Introduction to exercise 6a: Abstract Syntax Tree (AST)

CS-460: Programming Language

Robert Bruce

What is an Abstract Syntax Tree (AST)?

- An Abstract Syntax Tree (AST) is an abridged version of a Concrete Syntax Tree (CST).
- An AST uses the same LCRS (left-child right-sibling) binary tree data structure to store tokens as a CST.
- When applicable, the AST may also point into the symbol table (e.g. for function, procedure, and variable declarations or assignment statements). Note: the pointer I'm referring to does not impact left-child or right-sibling pointers. It would be a miscellaneous extra attribute.
- Recall that a CST contains every token present in the input program parsed. Consequently, a CST is syntactically verbose. In contrast to a CST, an AST removes unnecessary tokens from a CST.

Important note about Abstract Syntax Tree construction

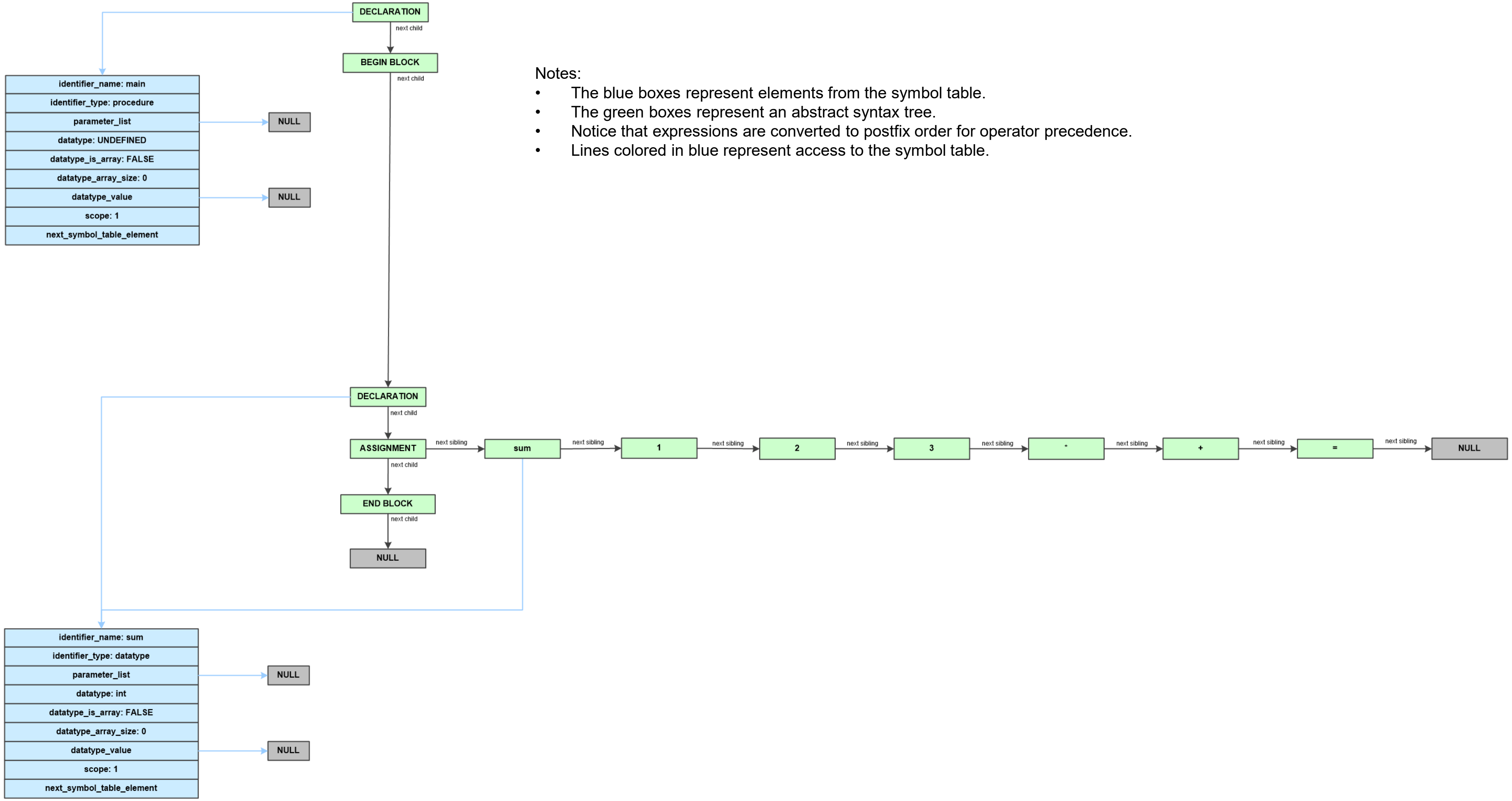
Note: The procedure for constructing an Abstract Syntax Tree (AST) can differ widely in real-world implementation.

