Systemnahes Programmieren

Michael Bührer 43279, Stefan Fink 43446, Janet Do 47486

May 28, 2016

Contents

1 Aufgabenstellung

Es soll die Funktionsweise eines Scanners sowie die Einordnung innerhalb eines Compilers verstanden werden. Ziel ist es einen funktionieren Compiler zu programmieren. Die Funktionalitäten eines Compilers liegen grundsätzlich im Einlesen einer Datei, verarbeiten des eingelesenen Inputs und deren Auswertung. Die Auswertung besteht darin den Code zu zerteilen und in diesem nach vorgegeben Bezeichner, Grundsymbolen, Zahlen etc. auch Tokens genannt, zu filtern. Kommen Symbole vor, die nicht in unserem Vokabular existieren, so werden sie als ungültig gewertet. Hierfür ist der Zustandsautomat verantwortlich. Wurden nun alle Tokens gefunden, so muss ihre Reihenfolge nach einer korrekten Grammatik überprüft werden. Dies übernimmt der Scanner.

2 Erläuterung

Die Einlese-Komponente wird später der Buffer. Der Buffer gibt die eingelesenen Zeichen an die Hauptkomponente, den Scanner weiter. Der Scanner hat Zugriff auf den Automaten, Buffer und Symboltabelle. Diese Komponente steuert alles. Der Scanner steuert den Buffer an, wenn er mehr Zeichen braucht, gibt diesen an den Automaten weiter, damit dieser sich immer im korrekten Zustand befindet und überprüft permanent, ob ein Token erkannt wurde. Die Symboltabelle beinhaltet unsere Tokens.

3 Scanner

3.1 Buffer

1. Konstruktor
   Im Konstruktor wird die Datei mittels des übergebenen Pfades eingelesen und auf fileDescriptor gespeichert. Ansonsten werden nur die notwendigen Erstinitialisierung, Bufferfüllung und Speicherallozierung vorgenommen.

```
Buffer::Buffer(const char *path) {
    #ifdef NODIRECT
        if ((this->fileDescriptor = open(path, O_RDONLY)) < 0) {
    #else
     if ((this->fileDescriptor = open(path, O_RDONLY | O_DIRECT)) < 0) {
    #endif
                perror("Öffnen fehlgeschlagen");
                exit(-1);
        }
allocMem(&memptr1, BUFFERSIZE, BUFFERSIZE + sizeof(char));
this->buffer1 = (char*)this->memptr1;
this->buffer1[BUFFERSIZE] = '\0';
this->totalBufferIndex = this->currentBufferIndex = 0;
this->column = this->oldColumn = 1;
this->line = 1;
this->atEof = false;

if (this->readBytes(this->buffer1, BUFFERSIZE) == 0) {
        fprintf(stderr, "Datei ist leer.\n");
        exit(-1);
}
        this->currentBuffer = &this->buffer1;
```

```
}
```

2. Diese Methode alloziert den Speicher

```
inline void Buffer::allocMem(void **mempt, int align, int size) {
        if (posix_memalign(mempt, align, size) != 0) {
                perror("Speicher allozieren fehlgeschlagen");
                exit(-1);
        }
}
```

3. Destruktor

```
Buffer::~Buffer(){
        if (this->fileDescriptor > 0) close(this->fileDescriptor);
        free(this->memptr1);
        free(this->memptr2);
}
```

4. Methode berechnet die Anzahl der eingelesenen Bytes mit Hilfe des fileDescriptor und gibt sie zurück.

```
int Buffer::readBytes(char* buf, int count) {
        if (fileDescriptor > 0) {
                int bytesRead = read(this->fileDescriptor, buf, count);
                if (bytesRead < 0) {
                        perror("Fehler beim lesen");
                        exit(-1);
                }
                if (bytesRead < BUFFERSIZE) buf[bytesRead] = '\0';
                return bytesRead;
        }
        return -1;
}
```

5. Holt ein Zeichen aus dem aktuellen Buffer, überprüft davor ob wir am Ende der Zeile angekommen sind. Am Ende
nach dem erfolgreichen Lesen werden die Indizes hochgezählt.

```
char Buffer::getChar() {
    if (this->atEof) return '\0';
        char curr = getCharFromCurrentBuffer();

        if (curr == LF) {
                this->oldColumn = this->column;
                this->column = 1;
                this->line++;
        } else this->column++;

    if (curr == '\0') {
        this->atEof = true;
    }
```

```
        this->currentBufferIndex++;
        this->totalBufferIndex++;

        return curr;
}
```

6. Diese Methode wird in der vorherigen, getChar()-Methode aufgerufen. Zu Beginn wird überprüft, ob der 1. BufferIndex überhaupt schon gesetzt wurde, falls nicht soll dies gemacht werden. Weiterhin weist der negative BufferIndex daraufhin, dass wir weiter "zurück" gehen soll im Dokument, wir aber uns bereits am Bufferanfang befinden.
   Ein mögliches Szenario: Buffer 1 wurde geladen und gelesen, ein Lexem beginnt am Ende des Buffer 1, Buffer 2 wird befüllt. Nun wird ein Fehler in Buffer 2 gemerkt, aber da der Anfang von dem Lexem sich in Buffer 1 befindet, muss dieser wieder geladen werden.
   Ist der BufferIndex bereits gefüllt und größer als die vorgegebene BUFFERSIZE, heißt dies, dass es noch mehr zu lesen gibt, d.h. der nächste Buffer soll befüllt werden.

```
char Buffer::getCharFromCurrentBuffer() {

        if (this->currentBufferIndex < 0) {
                if (this->currentBuffer == &this->buffer1) {
                        this->currentBuffer = &this->buffer2;
                }
                else if (this->currentBuffer == &this->buffer2) {
                this->currentBuffer = &this->buffer1;
                }
                this->currentBufferIndex = BUFFERSIZE-1;
        } else if (this->currentBufferIndex >= BUFFERSIZE) {
                if (this->currentBuffer == &this->buffer1) {
                        if (!this->memptr2) {
                        allocMem(&this->memptr2, BUFFERSIZE, BUFFERSIZE
                        + sizeof(char));
                                this->buffer2 = (char*)this->memptr2;
                                this->buffer2[BUFFERSIZE] = '\0';
                        }
                        this->readBytes(this->buffer2, BUFFERSIZE);
                        this->currentBuffer = &this->buffer2;
                }
                else if (this->currentBuffer == &this->buffer2) {
                        this->readBytes(this->buffer1, BUFFERSIZE);
                        this->currentBuffer = &this->buffer1;
                }
                this->currentBufferIndex = 0;
        }
        return (*this->currentBuffer)[this->currentBufferIndex];
}
```

7. Diese Methode geht ein Zeichen im Buffer zurück

```
                void Buffer::ungetChar() {
        if (this->totalBufferIndex == 0) {
                fprintf(stderr, "Kann nicht weiter zurückgehen.\n");
                exit(-1);
        }
```

```
        this->currentBufferIndex--;
        this->totalBufferIndex--;
        char curr = this->getCharFromCurrentBuffer();
        if (curr == LF) { // dann ans Ende der vorherigen Zeile
                this->column = this->oldColumn;
                this->line--;
        } else this->column--;
```

8. Getter-Methoden

```
int Buffer::getBuffersize() {
    return BUFFERSIZE;
}

int Buffer::getCol() {
        return this->column;
}

int Buffer::getLine() {
        return this->line;
}

void Buffer::ungetChar(int count) {
        for (int i = 0; i < count; i++) {
                this->ungetChar();
        }
}
```

3.1.1 Erklärung

3.2 Automat

3.3 Scanner

3.4 Symboltabelle