Simple Programming Language (SPL)

SER 502 - ASU

Team 4 Members:

Dinaker Prakash Kolipaka

Manju Bisht

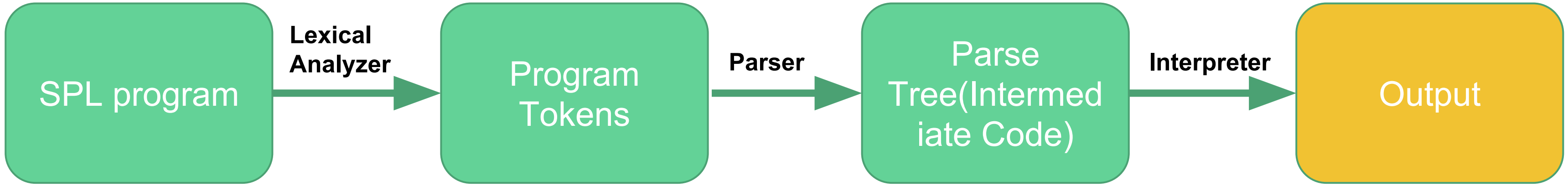
Ramya Varakantham

Vidhi Patel

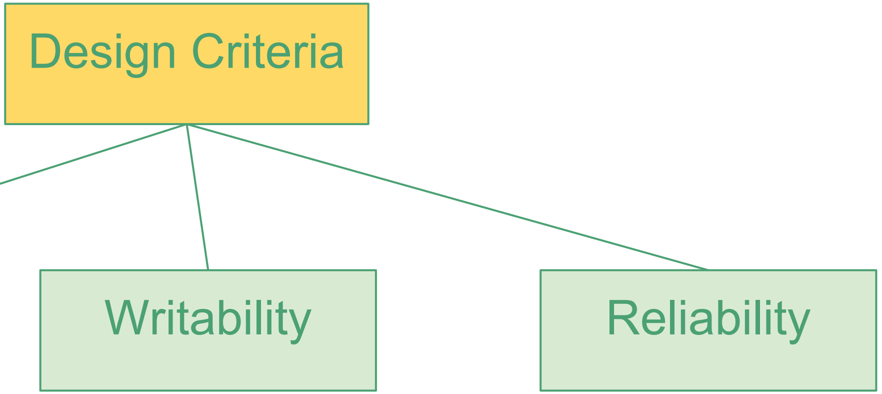
Agenda

SPL Introduction

- Stands for **Simple Programming Language**
- Aim of the project is **Design of SPL** and its **Compiler Construction**
- Design is derived from existing languages for **adaptability**
- Language written completely in **Prolog**



Language Design Criteria



The quality of a language that enables the reader (even non-programmers) to understand the nature of the computation or algorithm.

```
int x = 10;
x = x + 20;
print x;
```

This is the quality of expressivity in a language. Writability should be clear, concise, quick and correct

```
If ( true ) then {
...};
while ( true ) {
...};
```

Assurance that a program does not behave unexpectedly.

Error checking

Grammar Specification

1. CFG(Context-free Grammar) for the language. Tokens are:
   a. Terminals
   b. Non-terminals
2. 22 rules for grammar
3. Each of these rules are followed in the Parser.

1) ***Program → Statement-List***

2) ***Statement-List→Statement ';' Statement-List|Statement';'***

3) ***Statement→Assignment | Declaration | If-Statement | While-Statement | Print-Statement***

4) ***Assignment→Identifier '=' Expression | Identifier '=' Boolean | Identifier '=' Comparison***

5) .

6) .

7) ***If-Statement → 'if' '(' Condition ')' 'then' Block| 'if''('Condition')' then' Block 'else' Block***

8) ***Letter→ 'a' | 'b' | 'c' |...|'z'| 'A' | 'B' | 'C' |...| 'Z'***

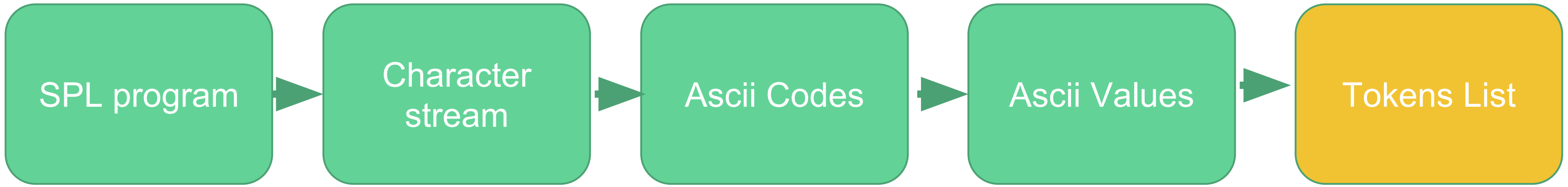
9) ***Boolean → 'true' | 'false'***

How to write an SPL program

- All statements end with a semi-colon (;).
- Two datatypes: int and bool.
- Statements are of types:
  - **Assignment : a = b; a = 1 + b; (**Should be same datatype**)**
  - **Declaration: int a; bool b = true; (**Types cannot change later in the program**)**
  - **Conditional: a = b > c;**
  - **Evaluation: temp = b - 3 + 4 / e;**
  - **Print: print a; print a + b ;**
  - **Comments: # This is a comment**
- Each of the words in the program should be separated by a space:
  - **Int a = b + 2 * c;**
- Loops:
  - **If-then-else: if (true) { print a; } else { print b ; };**
  - **While: while(a > b) { a = b + 3 ; };**

Lexical Analysis

- Read the SPL program.
- Convert into Tokens.
- Generate a List of Tokens.



