# 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.