Abstract Syntax Trees

1. What are they?
2. Why build them?
3. How do you build them?
4. What do I do with them now?

2. What's an AST?

- AST are an internal representation of an input stream.

- They should be designed so that they are more convenient to "play" with than the text input stream.

- ASTs are useful for all but the simplest translations: i.e., syntax-directed translations.

- ANTLR provides an automatic and an explicit mechanism.

- The programmer defines the node contents, but must be able to view ASTs as child-sibling trees.

2. Why build ASTs?

- Often, the programmer must obtain information from random portions of an input stream. It's much easier to walk a tree than it is to cotinually rewind and parse an input stream.

- It's not that hard to build them.

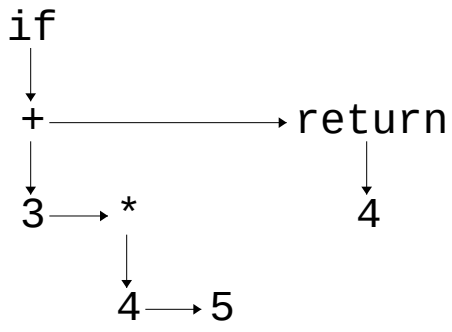
- SORCERER provides a very convenient method of traversing and rewriting trees.

3. How do you build ASTs?

Automatic mechanism: designed to be good at creating expression trees. But, works pretty well for simple tree construction. For example, given input:

```
if 3+4*5 then return 4;
```

the tree:



could be constructed easily with the automatic mechanism (grammar on next slide).

```
#header <<
    #include "charbuf.h"
    #define AST_FIELDS   int token, ival;
>>
<<
/* required function: how to convert from attribute
to AST node */
void
zzcr_ast(AST *node, Attrib *cur, int token, char *text)
{
    node->token = token;
    node->ival = atoi(text);
}
main()
{
    AST *root=NULL;
    ANTLR(e(&root), stdin);
}
>>

stat:   "if"^ e "then"! stat ";"!
    |   "return"^ e
    ;
e   :   e1 ("\+"^ e1)* ;
e1  :   e2 ("\*"^ e2)* ;
e2  :   "[0-9]+" ;
```

ASTs can be built explicitly as well (here we assume that a `zzmk_ast()` function exists that takes an `Attrib` as an argument):

```
stat!:   "if" e "then" stat ";"  <<#0=#(#[$1],#2,#4);>>
     |   "return" e              <<#0=#(#[$1],#2);>>
     ;
```

Key elements: node constructors `#[...]` and tree constructors `#(...)`. Both are shorthands for function calls: `zzmk_ast()` and `zztmake()`. Note the "!" on the rule ref: implies that ANTLR should shut off *automatic* AST construction for rule `stat`.

4. What do I do with them now?

- Can either right recursive routines to walk the tree and/or rewrite the tree.

- Or, can use SORCERER to walk and/or rewrite.