Lexical Analysis

```
bool b = true;
int num = 10;
if ( b )
then
{
print num / 10;
}
else
{
print num;
};
while ( num < 10 )
{
num = num + 1;
};
```

Token List =

[bool, b, =, true, ;, int, num, =, 8, ;, int, num2, ;, if, '(', b, ')', then, '{', print, num, /, 10, ;, '}', else, '{', print, num2, ;, '}', ;, while, '(', num, <, 10, ')', '{', num, =, num, +, 1, ;, '}', ;]

- *Every token is separated by comma (,).*
- *Statements are separated by semicolon (;).*
- *Special characters are not allowed in the program.*
- *Valid tokens consist of terminals defined in the grammar, english alphabet, integers, mathematical operators, brackets ((), {}).*
- *Output is a list of tokens.*

Parser

- Parser is a program that takes a list of tokens generated by the lexical analyzer and generates a parse tree.
- In SPL the parse tree generated is a dense parse tree with nodes like “program”, “statement_List”, “statement”, “assign” etc.
- The rules for SPL parser are written in DCG (Definite Clause Grammar).
- Since we define a set of rules for parsing, any syntax errors in the program result in false.
- For SPL the parse tree is the intermediate code.

Token List:

```
[int,a,=,1,;,print,a,;]
```

Parse Tree:

```
program(statement_List(statement(declar
e(int(id(a)), term(id(1)))),
statement_List(statement(print(expressio
n(term(id(a))))))))
```

Token List:

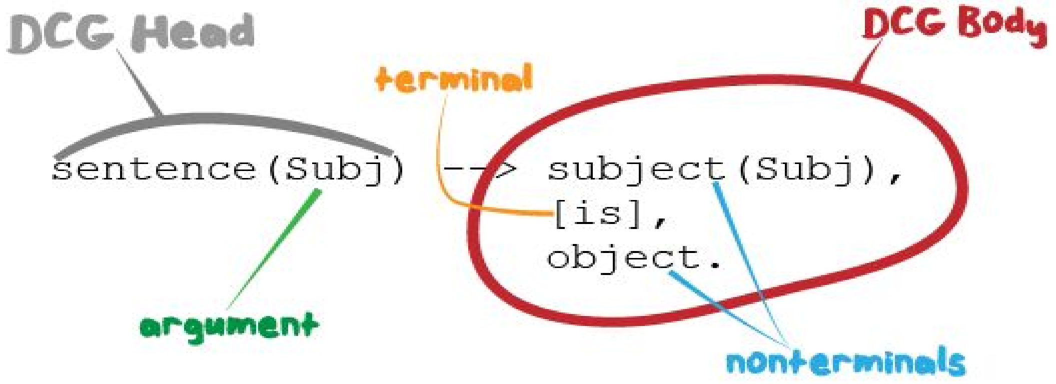
```
[a,=,a,*,2,+,3,;]
```

Parse Tree:

```
program(statement_List(statement(assig
n(id(a), expression(multiply(term(id(a)),
expression(add(term(id(2)),
expression(term(id(3))))))))))
```

Parser - DCG Rules

- A Prolog definite clause grammar (DCG) describes a Prolog list.
- A DCG rule has the form : head --> body.



```
sentence(Subj) --> subject(Subj),
                   [is],
                   object.
```

Example CFG:

Statement-List →
Statement‘;’Statement-List
| Statement ‘;’

DCG for the above CFG:

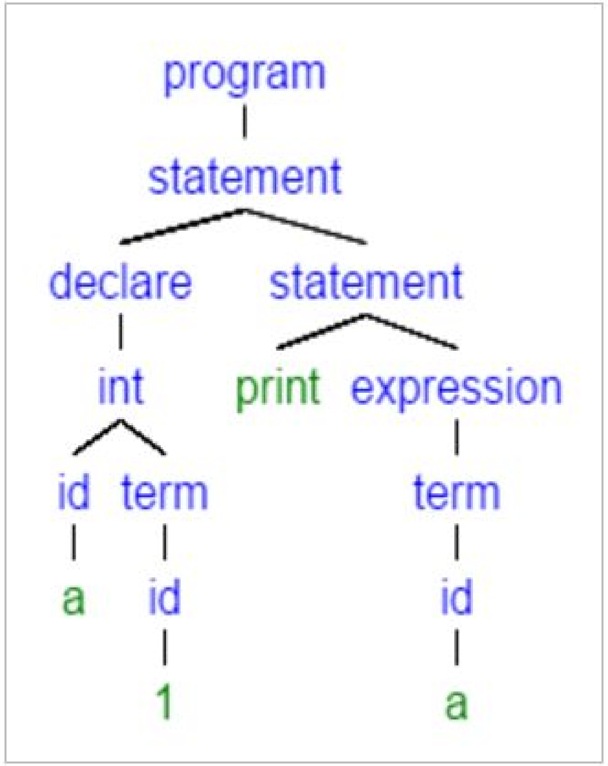
statement_list('statement_List'(S,L))-->
statement(S),[';'],statement_list(L).

statement_list('statement_List'(S))-->
statement(S),[';'],!.

