Introduction to exercise 3a: Concrete Syntax Tree (CST)

CS-460: Programming Language

Robert Bruce

What is a Concrete Syntax Tree (CST)?

- A Concrete Syntax Tree (CST) is a visual representation of the entire tokenized input program stored in a tree data structure.
- For this class, we will use an LCRS (left-child right-sibling) binary tree to store our tokens.

Example program

```
procedure main (void)
{
  int sum;

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

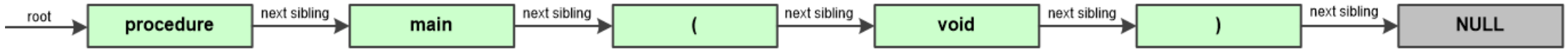
Example program

Break the program into a series of lines:

```
1. procedure main (void)
2. {
3.    int sum;
4.
5.    sum = 1 + 2 * 3;
6. }
```

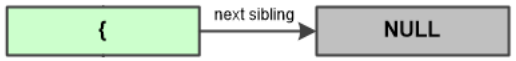
Line one from input program

```
procedure main (void)
```



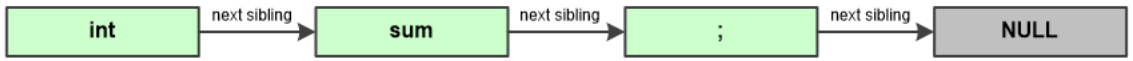
Line two from input program

{



Line three from input program

```
int sum;
```

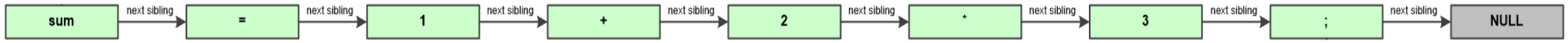


Line four from input program

- Line four is a blank line.
- There is nothing to tokenize and nothing to store in the tree.
- Proceed to the line five.

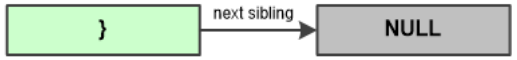
Line five from input program

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



Line six from input program

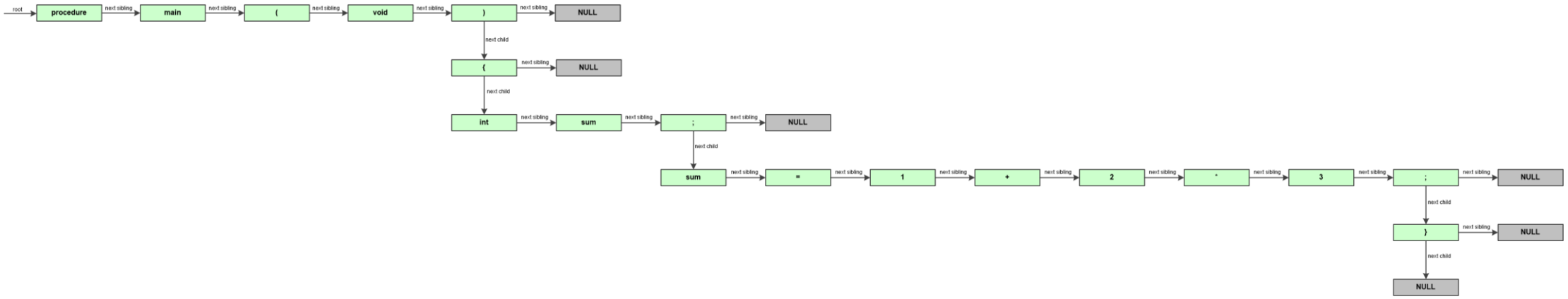
```
}
```



Example program inserted into an LCRS binary tree

```
procedure main (void)
{
  int sum;

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



Reflection

The impact of creating a Concrete Syntax Tree (CST):

- If you can successfully create a CST, it signifies your input program (i.e. the program that is parsed) is syntactically correct; however, the input program could still have errors. For example, an input program could refer to a variable that was never declared.

Next step:

- A CST is the first line of defense in looking for syntax errors. The next step is creating a symbol table of all procedures, functions, and variables. A symbol table (future in-class assignment) will be able to determine which variables are referenced (perhaps in an expression) yet never declared.

Final thought:

- The symbol table will also be useful when we create Abstract Syntax Tree (AST). This will also be a future in-class assignment.