Systemnahes Programmieren

Dokumentation

Teil 1

WS 2016/17

Boris Bopp (Matr. Nr: 53690)

Henrik Erhart (Matr. Nr: 54323)

Maksim Sunjajkin (Matr. Nr: 49946)

Vorwort

Ziel dieser Übung war es die Funktion eines Scanners als auch die eines Compilers kennenzulernen und die vorhandenen C/C++ Kenntnisse zu vertiefen.

Ein Compiler ist ein Programm das eine gegebenen Sprache in eine andere übersetzt.

Zum Beispiel: C/C++ -> Maschinensprache

Im ersten Teil unserer Übung beschäftigen wir uns mit der Lexikalischen Analyse. Das heißt mit der Zerlegung des Quellcodes in seine Bestandteile und einer darauffolgenden Erstellung einer Zwischendarstellung.

Inhaltsverzeichnis

Architektur

Die im ersten Teil zu bearbeitenden Teile des Compilers sind folgende:

- Buffer: Zuständig für die Eingabe, der Puffer besitz eine maximallänge und Speichert einen Teil der Eingabe zwischen.

- Automat: Der Automat ist bei der Token Erzeugung für die Erkennung des Typen zuständig, er besitzt einen Zustand der je nach eingelesenen Zeichen sich ändern kann.

- Scanner: Der Scanner verbindet Puffer, Automat und Symboltabelle. Er nimmt solange Zeichen vom Puffer entgegen und gibt sie an den Automaten weiter bis er ein gültiges Token erstellen kann, die daraus resultierenden Zustände speichert er in der Symboltabelle

- Tokens: Mit Tokens werden die Grundsymbole bezeichnet.

- Symboltabelle: In der Symboltabelle werden die Informationen die in Verbindung zu dem Token stehen gespeichert

Verzeichnisse und Dateien

```
|--     Automat/
|       |--     includes/
|       |--     src/
|       |--     makefile
|--     Buffer/
|       |--     includes/
|       |--     src/
|       |--     makefile
|--     Common/
|       |--     includes/
|       |--     src/
|       |--     makefile
|--     Scanner/
|       |--     includes/
|       |--     src/
|       |--     makefile
|--     Symboltable/
|       |--     includes/
|       |--     src/
|       |--     makefile
|--     makefile
```

Das Projekt wurde in 4 Teilprojekte unterteilt: Automat, Buffer, Scanner und Symboltable.

Jedes Teilprojekt enthält sowohl ein eigenes makefile als auch eine main-Klasse, um unabhängig deren Funktionalität testen zu können.

Makefiles werden benötigt um das Kompilieren und Linken automatisiert ausführen zu können.

Im Verzeichnis Common befinden sich selbst erstellte Libraries die in die Teilprojekte mit einbezogen werden. Dieses Projekt wurde angelegt um zirkuläre Referenzen zu vermeiden.

Um das Gesamtprojekt zu Kompilieren wird das makefile der obersten Ebene ausgeführt.

Code Dokumentation

Buffer

Wenn nicht anders vorgegeben wird ein Buffer mit einer Größe von 1024 Zeichen erstellt. Dieser wird mithilfe der Library *fstream* befüllt. Es werden jedoch jeweils nur [Buffergröße - 1] Zeichen kopiert damit man im Falle eines Dateiendes bei [Buffergröße - 1]ten Zeichen noch ein Eof Indikator einfügen kann.

