <SINGLE_QUOTE> <CHARACTER> <SINGLE_QUOTE> | <SINGLE_QUOTE> <ESCAPED_CHARACTER> <SINGLE_QUOTE> | <DOUBLE_QUOTE> <CHARACTER> <DOUBLE_QUOTE> | <DOUBLE_QUOTE> <ESCAPED_CHARACTER> <DOUBLE_QUOTE>

<NUMERICAL_OPERATOR> ::= <PLUS> | <MINUS> | <ASTERISK> | <DIVIDE> | <MODULO> | <CARET>

<BOOLEAN_OPERATOR> ::= <BOOLEAN_AND_OPERATOR> | <BOOLEAN_OR_OPERATOR>

<EQUALITY_EXPRESSION> ::= <BOOLEAN_EQUAL> | <BOOLEAN_NOT_EQUAL>

<RELATIONAL_EXPRESSION> ::= <LT> | <LT_EQUAL> | <GT> | <GT_EQUAL> | <BOOLEAN_EQUAL> | <BOOLEAN_NOT_EQUAL>

<NUMERICAL_EXPRESSION> ::= <NUMERICAL_OPERAND> | <L_PAREN> <NUMERICAL_OPERAND> <R_PAREN> | <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <NUMERICAL_EXPRESSION> | <L_PAREN> <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <NUMERICAL_EXPRESSION> <R_PAREN> | <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <L_PAREN> <NUMERICAL_EXPRESSION> <R_PAREN> <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <NUMERICAL_EXPRESSION> | <L_PAREN> <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <NUMERICAL_EXPRESSION> <R_PAREN> | <NUMERICAL_OPERAND> <NUMERICAL_OPERATOR> <L_PAREN> <NUMERICAL_EXPRESSION> <R_PAREN>

<BOOLEAN_EXPRESSION> ::= <BOOLEAN_TRUE> | <BOOLEAN_FALSE> | <IDENTIFIER> | <IDENTIFIER> <BOOLEAN_OPERATOR> <BOOLEAN_EXPRESSION> | <L_PAREN> <IDENTIFIER> <BOOLEAN_OPERATOR> <BOOLEAN_EXPRESSION> <R_PAREN> | <NUMERICAL_EXPRESSION> <BOOLEAN_EQUAL> <NUMERICAL_EXPRESSION> | <NUMERICAL_EXPRESSION> <BOOLEAN_NOT_EQUAL> <NUMERICAL_EXPRESSION> | <NUMERICAL_EXPRESSION> <LT_EQUAL> <NUMERICAL_EXPRESSION> | <NUMERICAL_EXPRESSION> <GT_EQUAL> <NUMERICAL_EXPRESSION> | <NUMERICAL_EXPRESSION> <LT> <NUMERICAL_EXPRESSION> | <NUMERICAL_EXPRESSION> <GT> <NUMERICAL_EXPRESSION>

<INITIALIZATION_EXPRESSION> ::= <IDENTIFIER> <ASSIGNMENT_OPERATOR> <EXPRESSION> | <IDENTIFIER> <ASSIGNMENT_OPERATOR> <SINGLE_QUOTED_STRING> | <IDENTIFIER> <ASSIGNMENT_OPERATOR> <DOUBLE_QUOTED_STRING>

<EXPRESSION> ::= <BOOLEAN_EXPRESSION> | <NUMERICAL_EXPRESSION>

<SELECTION_STATEMENT> ::= if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <STATEMENT> | if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <STATEMENT> else <STATEMENT> | if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <BLOCK_STATEMENT> | if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <BLOCK_STATEMENT> else <STATEMENT> | if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <BLOCK_STATEMENT> else <BLOCK_STATEMENT> | if <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <STATEMENT> else <BLOCK_STATEMENT>

<ITERATION_STATEMENT> ::= for <L_PAREN> <INITIALIZATION_EXPRESSION> <SEMICOLON> <BOOLEAN_EXPRESSION> <SEMICOLON> <EXPRESSION> <R_PAREN> <STATEMENT> | for <L_PAREN> <INITIALIZATION_EXPRESSION> <SEMICOLON> <BOOLEAN_EXPRESSION> <SEMICOLON> <EXPRESSION> <R_PAREN> <BLOCK_STATEMENT> | while <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <STATEMENT> | while <L_PAREN> <BOOLEAN_EXPRESSION> <R_PAREN> <BLOCK_STATEMENT>

<ASSIGNMENT_STATEMENT> ::= <IDENTIFIER> <ASSIGNMENT_OPERATOR> <EXPRESSION> <SEMICOLON> | <IDENTIFIER> <ASSIGNMENT_OPERATOR> <SINGLE_QUOTED_STRING> <SEMICOLON> | <IDENTIFIER> <ASSIGNMENT_OPERATOR> <DOUBLE_QUOTED_STRING> <SEMICOLON>