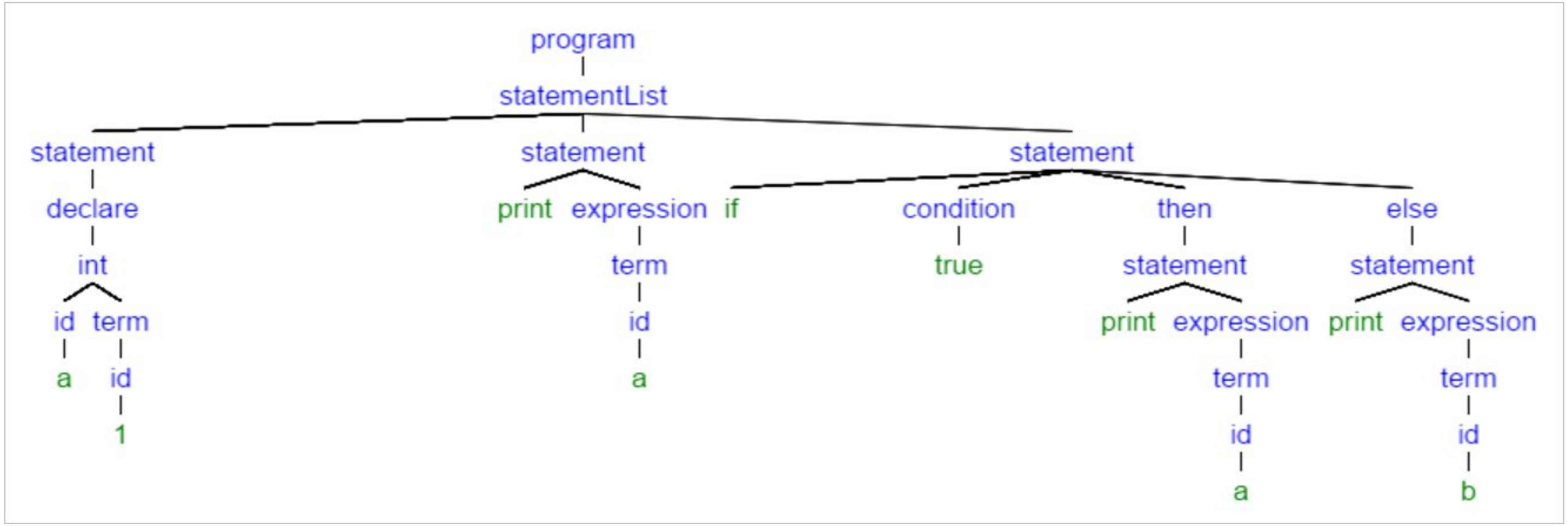
Parse Tree - 1

program(statement_List(statement(declare(int(id(a)), term(id(1)))), statement_List(statement(print(expression(term(id(a))))))))

- Here the leaf nodes are either identifiers or numbers.
- The internal nodes are non-terminals like statement, expression, etc.



Parse Tree -2



program(statement_List(statement(declare(int(id(a)), term(id(1)))), statement_List(statement(print(expression(term(id(a))))), statement_List(statement(if(condition(true), then(statement_List(statement(print(expression(term(id(a))))))), else(statement_List(statement(print(expression(term(id(b))))))))))))

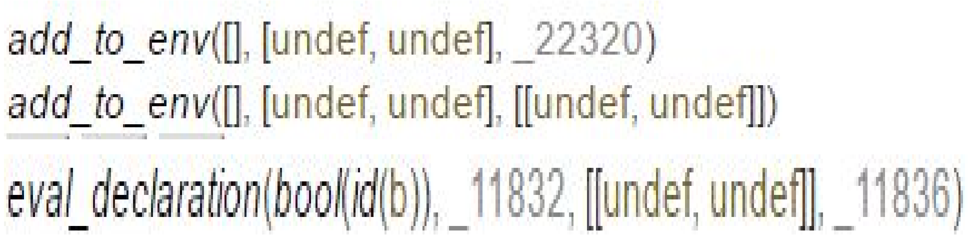
Interpreter

- Semantics provide more mathematical description of program behavior.
- Interpreter is formally defined using Operational semantics, where in we describe actions in terms of operators for a reduction machine.
- Interpreter takes as input the parse tree and evaluates it to generate output.
- Reduction rules are defined in Prolog and are based on previously defined context-free grammar rules.
- Environment is implemented using list of lists and is passed with every reduction rule.

Eg: reduce_Statement_List(statement_List(T, E), Env, Env_New) :-
reduce_Statement(T, Env, Env_N),
reduce_Statement_List(E, Env_N, Env_New).