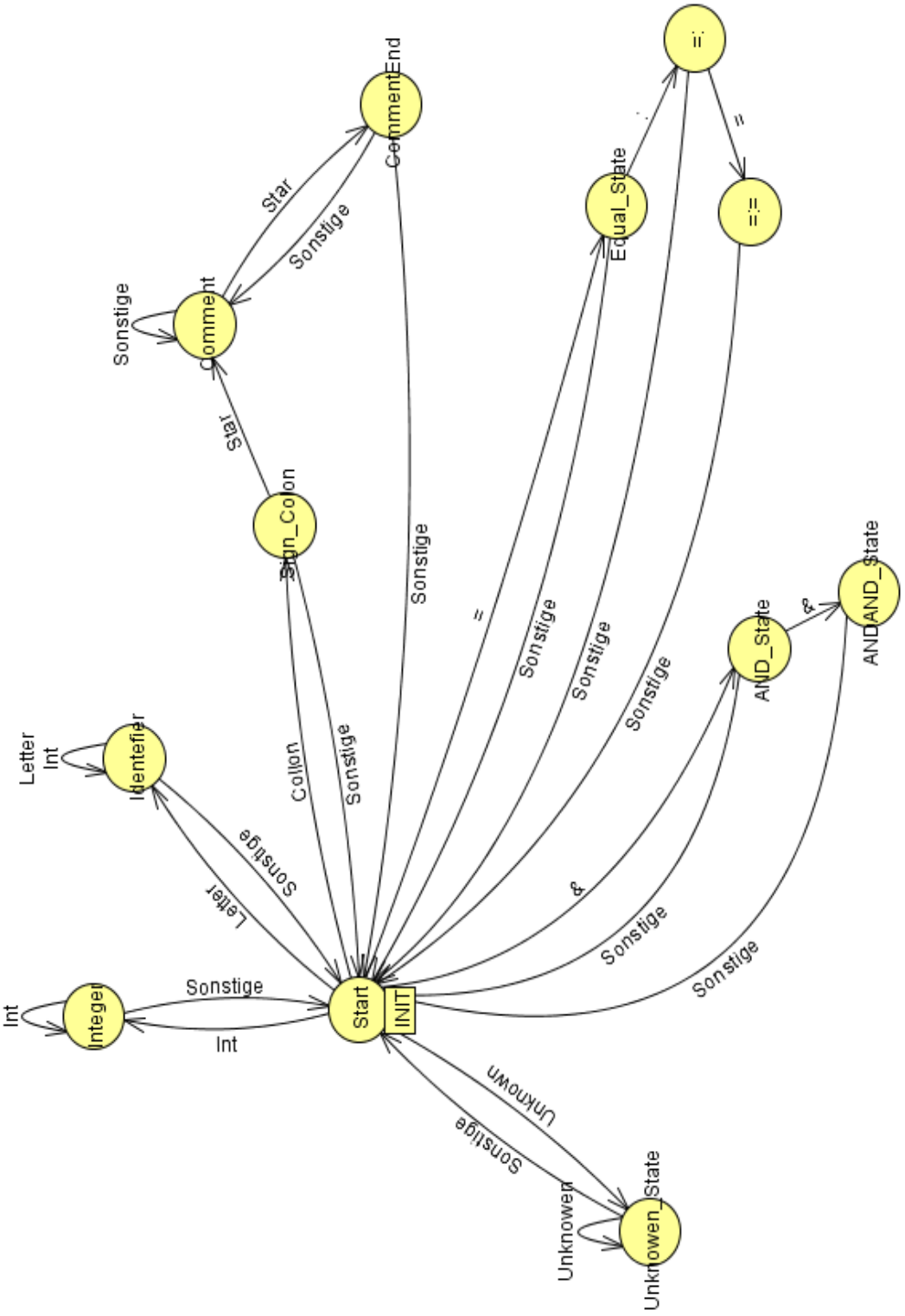
Der Buffer wird zusätzlich als Node in einer Verketteten Liste gespeichert, was uns erlaubt Zeichen aus allen vorherigen Bufferblöcken zu lesen.

Scanner

Der Scanner liest mit Hilfe des Buffers die Zeichen einzeln ein und gibt sie an den Automaten zur Verfolgung der Zustände weiter. Der Automat bestimmt ihren Typ und gibt ihn an den Scanner. Mit dieser Information kann der Scanner ein Token erstellen und es hinzufügen, falls es dieses noch nicht gibt. Sollte das Token schon existieren wird nur die Lexem Information (Wert, Position) hinzugefügt.

Automat

Folgendes Bild zeigt die Zustände und Übergänge des Programmierten Automaten



Symboltable

Die Symboltabelle dient zur Speicherung von Zusatzinformationen zu Identifiern. Somit kann beispielsweise überprüft werden, ob eine Variable bereits zuvor deklariert wurde, was eine ungültige Operation darstellen würde. Dabei stellt der Identifier den Schlüssel (das Suchkriterium) dar.

Um diese Funktion zu realisieren erfordert die Datensammlung schnelle Zugriffszeit zum Finden der Identifier. Dazu eignet sich bestens eine Hashmap, in der die Symboltabelle auch implementiert wurde.

Schlüsselwörter

Zur Erkennung von Schlüsselwörtern, wird die Symboltabelle beim Konstruktor Aufruf mit allen bekannten Schlüsselwörtern initialisiert. Dies vereinfacht dem Scanner die Erkennung der Schlüsselwörter.

Stringtable

Wie vorgegeben, werden die Lexeme der Tokens in einer Stringtabelle angelegt und verwaltet. Dies führt theoretisch zu einem besseren Laufzeitverhalten, da wenige große Speicheranforderungen schneller abgearbeitet werden können als viele kleine Anforderungen der insgesamt gleichen Größe. Besondere Relevanz der Optimierung konnten wir nicht feststellen, jedoch wurde beim Gespräch mit dem Betreuer des Labors vorgegeben, dass die Stringtabelle verpflichtend zu implementieren sei, um die Erstellung von Datenstrukturen zu üben.

Aufgetretene Probleme und Lösungen

Zur ersten Abgabe hatte das Programm drei Fehler, diese wären:

1. Das Token „=:“ wurde als „Unknown“ deklariert
   Lösung: Im Automaten war ein Übergang falsch beschrieben -> Verbesserung am Automaten

2. Das „EoF“-Token wurde nicht erstellt, wenn vor dem Ende der Datei keine Leerzeile bestand.
   Lösung: Wir fügen nun eine Leerzeile ein sollte es zu Schluss keine geben.

3. Zeilen die während einem Kommentar eingelesen wurden, wurden nicht mitgezählt.
   Lösung: Anpassung eines fehlerhaften Überganges im Automaten.