Compiler Construction

Symbol Table

Symbol Table: definition

Misnamed!

Maintain the current **environment(s)**

Relation with binder

Maintained and widely used by the binder to build the threaded AST

Preliminary remark

Maintaining a such table is easy if all identifiers are unique!
$\Rightarrow$ Use a single hash table
$\Rightarrow$ identifier is the key, node definition the value

Preliminary remark

Maintaining a such table is easy if all identifiers are unique!
$\Rightarrow$ Use a single hash table
$\Rightarrow$ identifier is the key, node definition the value

How to handle scopes?

Binder: recall

1. Start with an empty environment
2. For each declaration, add the identifier to the environment
3. For each use, link to the correct declaration
4. At the end of a scope remove identifiers that are no longer visible

Strategy 1: binder and symbol table

1. Start with an empty environment
2. When entering scope, add new **empty** environment
3. For each declaration, add the identifier to the latest environment
4. For each use, walk all environment to find latest declaration
5. At the end of a scope remove last environment

Strategy 1: technical details

Symbol table

Maintain a **stack** of environment!

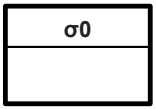
Environment

Use a simple **hashtable** to represent an environment

Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

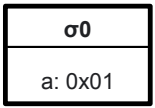
Symbol Table:



Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

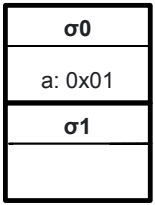
Symbol Table:



Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

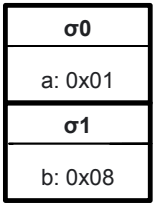
Symbol Table:



Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

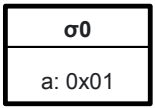
Symbol Table:



Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

Symbol Table:



Strategy 1: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

Symbol Table:

Pros and cons

Pros:

- Easy to implement!
- Lookup in a environment fast O(1)

Cons:

- Full Lookup: O(|S|) with |S| the number of nested scopes

Pros and cons

Pros:

- Easy to implement!
- Lookup in a environment fast O(1)

Cons:

- Full Lookup: O(|S|) with |S| the number of nested scopes

⇒ **Can we improve the full lookup cost?**

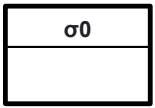
Strategy 2: binder and symbol table

1. Start with an empty environment
2. When entering scope, add a copy of the latest environment
3. For each declaration, add the identifier to the latest environment
4. For each use, walk all environment to find latest declaration
5. At the end of a scope remove last environment

Strategy 2: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

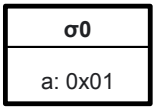
Symbol Table:



Strategy 2: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

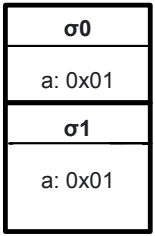
Symbol Table:



Strategy 2: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

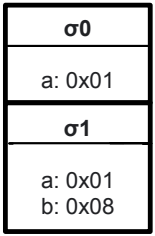
Symbol Table:



Strategy 2: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```

Symbol Table:



Pros and cons

Pros:

- Relatively easy to implement!
- Lookup in a environment fast O(1)
- Full Lookup fast O(1)

Cons:

- Memory usage
- Beware with chunks

Pros and cons

Pros:

- Relatively easy to implement!
- Lookup in a environment fast O(1)
- Full Lookup fast O(1)

Cons:

- Memory usage
- Beware with chunks

$\Rightarrow$ **Can we do better?**

Strategy 3: Tuned symbol table

Main Idea

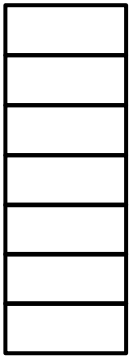
Mix between hashtable and (doubly) linked lists!