- I view the AST as a Concrete Syntax Tree that has been prepared for program interpretation and execution.
- My ASTs contain all Boolean and numerical expressions (from assignment statements) in postfix notation. This makes it easy to compute the result of the expression on-the-fly during run-time of an interpreted program.
- The data structure for each element of my AST contains an extra field: a pointer into the symbol table. For example, a variable listed in my AST would point to the appropriate entry in the symbol table for that variable. This makes it very easy to know which variable to update and/or retrieve a value from during run-time of an interpreted program.

Simple program integrating AST with Symbol Table

```
procedure main (void)
{
  int sum;
  sum = 1 + 2 * 3;
}
```

Simple program integrating AST with Symbol Table



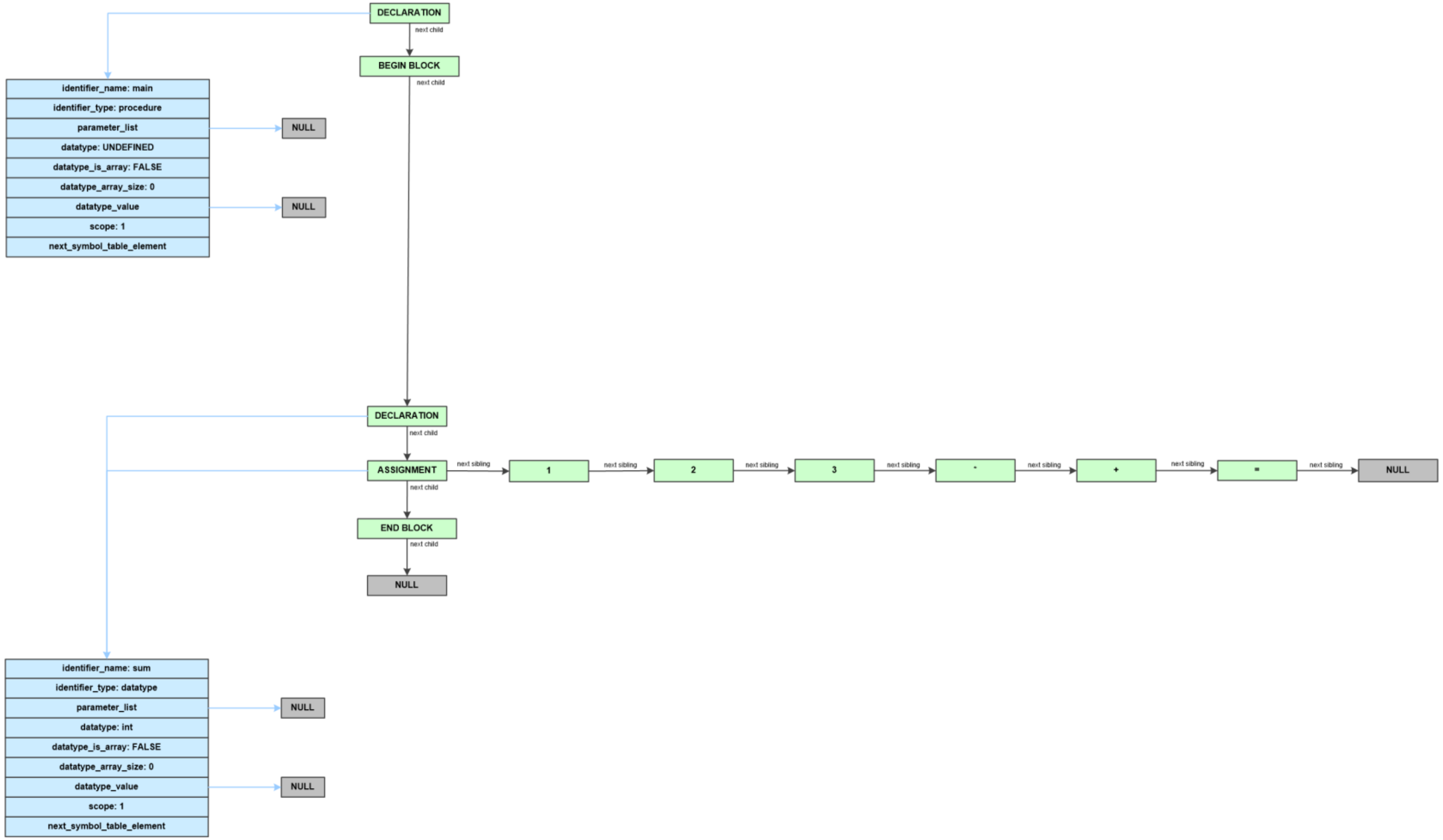
Notes:

- The blue boxes represent elements from the symbol table.
- The green boxes represent an abstract syntax tree.
- Notice that expressions are converted to postfix order for operator precedence.
- Lines colored in blue represent access to the symbol table.

Reflection: Simple program integrating AST with Symbol Table

Note: in the previous example, you may believe I was inefficient in storing the variable "sum" - I concur my method is inefficient - as an extra element of the Abstract Syntax Tree since the ASSIGNMENT element could point to "sum" in the Symbol table:

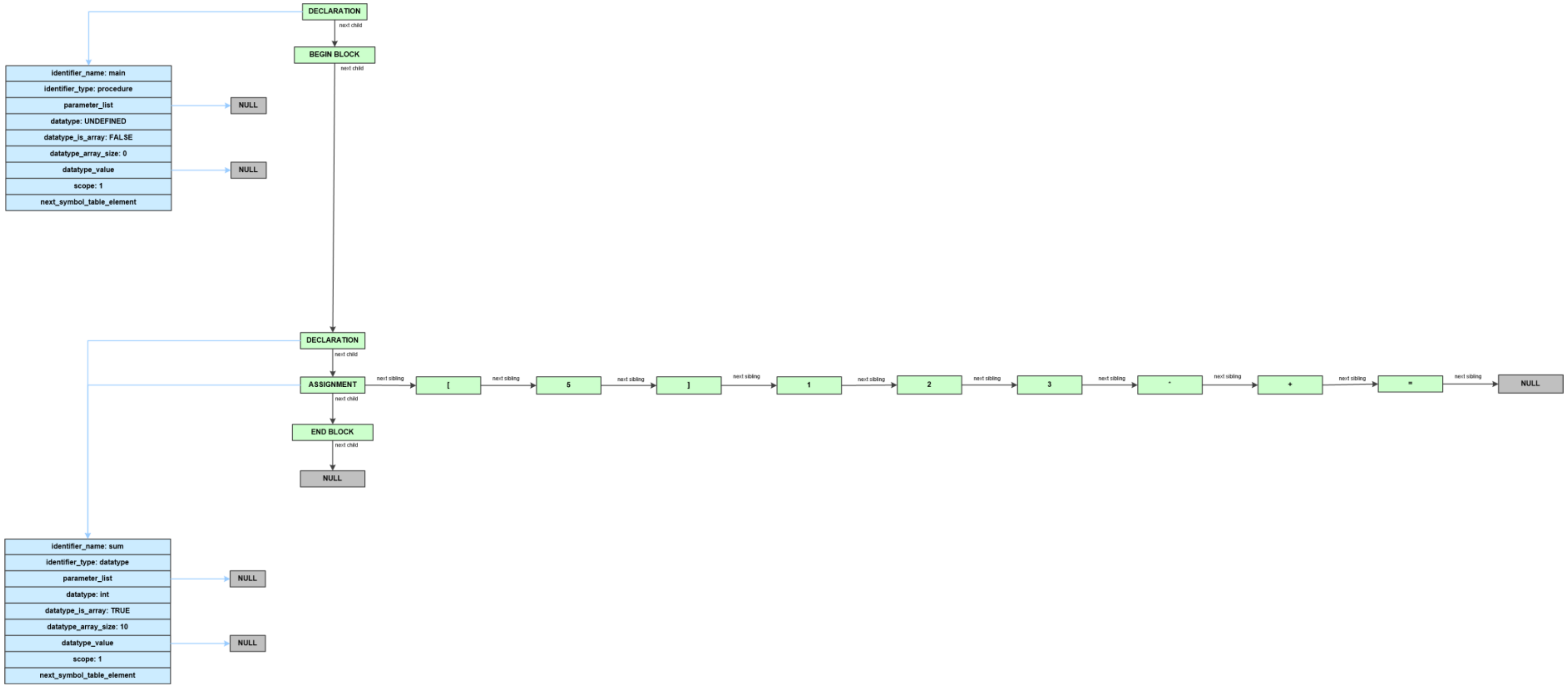
```
sum = 1 + 2 * 3;
```



Reflection: Simple program integrating AST with Symbol Table

Now consider another scenario:

```
procedure main (void)
{
  int sum[10];
  sum[5] = 1 + 2 * 3;
}
```



Example program

```
function int sum_of_first_n_squares (int n)
{
  int sum;
  sum = 0;
  if (n >= 1)
  {
    sum = n * (n + 1) * (2 * n + 1) / 6;
  }
  return sum;
}

procedure main (void)
{
  int n;
  int sum;
  n = 100;
  sum = sum_of_first_n_squares (n);
  printf ("sum of the squares of the first %d numbers = %d\n", n, sum);
}
```

Example program