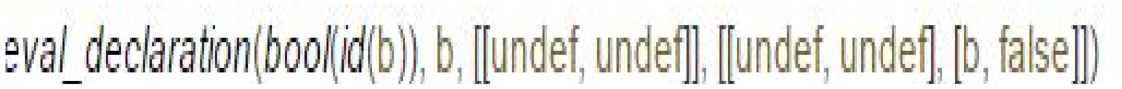
Interpreter

- Environment is updated for every declaration and assignment.



```
add_to_env([], [undef, undef], _22320)
add_to_env([], [undef, undef], [[undef, undef]])
eval_declaration(bool(id(b)), _11832, [[undef, undef]], _11836)
```

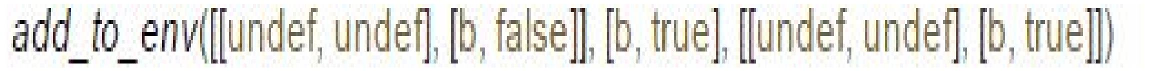
- Default values are provided for identifiers at the time of declaration.
- Default value for identifier of type integer is ‘0’ and for type boolean is ‘false’.



```
eval_declaration(bool(id(b)), b, [[undef, undef]], [[undef, undef], [b, false]])
```

Interpreter

- During assignment, environment would be updated with the assigned value for the corresponding identifier.



```
add_to_env([[undef, undef], [b, false]], [b, true], [[undef, undef], [b, true]])
```

- We have provided run-time error checking for the following:
  a. x = true; (Error: Boolean variable not declared yet)
  b. y = 10; (Error: Integer variable not declared yet)
  c. int 5 = 6; (Error: Value cannot be assigned to an integer)
  d. bool b = 5; (Error: type mismatch)
  e. int 10; (Error: Declaration cannot take integer value)
  f. x / 0 ; (Error: Division by zero)

Language Specification

- Imperative based language.
- Supports basic mathematics operations.
- Support integer and boolean data types.
- Supports if-then-else condition.
- Supports while loop.
- Variables should be written with first letter small.
- Each word, operator, bracket should have space between them.
- Each statement should end with a semicolon.
- Default value for integer variable is 0.
- Default value for boolean is false.

Language Specification

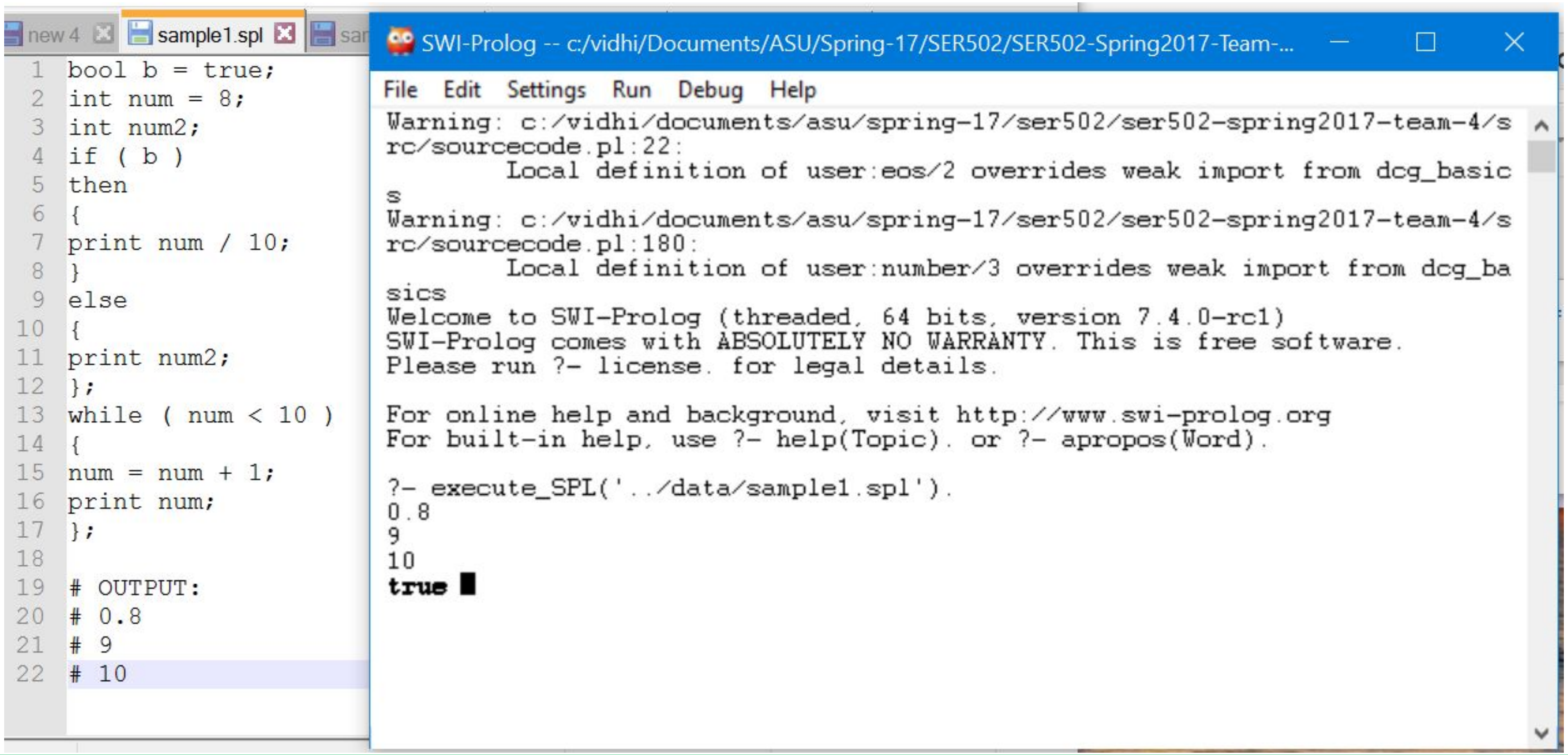
- There should be no extra spaces or tabs in the program.
- Error handling - type mismatch.
- Error handling - syntax error.
- Error handling - variable not defined.
- Error handling - identifiers cannot be keywords.
- Error handling - identifiers cannot be integers.
- Language can be extended to accommodate more data types and functionalities.

