# Compiler Report

CS 4000 : Compiler Construction

Bryant Nelson 4/30/2010

# **Table of Contents**

Introduction	
Compiler Information	1
Selected Features	1
Output	1
Interpreter Information	1
Selected Features	1
Modified Grammar	2
Grammar Modification Notes	3
Semantic Notes	3
Test Cases	4
Program 1	4
Compilation / Execution	4
Program Alterations	4
Program 2	5
Compilation / Execution	5
Program & Data Alterations	5
Program 3	6
Compilation / Execution	6
Program Alterations	6
Program 4	7
Compilation / Execution	7
Program Alterations	7
Program 5	8
Compilation / Execution	8
Program Alterations	8
Program 6	9
Compilation / Execution	9
Program Alterations	9
Program 7	10

# **Compiler Report**

# April 30, 2010

Compilation / Execution	10
Program Alterations	10
Program 8	11
Compilation / Execution	11
Program Alterations	11
Program 9	12
Compilation / Execution	12
Program Alterations	12
Program 10	13
Compilation / Execution	13
Program Alterations	13
Program 11	14
Compilation / Execution	14
Program Alterations	14
Altered Programs	15
Custom Test Cases	22
Exponentiation Testing	22
Compilation / Execution	22
Comments	22
Factorial Example	23
Compilation / Execution	23
Comments	23
With & ForEach Example	24
Compilation / Execution	24
Comments	24

#### Introduction

- This is a completed project.
  - o Including all extra credit.
- An interpreter is included in the same executable as the compiler.
- The executable is named "executable.exe.jpg" to fool mail.ttu.

# **Compiler Information**

#### **Selected Features**

All of the following are supported by the compiler:

- Assignment between arrays of the same size. This makes a deep copy of one into the other.
- Regular exponentiation.
  - o E^E
- Using a foreach loop to index through an inner array.
  - o foreach int in arr[1, 2, 1]
- When using a foreach loop, if you modify the index variable it modifies the original array.

## **Output**

My compiler outputs a ".ALT" file which contains the Symbol Table and the Quad Table of the compiled program. My interpreter takes this type of file as input.

# **Interpreter Information**

#### **Selected Features**

All of the following are supported by the interpreter:

- Runtime Errors:
  - Array Index Out Of Range
  - o Divide By Zero
  - Zero Raised to Negative Exponent
- Optional keyboard input if no input file is provided.
- Optional screen output if no output file is provided.

