Sequence 2.1 – Abstract Syntax Tree

P. de Oliveira Castro S. Tardieu

Understanding the source code

- The compiler job is to understand the program source code and transform it into an executable representation.
- In order to do this, the compiler needs to assign meaning to the characters it encounters in the program source.

```
let var a := 3 in
  a := a + 1;
  print_int(a);
  print("\n")   /* Go to next line */
end
```

In the above source code, the compiler must understand the meaning of letters, spaces, numbers, semicolons, quotes, etc.

Giving meaning to symbols and words

- Symbols and words may have simultaneous meanings depending on the compilation phase.
- The second a in `a := a + 1` is an identifier. It is also the left operand of the binary operation +.
- Different phases during the compilation process will make use of the various meanings.

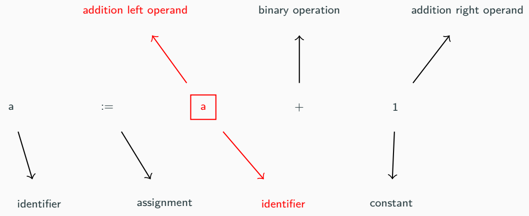

Figure 1: Understanding a := a + 1

Representing the source using an abstract syntactic tree (AST)

A tree can represent the program source in a more structured way:

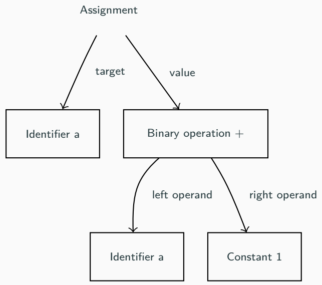

Figure 2: AST for a := a + 1

The tree does not store superfluous information such as comments which have no use during the compilation process. The tree is an abstract represention of the syntax of the program source code.

Representing the tree in C++

- The tree is represented through a collection of classes inheriting from the command Node type with two direct descendants: Expression (to represent Tiger expression) and Decl (to represent variables and function declarations).
- Every node descendant flavour contains relevant attributes and subtrees.

```
class BinaryOperation : public Expression {
  std::string operation;  // Binary operator such as "+"
  Expression *left;       // Left operand
  Expression *right;      // Right operand
  ...
}
```

Conclusion

- Instead of continuously working from the program source code, the compiler will first analyze it and build a corresponding tree which conveys the same information in a more abstract way.
- The tree stores all the relevant information about every entity encountered in the source code.
- The tree is used in subsequent compilation phases to analyze the meaning of a program and translate it to machine code.