Execution and Sample Programs - 1

```
bool b = true;
int num = 10;
if ( b )
then
{
print num / 10;
}
else
{
print num;
};
while ( num < 10 )
{
num = num + 1;
print num;
};
```

```
# Output:

# 0.8
# 9
# 10
```

Execution and Sample Programs - 1


new 4
sample1.spl
SWI-Prolog -- c:/vidhi/Documents/ASU/Spring-17/SER502/SER502-Spring2017-Team-...
File Edit Settings Run Debug Help

```
bool b = true;
int num = 8;
int num2;
if ( b )
then
{
print num / 10;
}
else
{
print num2;
};
while ( num < 10 )
{
num = num + 1;
print num;
};

# OUTPUT:
# 0.8
# 9
# 10
```

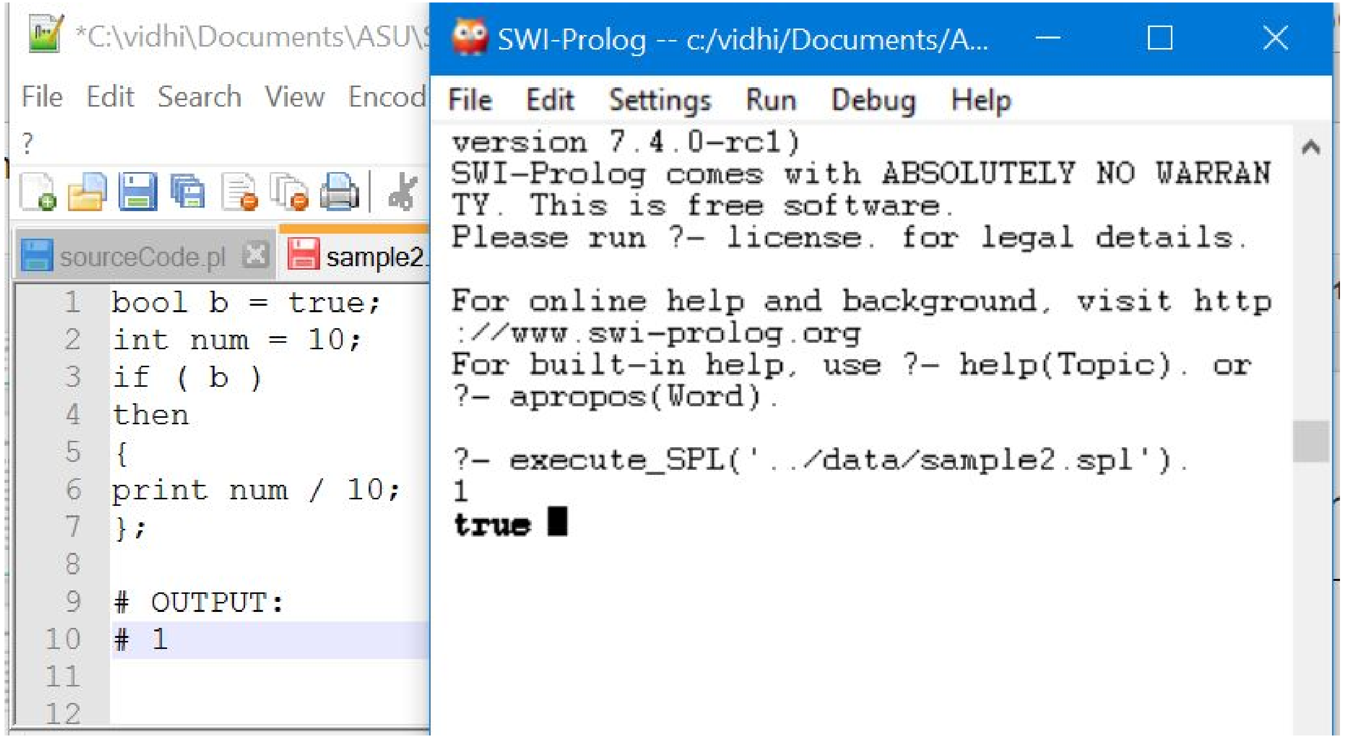
```
Warning: c:/vidhi/documents/asu/spring-17/ser502/ser502-spring2017-team-4/s
rc/sourcecode.pl:22:
        Local definition of user:eos/2 overrides weak import from dcg_basic
s
Warning: c:/vidhi/documents/asu/spring-17/ser502/ser502-spring2017-team-4/s
rc/sourcecode.pl:180:
        Local definition of user:number/3 overrides weak import from dcg_ba
sics
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- execute_SPL('../data/sample1.spl').
0.8
9
10
true
```

Execution and Sample Programs - 2

```
1.    bool b = true;
2.    int num = 10;
3.    if ( b )
4.    then
5.    {
6.    print num /
      10;
7.    };
```

# Output

1


*C:\vidhi\Documents\ASU\S
File Edit Search View Encod
?
sourceCode.pl
sample2.