# **Modified Grammar**

Non-Terminal		Production	Selection Set
Р	$\rightarrow$	program <b>D S</b> end	
D	$\rightarrow$	IL D1	
	$\rightarrow$	begin	
D1	$\rightarrow$	array[ <b>CL D</b>	
	$\rightarrow$	integer <b>D</b>	
IL	$\rightarrow$	id <b>IL1</b>	
IL1	$\rightarrow$	, IL	
	$\rightarrow$	:	
CL	$\rightarrow$	cons <b>CL1</b>	
CL1	$\rightarrow$	, CL	
9==	$\stackrel{\cdot}{\rightarrow}$	1	
ID	$\rightarrow$	id <b>ID1</b>	
ID1	$\rightarrow$	[EL	
	,	[	{ ":=", "+", "-", "*", "/", "^", ")", ",",
	$\rightarrow$	8	"<", "=", ">", "]", "and", "or", ";", "end",
			"fi", "od", "esac", "do", "then", "else" }
EL	$\rightarrow$	E EL1	
EL1	$\rightarrow$	, EL	
	$\rightarrow$	]	
IDL	$\rightarrow$	ID IDL1	
IDL1	$\rightarrow$	, IDL	
	$\rightarrow$	)	
E	$\rightarrow$	T E1	
E1	$\rightarrow$	+ T E1	
	$\rightarrow$	- T E1	
	_		{ ")", ",", "<", "=", ">", "]", "and", "or",
	$\rightarrow$	ε	";", "end", "fi", "od", "esac", "do", "then",
-		D T1	"else" }
T	$\rightarrow$	B T1	
T1	$\rightarrow$	* B T1	
	$\rightarrow$	/ B T1	{ "+", "-", ")", ",", "<", "=", ">", "]",
	$\rightarrow$	٤	"and", "or", ";", "end", "fi", "od", "esac",
	_ ^		"do", "then", "else" }
В	$\rightarrow$	F B1	
B1	$\rightarrow$	^ F B1	
			{ "+", "-", "*", "/", ")", ",", "<", "=",
	$\rightarrow$	3	">", "]", "and", "or", ";", "end", "fi",
_	•	15	"od", "esac", "do", "then", "else" }
F	$\rightarrow$	ID	
	$\rightarrow$	cons	
		exp(E,E)	
	$\rightarrow$	(E)	
С	$\rightarrow$	X C1	

```
C1
       \rightarrow
              or X C1
       \rightarrow
                                                                        { "do", "then", ")", "]" }
              Y X1
 X
X1
       \rightarrow
              and Y X1
       \rightarrow
                                                                   { "or", "do", "then", ")", "]" }
       \rightarrow
 Υ
              E Y1
       \rightarrow
              not(C)
              [C]
       \rightarrow
Y1
              < E
       \rightarrow
              > E
       \rightarrow
              = E
       \rightarrow
              ID := E S1
       \rightarrow
              read(IDL S1
        \rightarrow
              write(IDL S1
        \rightarrow
              readln(IDL S1
       \rightarrow
              writeln(IDL S1
       \rightarrow
              case C do S M S1
       \rightarrow
              while C do S od S1
       \rightarrow
              if C then S S2
              foreach id in ID do S od S1
              with D begin S end S1
S1
       \rightarrow
              ; S3
       \rightarrow
              fi S1
S2
              else S fi S1
       \rightarrow
       \rightarrow
              S
S3
       \rightarrow
              ε
                                                          { "end", ":", "od", "fi", "esac", "else" }
              : C do S M
       \rightarrow
              esac
```

#### **Grammar Modification Notes**

- 1. The addition of **S3** is to ensure that every line ends in a ';', excluding the final 'end'.
- 2. The foreach loop has been altered by adding **ID** as the second argument. This allows the programmer to iterate through a sub-array.
- 3. The original definition for exponentiation is included as well as the revised definition.

#### **Semantic Notes**

- 1. My "readln()" function expects each identifier argument to be on a different line.
- 2. My "writeln()" function writes each identifier argument on a different line.

#### **Test Cases**

## **Program 1**

#### **Compilation / Execution**

```
CS4900 Compiler Testing\C5 4000 Compiler.exe

CS4900 Compiler
By: Bryant Nelson

CS4900 Compiler
By: Bryant Nelson

Compile Program

Run Program
Enter the file name of your program: 1.prog
Enter the name of your program: 1

Compilation Completed Successfully!

Please select from the following menu:
Compile Program
Run Program
Enter the following menu:
Compile Program
Enter the name of your program: 1

Enter the Input Source, for keyboard leave blank: 1.dat
Enter the Output Destination, for screen leave blank: 1.out

Program Completed!
```

#### **Program Alterations**

Added a semicolon to the line before 'end'. My grammar requires it, as noted above.

# **Program 2**

#### **Compilation / Execution**

```
CS4000 Compiler Testing\CS 4000 Compiler.exe

CS4000 Compiler
By: Bryant Nelson

COMPILE Program
Compiler
Compile Program
Compiler
Compiler

Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
Compiler
```

#### **Program & Data Alterations**

- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- Changed the data file so that each value was on a different line. As stated above, my semantics for "readln()" expect it this way.