<PRINTF_STATEMENT> ::= printf <L_PAREN> <DOUBLE_QUOTED_STRING> <R_PAREN> <SEMICOLON> | printf <L_PAREN> <SINGLE_QUOTED_STRING> <R_PAREN> <SEMICOLON> | printf <L_PAREN> <DOUBLE_QUOTED_STRING> <COMMA> <IDENTIFIER_AND_IDENTIFIER_ARRAY_LIST> <R_PAREN> <SEMICOLON> | printf <L_PAREN> <SINGLE_QUOTED_STRING> <COMMA> <IDENTIFIER_AND_IDENTIFIER_ARRAY_LIST> <R_PAREN> <SEMICOLON>

<GETCHAR_FUNCTION> ::= getchar <L_PAREN> <IDENTIFIER> <R_PAREN>

<USER_DEFINED_FUNCTION> ::= <IDENTIFIER> <L_PAREN> <IDENTIFIER_AND_IDENTIFIER_ARRAY_LIST> <R_PAREN> | <IDENTIFIER> <L_PAREN> <EXPRESSION> <R_PAREN>

<DECLARATION_STATEMENT> ::= <DATATYPE_SPECIFIER> <IDENTIFIER> <SEMICOLON> | <DATATYPE_SPECIFIER> <IDENTIFIER_AND_IDENTIFIER_ARRAY_LIST> <SEMICOLON>

<RETURN_STATEMENT> ::= return <EXPRESSION> <SEMICOLON> | return <SINGLE_QUOTED_STRING> <SEMICOLON> | return <DOUBLE_QUOTED_STRING> <SEMICOLON>

<STATEMENT> ::= <DECLARATION_STATEMENT> | <ASSIGNMENT_STATEMENT> | <ITERATION_STATEMENT> | <SELECTION_STATEMENT> | <PRINTF_STATEMENT> | <RETURN_STATEMENT>

<COMPOUND_STATEMENT> ::= <STATEMENT> | <STATEMENT> <COMPOUND_STATEMENT>

<BLOCK_STATEMENT> ::= <L_BRACE> <COMPOUND_STATEMENT> <R_BRACE>

```
<PARAMETER_LIST> ::= <DATATYPE_SPECIFIER> <IDENTIFIER> |
   <DATATYPE_SPECIFIER> <IDENTIFIER> <PARAMETER_LIST>

<FUNCTION_DECLARATION> ::= function <DATATYPE_SPECIFIER> <IDENTIFIER>
   <L_PAREN> <PARAMETER_LIST> <R_PAREN> < L_BRACE> <COMPOUND_STATEMENT>
   <R_BRACE> | function <DATATYPE_SPECIFIER> <IDENTIFIER> <L_PAREN> void
   <R_PAREN> < L_BRACE> <COMPOUND_STATEMENT> <R_BRACE>

<PROCEDURE_DECLARATION> ::= procedure <IDENTIFIER> <L_PAREN>
   <PARAMETER_LIST> <R_PAREN> < L_BRACE> <COMPOUND_STATEMENT> <R_BRACE>
   | procedure <IDENTIFIER> <L_PAREN> void <R_PAREN> < L_BRACE>
   <COMPOUND_STATEMENT> <R_BRACE>

<MAIN_PROCEDURE> ::= procedure main <L_PAREN> void <R_PAREN>
   <BLOCK_STATEMENT>

<PROGRAM> ::= <MAIN_PROCEDURE> | <FUNCTION_DECLARATION> <PROGRAM> |
   <PROCEDURE_DECLARATION> <PROGRAM> | <DECLARATION_STATEMENT> <PROGRAM>
```

The language contains the following datatypes:

- **char** : holds one character. Strings are implemented by defining an array of char of a given size using an array element. For example, char my_string[256] would enable one to store strings up to 256 bytes in length (accessed as 0 to 255 in the indices).
- **bool** : holds the Boolean value TRUE or FALSE.
- **int** : holds a 32-bit signed integer.

Your language must support the following statements:

- Declaration statement.
- Assignment statement.

Selection statement: if-then-else.

- Iteration statements: for and while.

This language has the following built-in input-output subroutines:

- **getchar()** : reads one character cast as integer from standard input (keyboard). If no character was read from keyboard, -1 is returned.
- **printf()** : outputs a formatted string to the screen. Example: printf ("The magic number is %d\n", number);

A program must minimally contain the following:

- A procedure named "main".
- The main procedure must contain no input parameters. Example:

```
procedure main (void) {}
```

Rules for passing arrays to functions or procedures:

- Since the language does not support array pointers, arrays of all datatypes are pass-by-value rather than pass-by-reference.

Examples of passing string variable, char my_string[255] to a function or procedure:

- `my_string_function (my_string)` : This will pass the entire 255-byte string to the function(). This function or procedure should be declared to accept at least 255 bytes!
- `my_string_function (my_string[0])` : This will only send one byte to the function.
- `my_string_function (my_string[12])` : This will only send one byte to the function.