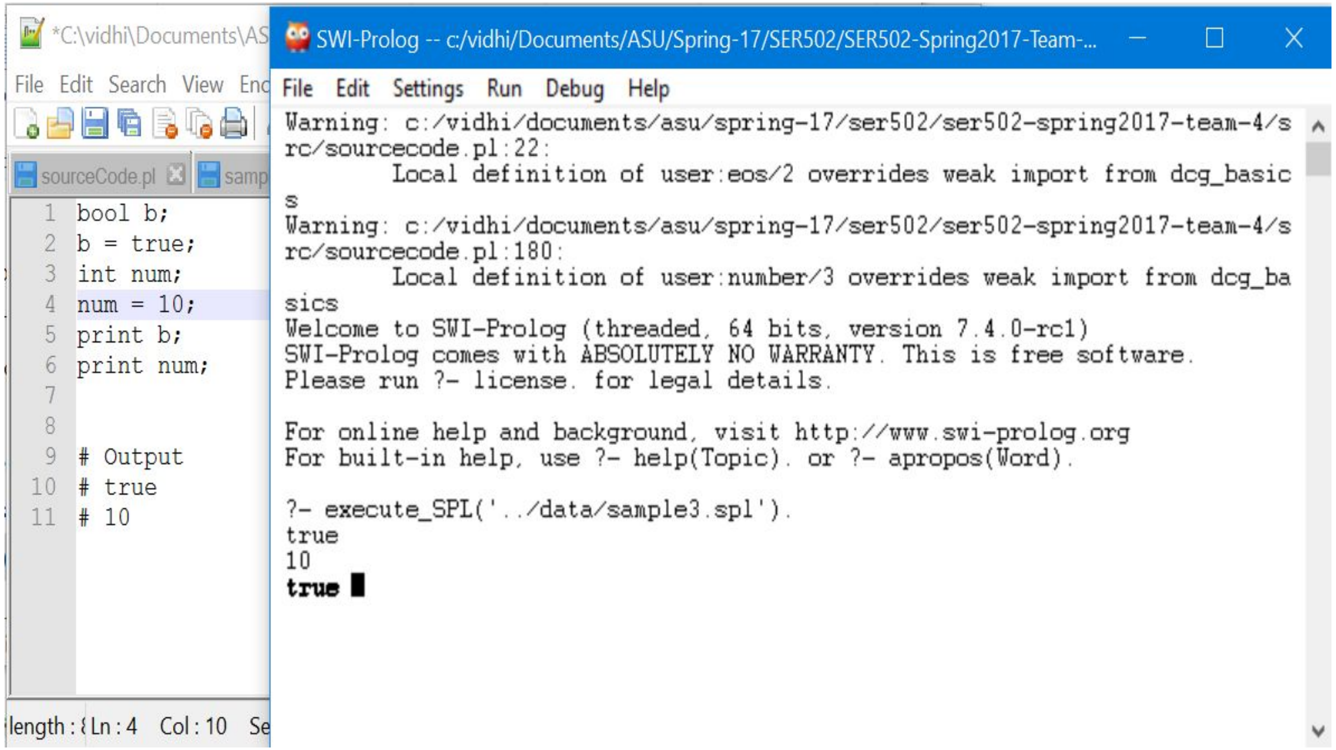
```
bool b = true;
int num = 10;
if ( b )
then
{
print num / 10;
};

# OUTPUT:
# 1
```

SWI-Prolog -- c:/vidhi/Documents/A...
File Edit Settings Run Debug Help

```
version 7.4.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRAN
TY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http
://www.swi-prolog.org
For built-in help, use ?- help(Topic). or
?- apropos(Word).

?- execute_SPL('../data/sample2.spl').
1
true
```

Execution and Sample Programs - 3

```
1.    bool b;
2.    b = true;
3.    int num;
4.    num = 10;
5.    print b;
6.    print num;


# Output:

# true
# 10
```


*C:\vidhi\Documents\AS
File Edit Search View Enc
sourceCode.pl
samp
length : { Ln : 4 Col : 10 Se
SWI-Prolog -- c:/vidhi/Documents/ASU/Spring-17/SER502/SER502-Spring2017-Team-...
File Edit Settings Run Debug Help

```
bool b;
b = true;
int num;
num = 10;
print b;
print num;


# Output
# true
# 10
```