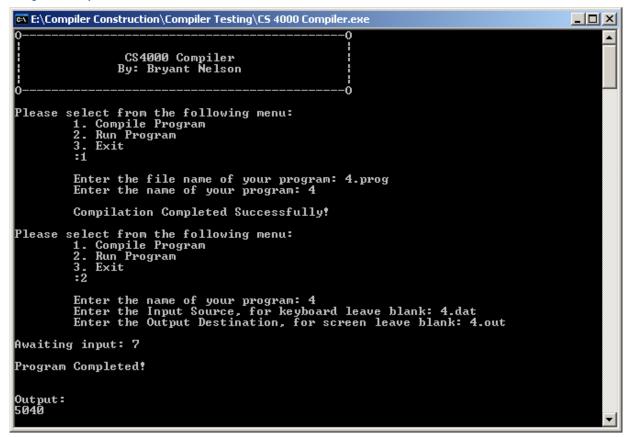
# **Program 3**

#### **Compilation / Execution**

- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- Since my grammar expects every line to end with a semicolon, the line that was originally supposed to cause an error will not. However, I removed the semicolon on that line in order to intentionally create an error.

# **Program 4**

#### **Compilation / Execution**



- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- I also had to add a semicolon to the last line of the while loop.

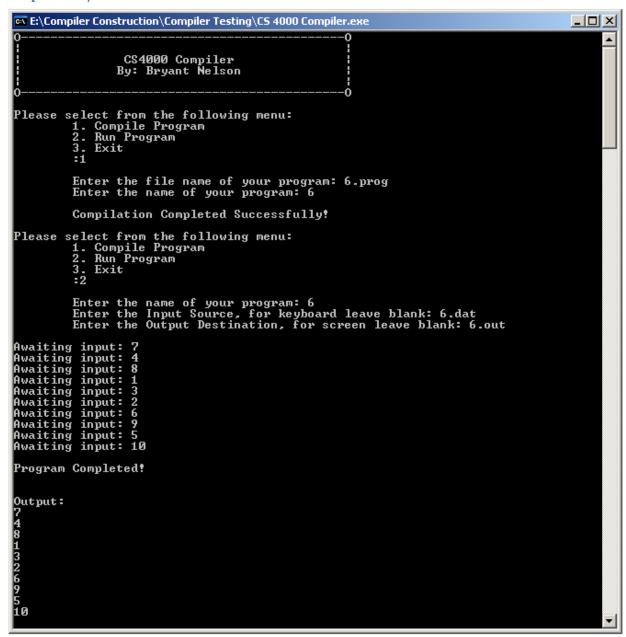
# **Program 5**

#### **Compilation / Execution**

- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- I also had to add a semicolon to the last line of the while loop.

## **Program 6**

#### **Compilation / Execution**



- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- Added a semicolon to the last line of the while loop, and to the last line of the foreach loop.
- Changed "i := 0;" to "i := 1;" since my indices start at 0.
- Removed "or i = 10" from the while condition since my indices start at 0.

## **Program 7**

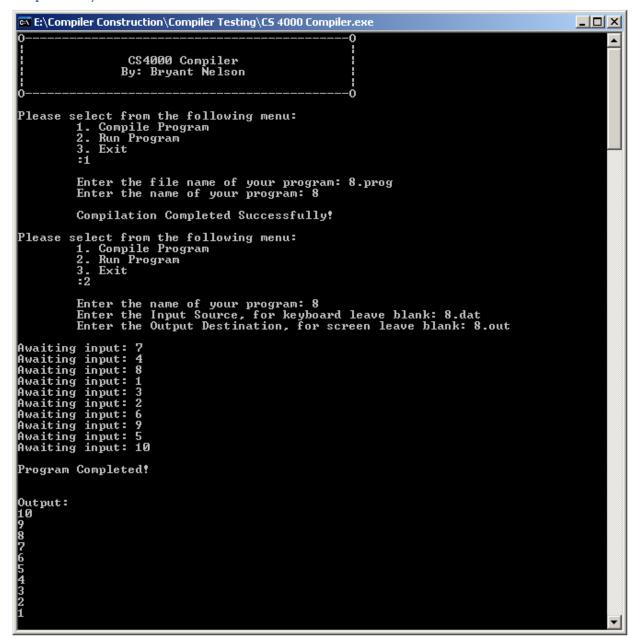
#### **Compilation / Execution**

```
E:\Compiler Construction\Compiler Testing\CS 4000 Compiler.exe
                                                                                                                                                                CS4000 Compiler
                              By: Bryant Nelson
Please select from the following menu:
1. Compile Program
2. Run Program
3. Exit
:1
                 Enter the file name of your program: 7.prog Enter the name of your program: 7
                 Compilation Completed Successfully!
Please select from the following menu:
1. Compile Program
2. Run Program
3. Exit
:2
                 Enter the name of your program: 7
Enter the Input Source, for keyboard leave blank: 7.dat
Enter the Output Destination, for screen leave blank: 7.out
Awaiting input: 7
Awaiting input: 4
Awaiting input: 8
Awaiting input: 1
Awaiting input: 3
Awaiting input: 2
Awaiting input: 6
Awaiting input: 5
Awaiting input: 5
Awaiting input: 5
Awaiting input: 10
Program Completed!
 Output:
 14
8
16
2
6
4
12
18
10
20
```

- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- Added a semicolon to the last line of both while loops and the foreach loop.
- Added a semicolon after the foreach loop's closing 'od'.
- Changed "i := 0;" to "i := 1;", in both instances, since my indices start at 0.
- Removed "or i = 10" from the while conditions since my indices start at 0.

#### **Program 8**

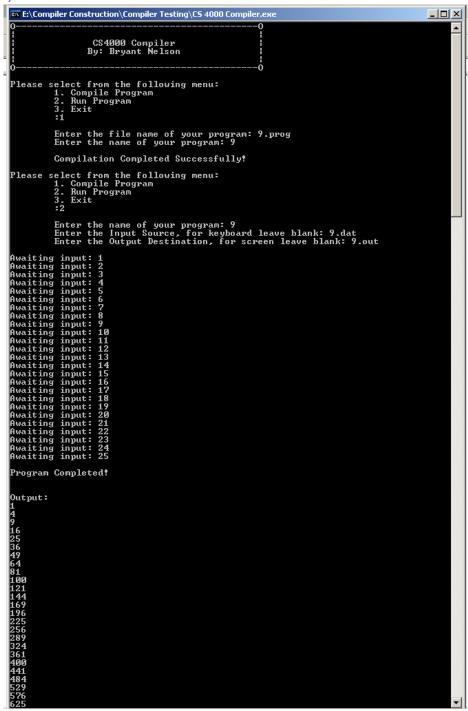
#### **Compilation / Execution**



- Added a semicolon to the line before 'end'. My grammar requires it, as noted above.
- Added a semicolon to the last line of all while loops.
- Changed "i := 0;" to "i := 1;", in all instances, since my indices start at 0, same for 'j'.
- Removed "or i = 10" from the while conditions since my indices start at 0, same for 'j'.

# **Program 9**

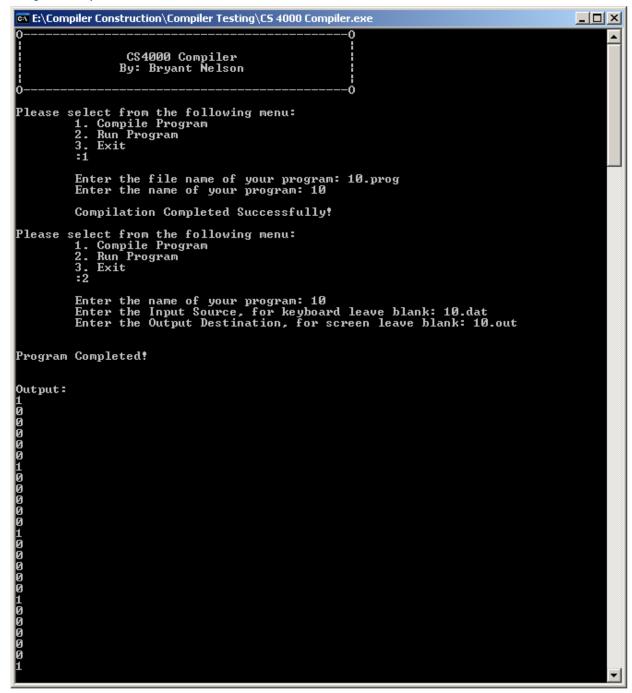
#### **Compilation / Execution**



- Added a semicolon everywhere that my altered grammar requires.
- Adjusted all indices and index counters to account for my zero indexed arrays.