Break the program into a series of lines:

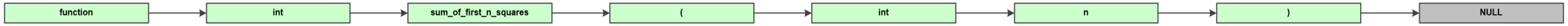
```
function int sum_of_first_n_squares (int n)
{
  int sum;
  sum = 0;
  if (n >= 1)
  {
    sum = n * (n + 1) * (2 * n + 1) / 6;
  }
  return sum;
}

procedure main (void)
{
  int n;
  int sum;
  n = 100;
  sum = sum_of_first_n_squares (n);
  printf ("sum of the squares of the first %d numbers = %d\n", n, sum);
}
```

Concrete Syntax Tree versus Abstract Syntax Tree: Line 1

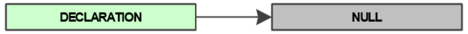
Line 1:

```
function int sum_of_first_n_squares (int n)
```

Concrete Syntax Tree for line 1:



Abstract Syntax Tree for line 1:



Concrete Syntax Tree versus Abstract Syntax Tree: Line 2

Line 2:

{

Concrete Syntax Tree for line 2:

Abstract Syntax Tree for line 2:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 3

Line 3:

```
int sum;
```

Concrete Syntax Tree for line 3:

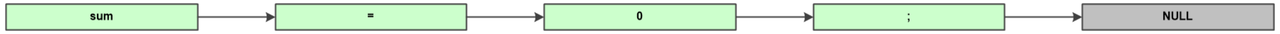
Abstract Syntax Tree for line 3:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 4

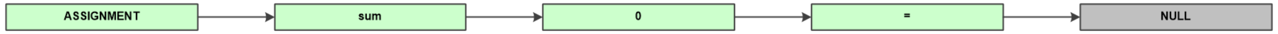
Line 4:

```
sum = 0;
```

Concrete Syntax Tree for line 4:



Abstract Syntax Tree for line 4:

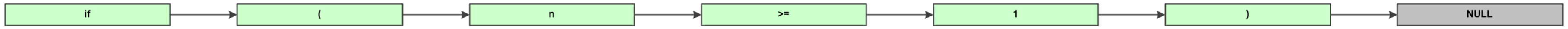


Concrete Syntax Tree versus Abstract Syntax Tree: Line 5

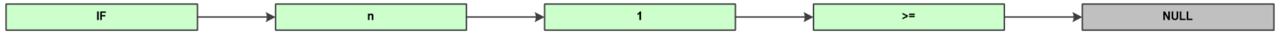
Line 5:

```
if (n >= 1)
```

Concrete Syntax Tree for line 5:



Abstract Syntax Tree for line 5:



Concrete Syntax Tree versus Abstract Syntax Tree: Line 6

Line 6:

{

Concrete Syntax Tree for line 6:

Abstract Syntax Tree for line 6:

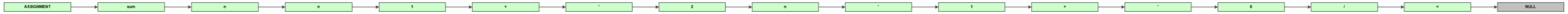
Concrete Syntax Tree versus Abstract Syntax Tree: Line 7

Line 7:

```
sum = n * (n + 1) * (2 * n + 1) / 6;
```

Concrete Syntax Tree for line 7:

Abstract Syntax Tree for line 7:



Concrete Syntax Tree versus Abstract Syntax Tree: Line 8

Line 8:

```
}
```

Concrete Syntax Tree for line 8:

Abstract Syntax Tree for line 8:

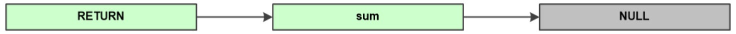
Concrete Syntax Tree versus Abstract Syntax Tree: Line 9

Line 9:

```
return sum;
```

Concrete Syntax Tree for line 9:

Abstract Syntax Tree for line 9:



Concrete Syntax Tree versus Abstract Syntax Tree: Line 10

Line 10:

}

Concrete Syntax Tree for line 10:

Abstract Syntax Tree for line 10:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 11

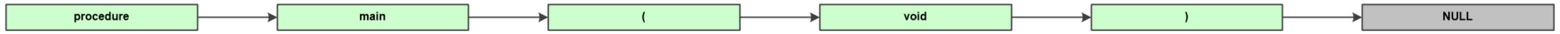
Line 11:
There is nothing to process on line 11. Skip this line since there was nothing tokenized in the CST.

Concrete Syntax Tree versus Abstract Syntax Tree: Line 12

Line 12:

```
procedure main (void)
```

Concrete Syntax Tree for line 12:



Abstract Syntax Tree for line 12:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 13

Line 13:

{

Concrete Syntax Tree for line 13:

Abstract Syntax Tree for line 13:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 14

Line 14:

```
int n;
```

Concrete Syntax Tree for line 14:

Abstract Syntax Tree for line 14:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 15

Line 15:

```
int sum;
```

Concrete Syntax Tree for line 15:

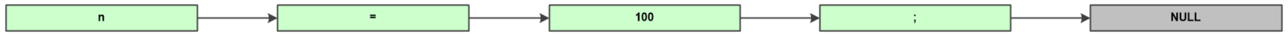
Abstract Syntax Tree for line 15:

Concrete Syntax Tree versus Abstract Syntax Tree: Line 16

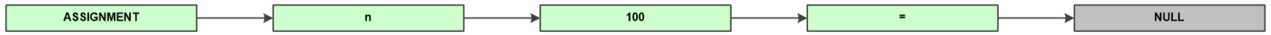
Line 16:

```
n = 100;
```

Concrete Syntax Tree for line 16:



Abstract Syntax Tree for line 16:

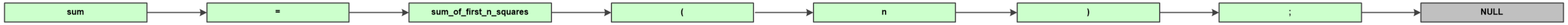


Concrete Syntax Tree versus Abstract Syntax Tree: Line 17

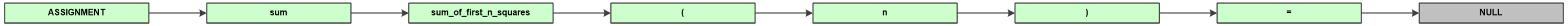
Line 17:

```
sum = sum_of_first_n_squares (n);
```

Concrete Syntax Tree for line 17:



Abstract Syntax Tree for line 17:



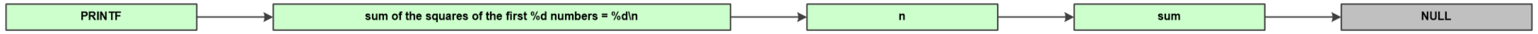
Concrete Syntax Tree versus Abstract Syntax Tree: Line 18

Line 18:

```
printf ("sum of the squares of the first %d numbers = %d\n", n, sum);
```

Concrete Syntax Tree for line 18:

Abstract Syntax Tree for line 18:



Concrete Syntax Tree versus Abstract Syntax Tree: Line 19

Line 19:

```
}
```

Concrete Syntax Tree for line 19:

Abstract Syntax Tree for line 19:

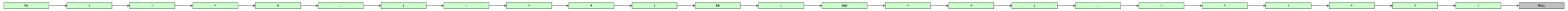
Concrete Syntax Tree versus Abstract Syntax Tree: example FOR statement

```
for (i = 0; (i < 4) && (digit > -1); i = i + 1)
```

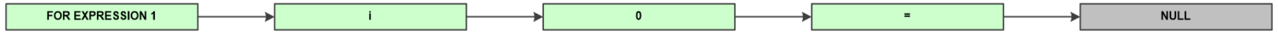
The above FOR statement can be broken down into three components:

- Expression 1: *i = 0;*
- Expression 2: *(i < 4) && (digit > -1);*
- Expression 3: *i = i + 1*

Concrete Syntax Tree:



Abstract Syntax Tree (Expression 1):



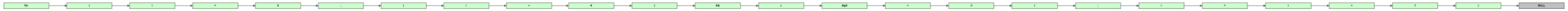
Concrete Syntax Tree versus Abstract Syntax Tree: example FOR statement

```
for (i = 0; (i < 4) && (digit > -1); i = i + 1)
```

The above FOR statement can be broken down into three components:

- Expression 1: *i = 0;*
- Expression 2: *(i < 4) && (digit > -1);*
- Expression 3: *i = i + 1*

Concrete Syntax Tree:



Abstract Syntax Tree (Expression 2):

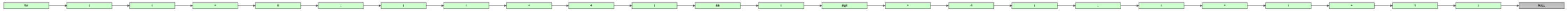
Concrete Syntax Tree versus Abstract Syntax Tree: example FOR statement

```
for (i = 0; (i < 4) && (digit > -1); i = i + 1)
```

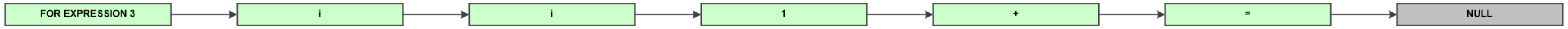
The above FOR statement can be broken down into three components:

- Expression 1: *i = 0;*
- Expression 2: *(i < 4) && (digit > -1);*
- Expression 3: *i = i + 1*

Concrete Syntax Tree:



Abstract Syntax Tree (Expression 3):



Concrete Syntax Tree versus Abstract Syntax Tree: example FOR statement

```
for (i = 0; (i < 4) && (digit > -1); i = i + 1)
```

The above FOR statement can be broken down into three components:

- Expression 1: *i = 0;*
- Expression 2: *(i < 4) && (digit > -1);*
- Expression 3: *i = i + 1*

Concrete Syntax Tree:

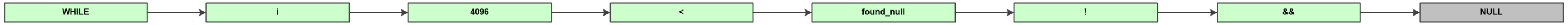
Abstract Syntax Tree (Expression 1, Expression 2, and Expression 3):

Concrete Syntax Tree versus Abstract Syntax Tree: example while statement

```
while ((i < 4096) && (!found_null))
```

Concrete Syntax Tree:

Abstract Syntax Tree:



Concrete Syntax Trees: More examples?

Yes! There are more examples of Concrete Syntax Trees posted to Canvas.

- Please see sample input test programs on Canvas for Programming Assignment 5: Abstract Syntax Tree.
- For each of the five sample input test programs, I've also created a Concrete Syntax Tree.

You are welcome to view these additional examples during our in-class exercise.