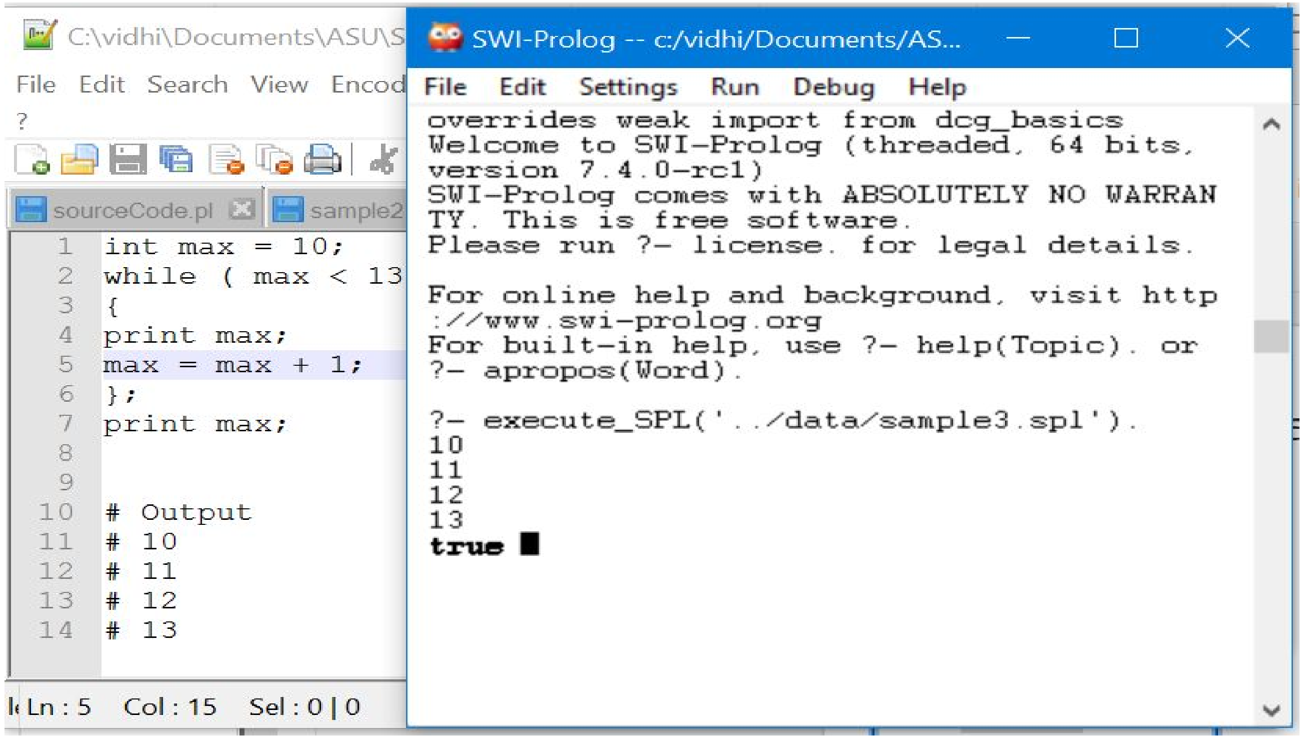
```
Warning: c:/vidhi/documents/asu/spring-17/ser502/ser502-spring2017-team-4/s
rc/sourcecode.pl:22:
        Local definition of user:eos/2 overrides weak import from dcg_basic
s
Warning: c:/vidhi/documents/asu/spring-17/ser502/ser502-spring2017-team-4/s
rc/sourcecode.pl:180:
        Local definition of user:number/3 overrides weak import from dcg_ba
sics
Welcome to SWI-Prolog (threaded, 64 bits, version 7.4.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- execute_SPL('../data/sample3.spl').
true
10
true
```

Execution and Sample Programs - 4

```
1.    int max = 10;
2.    while ( max < 13 )
3.    {
4.    print max;
5.    max = max + 1;
6.    };
7.    print max;
```

Output:

10
11
12
13



C:\vidhi\Documents\ASU\S
File Edit Search View Encod
?
sourceCode.pl
sample2
Ln : 5 Col : 15 Sel : 0 | 0
SWI-Prolog -- c:/vidhi/Documents/AS...
File Edit Settings Run Debug Help

