Übungsblatt 1

Wichtig: Anmeldung im EST

Zur Teilnahme an den Übungen ist eine Anmeldung im EST (https://est.cs.fau.de) **unbedingt notwendig**.

Wichtig: Meilensteine

Die **erfolgreiche Bearbeitung** der Projektaufgaben ist **notwendiger Bestandteil der Modulleistung**. Im Laufe des Semesters wird es deshalb (voraussichtlich) fünf **Meilensteine** (als solche auf den Übungsblättern gekennzeichnet) geben, die Ihr Compiler erfolgreich bestehen muss. Bei nicht erfolgter oder erfolgloser Teilnahme an den Übungen melden Sie sich bitte *nicht* zur Prüfung an bzw. melden Sie sich *rechtzeitig* wieder von der Prüfung ab. Andernfalls wird das gesamte Modul als „durchgefallen“ gewertet!

Übungen nächste Woche

Nächste Woche (KW44, 30.10. – 03.11.) finden auf Grund der beiden Feiertage **keine** Übungen statt. Es wird dennoch ein **Übungsblatt** geben, das eine **Projektaufgabe** (Konstantenfaltung) beinhaltet.

Aufgabe 1.1: Phasen der Übersetzung

Aus welchen *Phasen* besteht ein Übersetzungsvorgang? Was ist die jeweilige Ein- und Ausgabe der Phasen?

Zu welchen Zeitpunkten des Übersetzungsprozesses können welche *Fehler* auftreten?

Aufgabe 1.2: Anforderungen an einen Übersetzer

Welche funktionalen und nicht-funktionalen *Anforderungen* werden an einen Übersetzer gestellt?

Aufgabe 1.3: Bootstrapping

Nach diesem Semester haben Sie bestimmt Ideen für eine neue, besonders tolle Sprache `L`. Natürlich wollen Sie Ihren `L`-Übersetzer auch in `L` implementieren – schließlich ist `L` besser als alle Sprachen zuvor ☺. Wie können Sie mit möglichst wenig Aufwand einen (effizienten) Übersetzer für `L` in `L` implementieren?

Aufgabe 1.4: Spezifikation der Sprache `e2`

Auf der Internetseite zur Übung finden Sie eine Sprachspezifikation für die kleine Programmiersprache `e2`, für die Sie im Rahmen der Projektübungen einen Übersetzer implementieren sollen.

a) Lesen Sie die Sprachbeschreibung und suchen Sie die zu den darin erläuterten Sprachkonstrukten gehörenden Produktionen in der beigefügten Grammatik.

b) Schreiben Sie ein `e2`-Programm, das den Benutzer nach einer Ganzzahl n fragt und anschließend die n-te Fibonacci-Zahl berechnet und ausgibt.

c) Umgangssprachliche (nicht-formale) Spezifikationen von Programmiersprachen sind häufig unklar oder unvollständig. Suchen Sie in der `e2`-Spezifikation nach Punkten, die nur unzureichend spezifiziert sind.

Verwendung von `git`

Die Abgabe der Lösungen erfolgt mittels `git`. Falls Sie `git` noch nie verwendet haben oder auf Schwierigkeiten stoßen, finden Sie z.B. unter http://www.vogella.com/tutorials/Git/article.html eine Einführung.

Projektübung 0 (Abgabe bis -)

Im Rahmen dieser Projektübung setzen Sie Ihr Arbeits-Repository auf und machen sich mit der Vorlage für die praktischen Übungen vertraut. Bitte arbeiten Sie die folgenden Teilaufgaben Schritt für Schritt durch.

a) Falls noch nicht geschehen: Melden Sie sich im EST zu der Veranstaltung *„Grundlagen des Übersetzerbaus (WS 17/18)“* an. Eine Anmeldung ist **unbedingt notwendig** für die Teilnahme am Übungsbetrieb!

Wichtig: Tragen Sie bei der Anmeldung die Matrikelnummer Ihres Übungspartners in das Feld *„Matrikelnummern“* ein (**Hinweis**: Sie können Ihren Übungspartner auch nachträglich über *„WS 17/18“* → *„Grundlagen des Übersetzerbaus“* → *„Anmeldeoptionen“* angeben). Alternativ können Sie auch die (exakte!) E-Mail-Adresse eintragen, mit der sich Ihr Partner im EST registriert hat. Stellen Sie sicher, dass Ihr Übungspartner auch Sie als Partner eingetragen hat!

b) Falls noch nicht (ggf. für eine andere Veranstaltung) geschehen: Richten Sie im EST einen sog. *„Externen Account“* ein. Gehen Sie dazu auf *„WS 17/18“* → *„Grundlagen des Übersetzerbaus“* → *„Externe Accounts“* und geben Sie Ihren Benutzernamen und Ihr Passwort ein. Der Benutzername wird für den Zugriff auf die *GitLab*-Instanz des Lehrstuhls verwendet.