A single hash table

GCC/Clang style symbol table

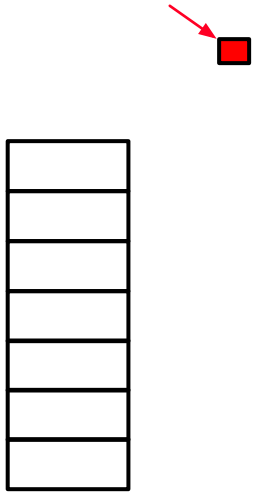
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



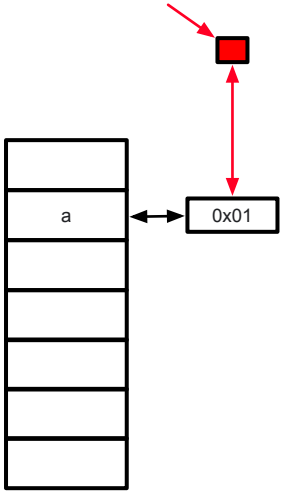
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



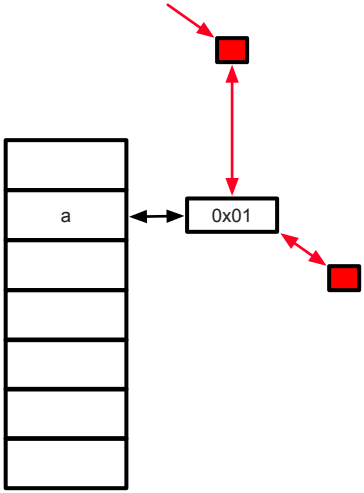
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



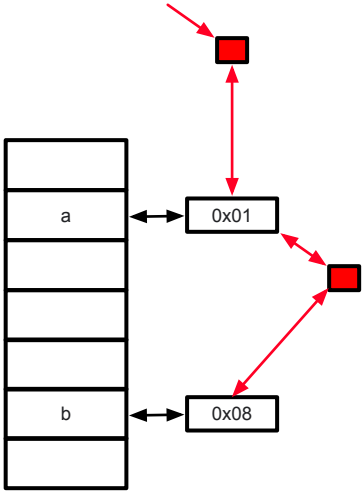
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



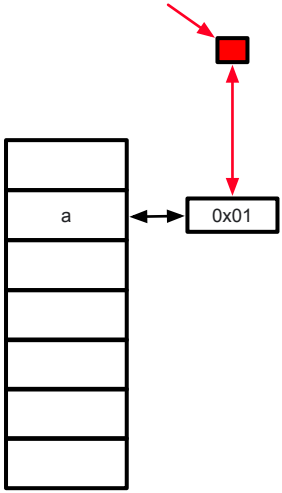
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



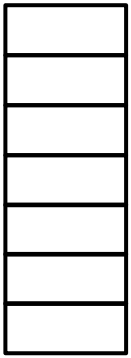
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



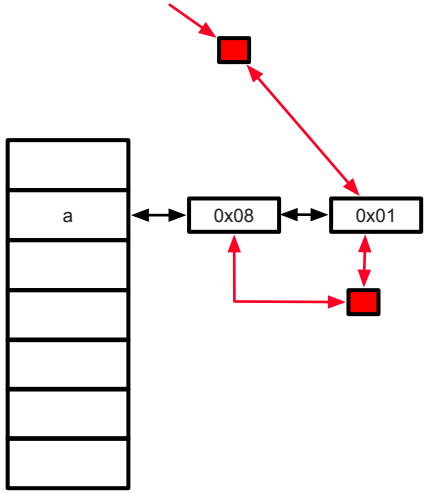
Strategy 3: example

```
let var a := 42
in
   let var b := 51
   in
       a + b
   end;
   a
end
```



Strategy 3: name duplication

```
let var a := 42
in
   let var a := 51
   in
       a * 2
   end;
   a
end
```



Scoped Symbol Table interface

```
template <typename Entry_T>
class Table
{
public:
  Table();

  auto put(symbol key, Entry_T& val) -> void;
  auto get(symbol key) const -> Entry_T*;

  auto scope_begin() -> void;
  auto scope_end() -> void;

  auto print(std::ostream& ostr) const -> void;
};
```

Summary