Aufgabe A5.3 – Symboltabelle: Referenzierungen und Funktionsaufrufe

November 21, 2025

Aufgabe A5.3 – Symboltabelle: Referenzierungen und Funktionsaufrufe

Zielsetzung

In dieser Aufgabe geht es darum, wie Bezeichner (z.B. Variablen) und Funktionsaufrufe in der Symboltabelle korrekt aufgelöst und referenziert werden. Dies beinhaltet:

- Auflösen von Funktionsaufrufen,
- Zuordnung der Parameter bei Funktionsaufrufen,
- Auflösung von Variablenreferenzen in verschachtelten Scopes.

1. Referenzierung von Variablen

Die Referenzierung von Variablen erfolgt durch die Methode `resolve()` in der Symboltabelle. Wenn das Symbol im aktuellen Scope nicht gefunden wird, wird der übergeordnete Scope durchsucht.

```
public class SymbolTable {
    private final Map<String, Symbol> symbols;
    private final SymbolTable enclosingScope;

    public SymbolTable(SymbolTable enclosingScope) {
        this.symbols = new HashMap<>();
        this.enclosingScope = enclosingScope;
    }

    public void bind(Symbol symbol) {
        if (symbols.containsKey(symbol.getName())) {
            throw new SemanticException("Symbol " + symbol.getName() + " is
        }
        symbols.put(symbol.getName(), symbol);
    }

    public Symbol resolve(String name) {
```

```
        Symbol symbol = symbols.get(name);
        if (symbol != null) {
            return symbol;
        }
        if (enclosingScope != null) {
            return enclosingScope.resolve(name);
        }
        return null; // Symbol nicht gefunden
    }
}
```

2. Funktionsaufrufe und Parameterzuweisung

Bei Funktionsaufrufen werden die Argumente mit den Parametern der Funktion abgeglichen. Wenn die Anzahl der Argumente und Parameter nicht übereinstimmen oder die Typen inkompatibel sind, wird eine Ausnahme geworfen.

```
public class FunctionSymbol extends Symbol {
    private final Type returnType;
    private final List<VariableSymbol> parameters;

    public FunctionSymbol(String name, Type returnType, List<VariableSymbol
        super(name);
        this.returnType = returnType;
        this.parameters = parameters;
    }

    public Type getReturnType() {
        return returnType;
    }

    public List<VariableSymbol> getParameters() {
        return parameters;
    }
}
```

3. Beispiel für die Handhabung von Funktionsaufrufen und Variablenreferenzen

Die folgende Methode zeigt, wie Funktionsaufrufe aufgelöst und mit Parametern abgeglichen werden können:

```
@Override
public AstNode visitFnCallExpr(MiniCParser.FnCallExprContext ctx) {
    String functionName = ctx.ID().getText();
    List<Expr> arguments = new ArrayList<>();

    if (ctx.args() != null) {
        for (MiniCParser.ExprContext exprCtx : ctx.args().expr()) {
```

```
            arguments.add((Expr) visit(exprCtx));
        }
    }

    // Funktionssymbol aus dem Scope aufl sen
    FunctionSymbol functionSymbol = (FunctionSymbol) currentScope.resolve(f
    if (functionSymbol == null) {
        throw new SemanticException("Function '" + functionName + "' is not d
    }

    // Argumente mit Parametern abgleichen
    List<VariableSymbol> params = functionSymbol.getParameters();
    if (arguments.size() != params.size()) {
        throw new SemanticException("Function '" + functionName + "' expects
                                    params.size() + " arguments, but " + a
    }

    for (int i = 0; i < arguments.size(); i++) {
        Expr argument = arguments.get(i);
        VariableSymbol paramSymbol = params.get(i);
        // Argument und Parameter abgleichen (z.B. Typ berpr fung)
        if (!argument.getType().equals(paramSymbol.getType())) {
            throw new SemanticException("Argument type mismatch: expected "
                                        paramSymbol.getType() + " but got
        }
    }

    // Funktionsaufruf erfolgreich verarbeiten
    return new CallExpr(functionName, arguments);
}
```