<SPLProgram> ::= ((<function definitions>))\* <program definition> EOPC

end potato.

<datatype> ::= ( int | flt | bool | char )

<literal> ::= <integer> | <float> | <character> | <boolean> | <string>

<integer> ::= <digit> { <digit> }\*

<float> ::= <digit> { <digit> }\* . <digit> { <digit> }\* [ E [ - ] <digit> { <digit> }\* ]

<character> ::= '<ASCIICharacter>'

<boolean> ::= true | false | TRUE | FALSE

<identifier> ::= <letter> { (( <letter> | <digit> | \_ )) }\*

<UB> ::= [ (( + | - )) ] <integer>

<variable definitions> ::= <datatype> <identifier> [ <UB> ] {, {<datatype> <identifier> [ <UB> ]}\* .

<constant definitions> ::= <datatype> <identifier> [ <UB> ] CONST .

<ASCIICharacter> ::= || Every printable ASCII character in range [ ' ','~' ]

<string> ::= "{<ASCIICharacter>}\*" | '{<ASCIICharacter>}\*'

<letter> ::= A | B | ... | Z | a | b | ... | z

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

<negation> ::= [ NOT ] <comparison>

<conjunction> ::= <negation> { (( AND | NAND )) <negation> }\*

<expression> ::= <conjunction> { (( OR | NOR | XOR )) <conjunction> }

<comparison> ::= <comparator> [ (( < | <= | == | > | >= | (( != | <> )) )) <comparator> ]

<comparator> ::= <term> { (( + | - )) <term> }\*

<comment> ::= || { <ASCIICharacter> }\* EOLC || single-line comment

| /\* { (( <ASCIICharacter> | EOLC )) }\* \*/ || multi-line (block) comment

<formalParameter> ::= [ (( IN | OUT | IO | REF )) ] <identifier> : <datatype> [ [ { , }\* ] ]

<program definition> ::= <datatype> main potato [<datatype> : <indentifier>]

{ <statement> }\*

cookedpotato.

<function definition> ::= <datatype> <identifier> [<datatype> : <indentifier>]

{ <statement> }\*

cookedpotato.

<handler definition> ::= <datatype> <identifier> [<datatype> : <indentifier>]

{ <statement> }\*

cookedpotato.

<exit statement> ::= cookedpotato.

<assertion> :: { <expression> }

<statement> ::= <print statement> | <expression> | <comment> | <input statment>

| <if statement> | <do while statement> | <for statement>

| <assignment statement> | <assertion>

<assignment statement> ::= <indentifier> { , <idnetifier> }\* = <expression> .

<print statement> ::= outpotato (( <string> | <expression> | \n | \t )) { ,

{( <string> | <expression> | \n | \t )) }\* .

<input statement> ::= inpotato [ <string> ] <variable> .

<if statement> ::= if ( <expression> )

{ <statement> }\*

{elif ( <expression> )

{ <statement> }\* }\*

else

{ <statement> }\*

cookedpotato. |

if ( <expression> )

{ <statement> }\*

cookedpotato.

<dowhile statement> ::= dowhile ( <expression> )

{ <statement> }\*

cookedpotato.

<for statement> ::= for ((<datatype>)) <indentifier> = <expression> . <expression> . <increment expression>

{ <statement> }\*

cookedpotato.

<return statement> ::= return potato [ ( <expression> ) ] .

SYNTAX DECRIPTION

POTATO programming language is a free-format language, a case-neutral programming language, and uses the ASCII character set.

Normal statements are not terminated by a period **BUT** functional statements are terminated by the reserve word cookedpotato.

A single-line **<comment>** may be appended to the end of any source line and treated line white space

<comment> ::= || { <ASCIICharacter> }\* EOLC || single-line comment

| /\* { (( <ASCIICharacter> | EOLC )) }\* \*/ || multi-line (block) comment

For example,

outpotato "Potato is my middle name" || print "Potato... etc"

A multi-line (block) **<comment>** may appear any place white-space may appear.

For example,

/\*

Pota-TOES... w e i r d... am I right?

\*/

outpotato "Potato is my middle name" /\* print

"Potato... etc" \*/

An **<identifier>** will eventually be used to name various syntactic entities.

<identifier> ::= (( <letter> | \_ )) { (( <letter> | <digit> | \_ )) }\*

A **<SPLProgram>** is a single program module definition. The flow-of-control begins with the first statement in the program module’s list of statements and continues until it terminates with an <exit statement>.

<SPLProgram> ::= ((<function definitions>))\* <program definition> EOPC

cookedpotato.

A **<program definition>** defines the main program module by providing (1) an optional data definitions section for local

variables and constants; and (2) a set of 0 or more executable statements that make up the program module body.

<program definition> ::= <datatype> main potato [<datatype> : <indentifier>]

{ <statement> }\*

cookedpotato.

A program module body consists of 0 or more statements that begin execution when the program is run. Flow-of-control begins with the

first statement in the program module’s body; flow-of-control terminates after the last statement is executed. There is no explicit instruction used

to terminate program module execution; instead, flow-of-control terminates when the flow-of-control “runs into” the program module cookedpotato.

reserved word.

<statement> ::= <print statement> | <expression> | <comment> | <input statment>

| <if statement> | <do while statement> | <for statement>

| <assignment statement> | <assertion>

The <PRINTStatement> adds string literals to the console output buffer using only the minimum number of characters required to

format the value being output. Notes (1) Output of ENDL causes the output buffer to be displayed as a 1 line on the user’s terminal and then

empties the output buffer (ENDL is the only way to add the end-of-line character to the output buffer); and (2) SPL does not allow for detailed

formatting of value added to the output buffer.

<PRINTStatement> ::= PRINT (( <string> | ENDL )) || \*\*\*SIMPLIFIED\*\*\*

{ , (( <string> | ENDL )) }\* .