# Strukturen und Klassen

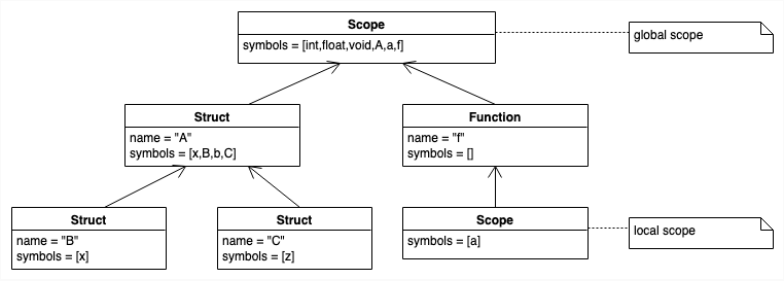
Carsten Gips (FH Bielefeld)

# Strukturen

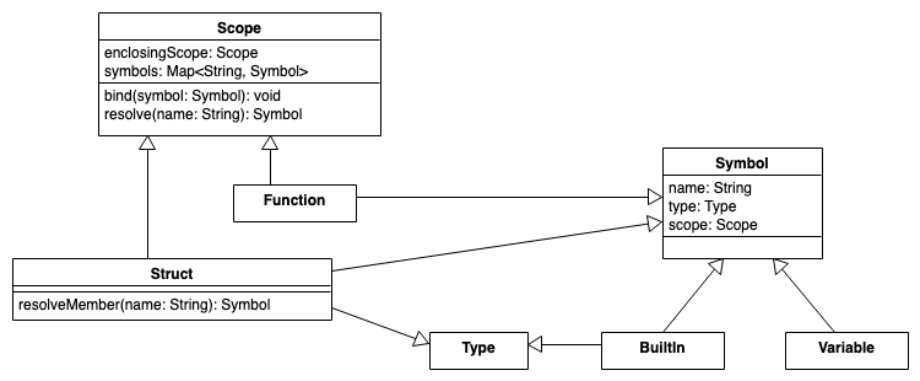
```
struct A {
    int x;
    struct B {int x;};
    B b;
    struct C {int z;};
};
A a;
void f() {
    A a;
    a.b.x = 42;
}
```

# Strukturen

```
struct A {
    int x;
    struct B {int x;};
    B b;
    struct C {int z;};
};
A a;
void f() {
    A a;
    a.b.x = 42;
}
```



# Strukturen: Erweiterung der Symbole und Scopes




Quelle: Eigene Modellierung nach einer Idee in (Parr 2010, 162)

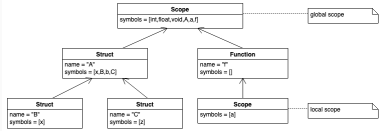
# Strukturen: Auflösen von Namen

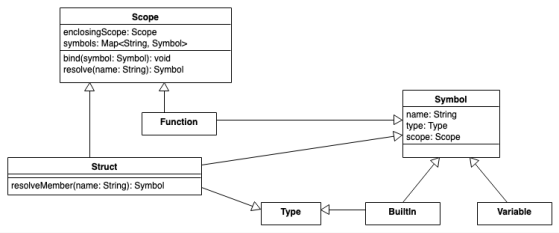
```
class Struct(Scope, Symbol, Type):
    def resolveMember(name):
        return symbols[name]
```

=> Auflösen von "`a.b`":

- `a` im "normalen" Modus mit `resolve()` über den aktuellen Scope
- Typ von `a` ist `Struct` mit Verweis auf den eigenen Scope
- `b` nur innerhalb des `Struct`-Scopes mit `resolveMember()`

```
struct A {
    int b;
};
void f() {
    A a;
    a.b = 42;
}
```