## **Program 10**

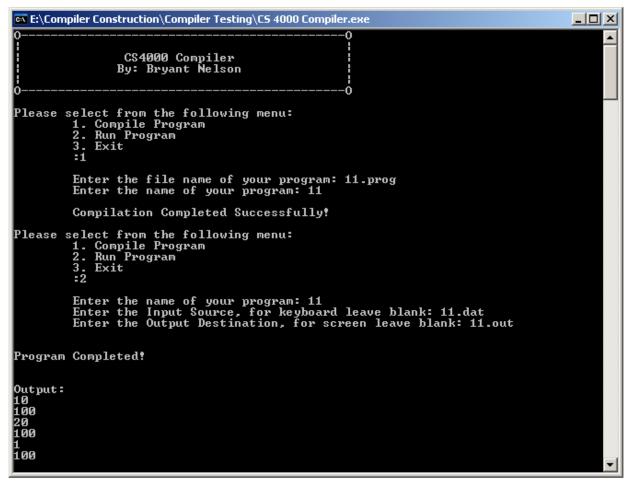
#### **Compilation / Execution**



- Added a semicolon everywhere that my altered grammar requires.
- Adjusted all indices and index counters to account for my zero indexed arrays.

## **Program 11**

## **Compilation / Execution**



- Added a semicolon everywhere that my altered grammar requires.
- Adjusted all indices and index counters to account for my zero indexed arrays.

# **Altered Programs**

```
program 1:
program
m : integer
begin
m := (40-10)/(9-3);
writeln(m);
end
data 1:
_____
program 2:
program
x1, x2, y1, y2, m : integer
begin
readln(x1,x2,y1,y2);
m := (y2-y1)/(x2-x1);
writeln(m);
end
data 2:
7
5
20
program 3: should generate an error
program
n,f,i : integer
begin
f:=1;
i:=1;
read(n);
while i < n \text{ or } i=n \text{ do}
      f := f*i;
      i := i+1;
od;
write(f);
end
data 3:
7
```

```
program 4:
program
n,f,i : integer
begin
f := 1;
i:=1;
read(n);
while i < n or i=n do
      f := f*i;
      i := i+1;
od;
write(f);
end
data 4:
_____
program 5: should generate an error
program
n,f,i: integer
begin
f:=1;
i:=1;
read(n);
while i < n \text{ or } i=n \text{ do}
      f := f*i;
       i := i+1;
od;
x:=f;
write(x);
end
data 5:
```

```
program 6:
(If your indices begin with zero change the program below to account for
that)
program
n:array[10]
x, i: integer
begin
i := 0;
while i < 10 do
    read(n[i]);
     i:=i+1;
od;
foreach x in n do
      writeln(x);
od;
end
data 6:
7 4 8 1 3 2 6 9 5 10
_____
program 7:
(If your indices begin with zero change the program below to account for
program
n:array[10]
x,i: integer
begin
i:=0;
while i < 10 do
     read(n[i]);
      i := i+1;
od;
foreach x in n do
      x := x * 2;
od;
i:=0;
while i < 10 do
     writeln(n[i]);
      i:=i+1;
od;
end
data 7:
7 4 8 1 3 2 6 9 5 10
```

```
program 8:
(If your indices begin with zero change the program below to account for
that)
program
n:array[10]
i,j,t: integer
begin
i:=0;
while i < 10 do
      read(n[i]);
      i:=i+1;
od;
i := 0;
while i < 10 do
      j:=0;
      while j < 10 do
            if n[i] > n[j] then
                   t:=n[i];
                   n[i]:=n[j];
                   n[j]:=t;
            fi;
            j := j+1;
      od;
i:=i+1;
od;
i := 0;
while i < 10 do
      writeln(n[i]);
      i := i+1;
od;
end
data 8:
7 4 8 1 3 2 6 9 5 10
```

