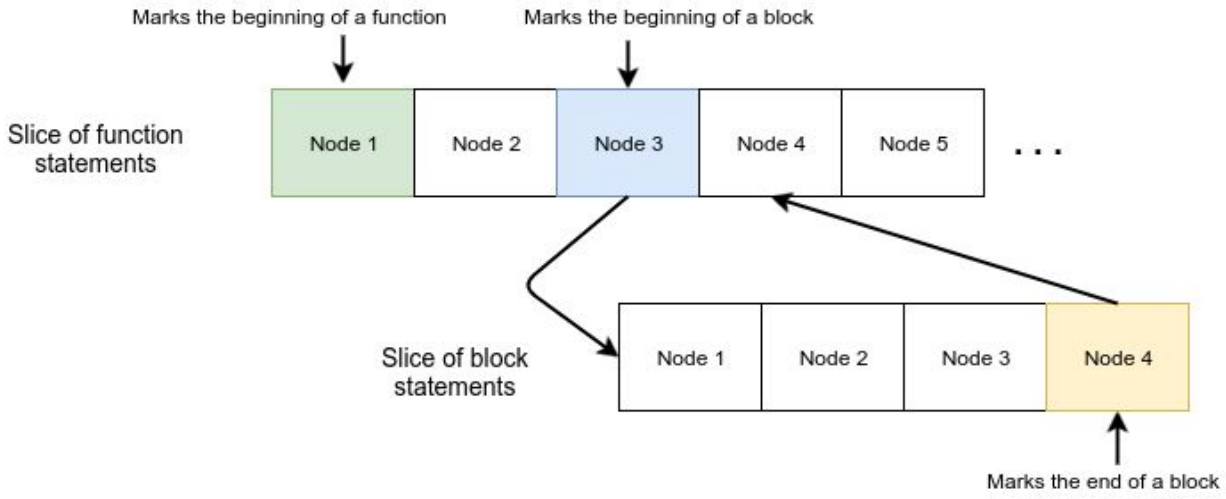
Three-address code

The three address code can be represented as a slice of function data structures `(Fn)`, with the first member as `main()` which will serve as an entry point to the program.



Functions

A function can be represented as -

```
type Fn struct {
      stmts    []Stmt
      symtab   map[string]*SrcVar
      labelmap map[string]int
}
```

- Each entry in the slice will represent a statement inside the function.
- `labelmap` is a map of following key-value pairs -

  ```
  label name : index into “stmts”
  ```

- `symtab` is a symbol table pertaining to a function and will be a map of following key-value pairs -

  ```
  variable name : pointer to SrcVar
  ```

- The advantage of using a slice is random access with dynamic size.

Statements

A statement can be represented as -

```
type Stmt struct {
      op  string
      dst string
      src []SrcVar
      blk *Block
}
```

- Since there can be multiple source variables, `src` is a slice containing the same.
- To differentiate between a *block header* and a normal statement, each `Stmt` contains a `blk` field with the following properties -
    - For statements, this field is `nil`.
    - For *block headers*, this field contains a pointer to the relevant block. *A block header marks the beginning of a block*.

A source variable can be represented as -

```
type SrcVar struct {
      typ string
      val string
}
```

Scoping

Labels

Since a function cannot contain duplicate labels, `labelmap` suffices for this purpose.

Block scopes (if, for, {...})

Each block DS contains a pointer to its parent's (i.e. the function) symbol table along with it's own symbol table which is initialized to be empty. When looking up a variable -

- Check if the variable exists in the local symbol table.
- If not, then fetch the variable in the parent's symbol table and copy it locally if found (similar to COW, only this time it's copy-on-lookup).

```
type Block struct {
      stmts     []Stmt
      symtab    map[string]*SrcVar
      symtabptr *symtab
}
```