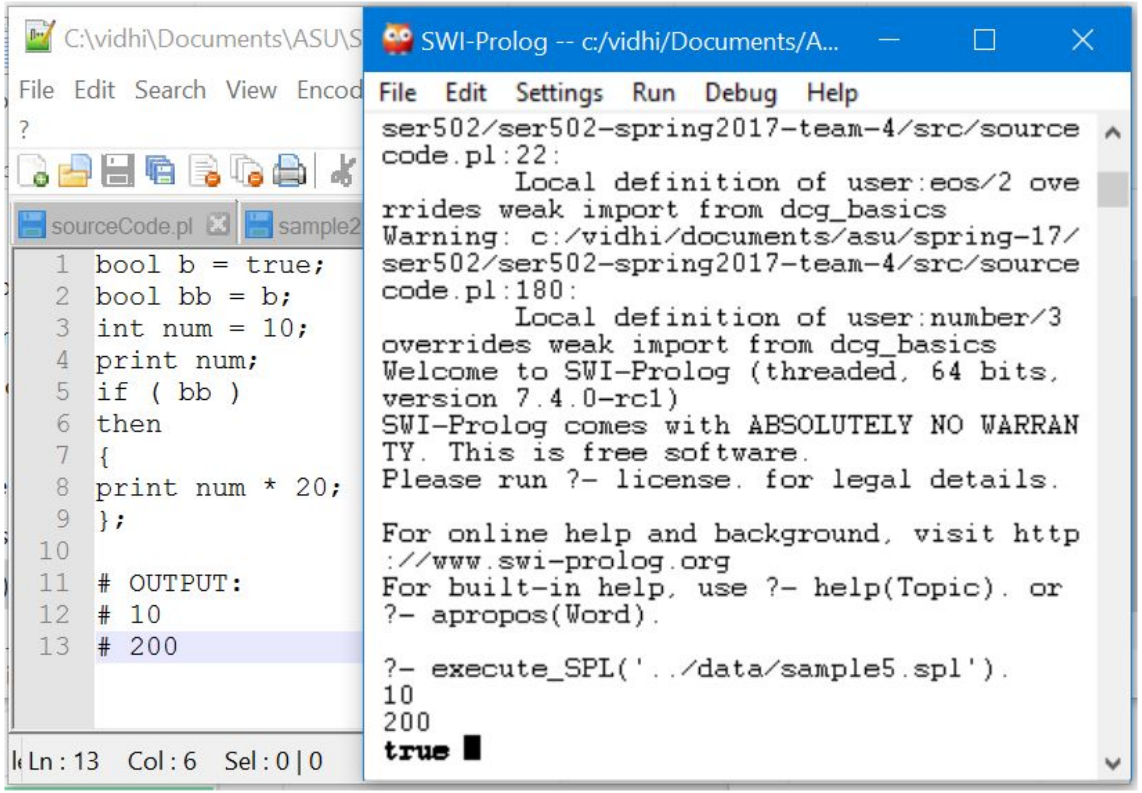
```
int max = 10;
while ( max < 13
{
print max;
max = max + 1;
};
print max;


# Output
# 10
# 11
# 12
# 13
```

```
overrides weak import from dcg_basics
Welcome to SWI-Prolog (threaded, 64 bits,
version 7.4.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRAN
TY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http
://www.swi-prolog.org
For built-in help, use ?- help(Topic). or
?- apropos(Word).

?- execute_SPL('../data/sample3.spl').
10
11
12
13
true
```

Execution and Sample Programs - 5

```
bool b = true;
bool bb = b;
int num = 10;
print num;
If ( bb )
then
{
print num * 20;
};
```

```
# Output:
# 10
# 200
```



C:\vidhi\Documents\ASU\S
File Edit Search View Encod
?
sourceCode.pl
sample2
Ln : 13 Col : 6 Sel : 0 | 0
SWI-Prolog -- c:/vidhi/Documents/A...
File Edit Settings Run Debug Help

```
bool b = true;
bool bb = b;
int num = 10;
print num;
if ( bb )
then
{
print num * 20;
};

# OUTPUT:
# 10
# 200
```

```
ser502/ser502-spring2017-team-4/src/source
code.pl:22:
        Local definition of user:eos/2 ove
rrides weak import from dcg_basics
Warning: c:/vidhi/documents/asu/spring-17/
ser502/ser502-spring2017-team-4/src/source
code.pl:180:
        Local definition of user:number/3
overrides weak import from dcg_basics
Welcome to SWI-Prolog (threaded, 64 bits,
version 7.4.0-rc1)
SWI-Prolog comes with ABSOLUTELY NO WARRAN
TY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http
://www.swi-prolog.org
For built-in help, use ?- help(Topic). or
?- apropos(Word).

?- execute_SPL('../data/sample5.spl').
10
200
true
```

Developer Perspective

- Prolog - We developed the complete language in Prolog as it was the major programming paradigm taught, and we decided to implement the knowledge gained in the semester in this project.
- Prolog proved to be an excellent choice to create the language SPL as it provided support for DCG, backtracking and methods were easy to write as predicates.
- Also, it proved possible to build the entire compiler purely in Prolog.
- For lexer, we chose list as an output data structure because it made the tokens processing and consuming by the parser easy and effective.
- For parser, we decided to go with the dense parse tree since it would be needed by the interpreter to understand the program structure.
- In interpreter, we used list of list as our environment data structure, again for the ease of use and process

Developer Perspective

- We chose imperative language as it is easy to write, understand, and read.
- Spaces between tokens for ease of read and and precessing.
- Semicolons at the end of each statement keeps the program structure neat.
- Program structure is inspired by C++.
- In order to understand the condition and statements of loops, we decided to use brackets for condition and parenthesis for statement block. Also, a semicolon at the end of the block would mean the loop has ended.

References

1) Toy compiler in Prolog- http://faculty.cooper.edu/smyth/cs225/ch7/prolog.htm
2) Definite Clause Grammar- http://www.amzi.com/manuals/amzi/pro/ref_dcg.htm
3) Read/Write files in Prolog- http://cs.union.edu/~striegnk/learn-prolog-now/html/node106.html#sec.l12.file.io
4) Interpreter- http://www.erlang.se/publications/prac_appl_prolog.pdf

5) Command line execution- http://www.swi-prolog.org/FAQ/WinExe.html, http://stackoverflow.com/questions/16301174/run-prolog-from-command-prompt-get-input-from-file-and-send-output-to-a-file, http://www.swi-prolog.org/pldoc/doc_for?object=section(2,%272.4%27,swi(%27/doc/Manual/cmdline.html%27))

6) DCG - http://www.pathwayslms.com/swipltuts/dcg/