\_\_\_\_\_

```
program 9:
(If your indices begin with zero change the program below to account for
that)
program
n:array[5,5]
i,j: integer
begin
i := 0;
while i < 5 do
      j:=0;
      while j < 5 do
            readln(n[i,j]);
            j:=j+1;
      od;
i:=i+1;
od;
i:=0;
while i < 5 do
      j:=0;
      while j < 5 do
            n[i,j] := exp(n[i,j],2);
      j:=j+1;
      od;
i := i+1;
od;
i := 0;
while i < 5 do
      j:=0;
      while j < 5 do
            writeln(n[i,j]);
            j:=j+1;
      od;
i := i+1;
od;
end
data 9:
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

```
16
17
18
19
20
21
22
23
24
25
_____
program 10:
(If your indices begin with zero change the program below to account for
that)
program
n:array[5,5]
i,j: integer
begin
i := 0;
while i < 5 do
     j:=0;
     while j < 5 do
      case i=j do
           n[i,j]:=1;
      :not(j=i) do
           n[i,j]:=0;
     esac;
     j := j + 1;
     od;
i:=i+1;
od;
i := 0;
while i < 5 do
      j:=0;
      while j < 5 do
           writeln(n[i,j]);
           j:=j+1;
      od;
i:=i+1;
od;
end
data 10:
```

-----

```
program 11:
(extra credit)
program
x ,y : integer
begin
x := 1;
y:=100;
with x:integer begin
      x:=10;
      writeln(x, y);
      with x:integer begin
            x:=20;
            writeln(x, y);
      end;
end;
writeln(x, y);
end
data 11:
```

#### **Custom Test Cases**

## **Exponentiation Testing**

#### **Compilation / Execution**

#### **Comments**

This program uses both means of exponentiation to show that they are in fact equal.

# **Factorial Example**

#### **Compilation / Execution**

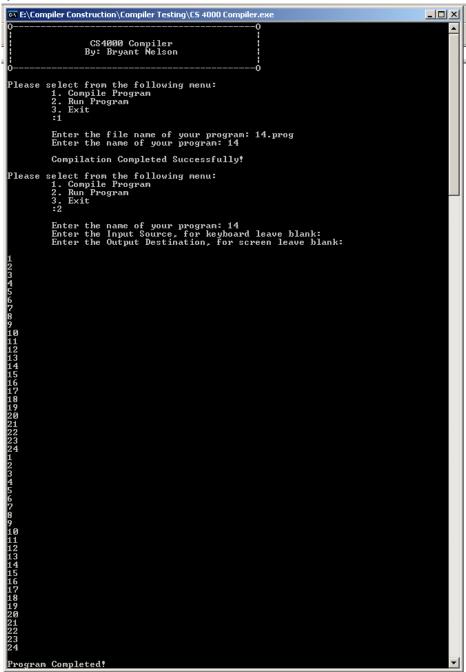
```
E:\Compiler Construction\Compiler Testing\CS 4000 Compiler.exe
                                                                                                                              CS4000 Compiler
By: Bryant Nelson
Please select from the following menu:
1. Compile Program
2. Run Program
3. Exit
:1
             Enter the file name of your program: 13.prog Enter the name of your program: 13
             Compilation Completed Successfully!
Please select from the following menu:
1. Compile Program
2. Run Program
3. Exit
:2
             Enter the name of your program: 13
Enter the Input Source, for keyboard leave blank:
Enter the Output Destination, for screen leave blank:
Awaiting input: -5
Awaiting input: 6
720
Awaiting input: 9
362880
Awaiting input: 5
Awaiting input: 7
5040
120
Awaiting input: 4
24
Awaiting input: -5
Program Completed!
```

#### **Comments**

This program reads in the first input, and uses it as the sentinel value to stop the program. It then continues to read in values and output the factorial of those values until the sentinel value is encountered.

# With & ForEach Example

#### **Compilation / Execution**



#### **Comments**

This program firmly illustrates the ability of the "with" statement. As you can see, the variable names are redefined as arrays of different sizes, and even as integers, nested within other "with" bodies. The size and type of each variable has to be tracked throughout both Semantic analysis and Execution.