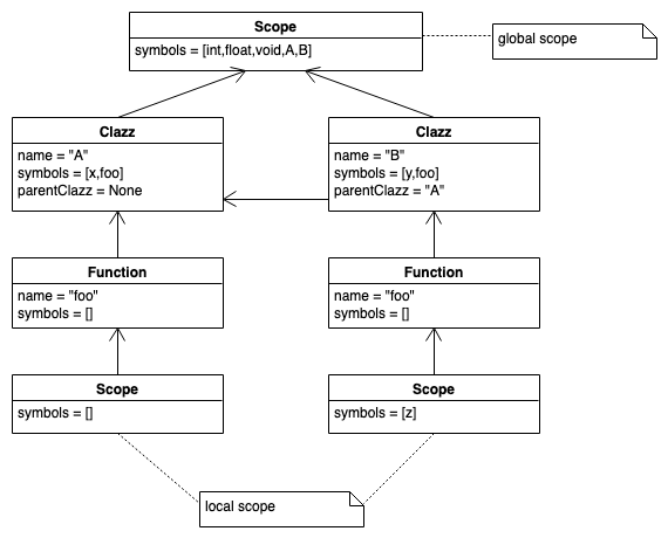

Quelle: Eigene Modellierung nach einer Idee in (Parr 2010, 162)

# Klassen

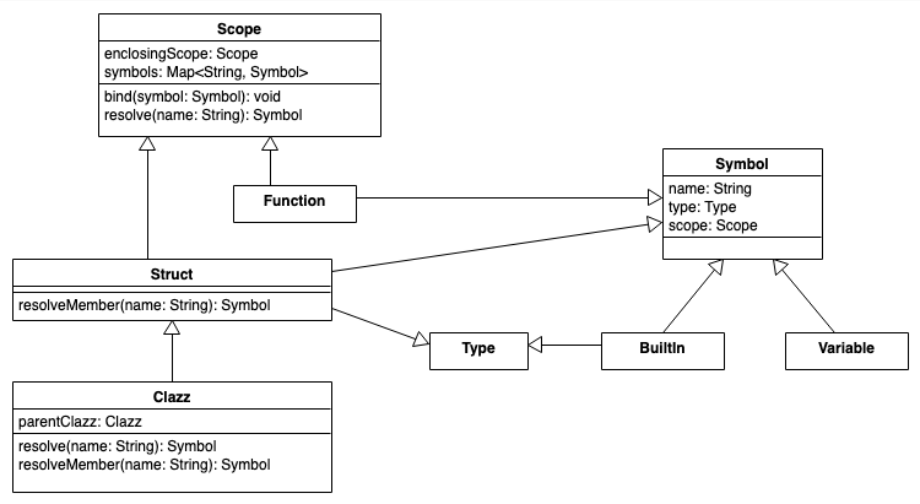
```
class A {
public:
    int x;
    void foo() { ; }
};
class B : public A {
public
    int y;
    void foo() {
        int z = x+y;
    }
};
```

# Klassen

```
class A {
public:
    int x;
    void foo() { ; }
};
class B : public A {
public
    int y;
    void foo() {
        int z = x+y;
    }
};
```



# Klassen: Erweiterung der Symbole und Scopes




Quelle: Eigene Modellierung nach einer Idee in (Parr 2010, 167)

# Klassen: Auflösen von Namen

```
class Clazz(Struct):
    Clazz parentClazz   # None if base class

    def resolve(name):
        # do we know "name" here?
        if symbols[name]: return symbols[name]
        # NEW: if not here, check any parent class ...
        if parentClazz != None: return parentClazz.resolve(name)
        # ... or enclosing scope if base class
        try: return enclosingScope.resolve(name)
        except: return None     # not found

    def resolveMember(name):
        if symbols[name]: return symbols[name]
        # NEW: check parent class
        try: return parentClazz.resolveMember(name)
        except: return None
```

Quelle: Eigene Implementierung nach einer Idee in (Parr 2010, 172)

# Wrap-Up

- Symboltabellen: Verwaltung von Symbolen und Typen (Informationen über Bezeichner)
- Strukturen und Klassen bilden eigenen Scope
- Strukturen/Klassen lösen etwas anders auf: Zugriff auf Attribute und Methoden

# LICENSE