Sie sollten innerhalb eines Tages eine E-Mail mit der Aufforderung erhalten, ein Passwort für *GitLab* zu setzen. Erst danach kann Ihr Benutzerkonto verwendet werden.

c) Melden Sie sich unter https://faui22m.informatik.uni-erlangen.de mit Ihrem *GitLab*-Benutzer an. Laden Sie über die Weboberfläche Ihren öffentlichen SSH-Schlüssel hoch, damit Sie anschließend auf Ihr Abgabe-Repository zugreifen können.[1]

d) Ihr Abgabe-Repository finden Sie unter folgender URL:

- https://faui22m.informatik.uni-erlangen.de/ue1-ws17/est_ab12cdef_xy98qwer

Dabei stehen `ab12cdef` und `xy98qwer` für Ihren und den Login Ihres Übungspartners (wie bei der Erstellung der *„Externen Accounts“* angegeben, siehe oben), in *alphabetischer Reihenfolge*.

Wichtig: Die Erstellung Ihres Abgabe-Repository muss von uns manuell angestoßen werden. Falls dies nicht zeitnah geschehen ist, melden Sie sich bitte per E-Mail bei uns.

e) Sobald Ihr Abgabe-Repository erstellt wurde und Sie Ihren SSH-Schlüssel zum Zugriff darauf eingerichtet haben, gehen Sie wie folgt vor, um Ihre lokale Kopie zu erzeugen und die Vorlage in diese zu kopieren:

- Klonen Sie Ihr eigenes Arbeits-Repository in ein Verzeichnis Ihrer Wahl (URL wie oben ersetzen):

```
$> git clone git@faui22m.informatik.uni-erlangen.de:ue1-ws17/est_ab12cdef_xy98qwer
```

- Wechseln Sie in das Verzeichnis mit dem geklonten Repository und holen Sie die Änderungen aus dem Vorlagen-Repository in Ihren lokalen Klon:

```
$> cd est_ab12cdef_xy98qwer
$> git pull git@faui22m.informatik.uni-erlangen.de:e2c/skeleton
```

- Pushen Sie Ihren lokalen Klon in Ihr Abgabe-Repository auf dem `git`-Server des Lehrstuhls:

```
$> git push -u origin master
```

[1] Falls Sie noch keinen SSH-Schlüssel erstellt haben, so finden Sie unter https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/ eine kurze Anleitung dazu.

Ab sofort können Sie mittels `git pull` Änderungen aus Ihrem Abgabe-Repository in Ihren lokalen Klon kopieren bzw. mittels `git push` Änderungen aus Ihrem lokalen Klon in Ihr Abgabe-Repository pushen.

f) **Wichtig**: Editieren Sie die bereits vorgegebene Datei `README.md`. Tragen Sie dort *unbedingt* die fehlenden Informationen zu den Gruppenmitgliedern (Name, Matrikelnummer, E-Mail-Adresse) ein. Dokumentieren Sie in dieser Datei etwaige Besonderheiten Ihrer Implementierung sowie die Aufrufkonventionen Ihres Übersetzers, sofern diese von der Vorlage bzw. den Angaben in den Aufgabenstellungen abweichen.

g) Wir verwenden *Gradle* zum Übersetzen und Testen des Übersetzers sowie zur Verwaltung aller Abhängigkeiten. Gradle bietet mit dem sog. *Gradle Wrapper* eine elegante Möglichkeit an, eine lokale Installation von Gradle in der richtigen Version einzurichten und zu verwenden. Führen Sie dazu zunächst folgenden Befehl auf der Kommandozeile aus:

```
./gradlew
```

Beim ersten Start wird die korrekte Gradle-Distribution heruntergeladen und eingerichtet. Am Ende sollte die Meldung „`BUILD SUCCESSFUL`“ erscheinen.[2]

h) Übersetzen Sie die Vorlage mit Hilfe des *Gradle Wrappers*:

```
./gradlew build
```

Nach dem Übersetzen sollte erneut die Meldung „`BUILD SUCCESSFUL`“ erscheinen. Sie können nun den Übersetzer mit Hilfe des Skripts `e2c` starten und dabei die zu übersetzende Eingabedatei angeben:

```
./e2c examples/ttt.e2
```

Da die Implementierung noch nicht vollständig ist, wird die Übersetzung je nach Eingabedatei mit einer Fehlermeldung abbrechen oder schlichtweg keine Ausgabe erzeugen. Wenn Sie den Übersetzer ohne weitere Kommandozeilenargumente starten, wird Ihnen die Liste der Kommandozeilenoptionen angezeigt.

Hinweis: Weitere Informationen zur Verwendung des Übersetzers finden Sie in der Datei `README.md`. Darin wird erklärt, wie Sie die Tests ausführen oder Style-Checks auf Ihre Implementierung anwenden können.

i) Der Einstiegspunkt für den Übersetzer befindet sich in der Klasse `e2c.E2Compiler`. Arbeiten Sie sich von dort aus in die grobe Struktur des Übersetzers ein und versuchen Sie, das Zusammenspiel der einzelnen Komponenten zu verstehen. Schauen Sie sich auch an, wie die übergebenen Kommandozeilenoptionen behandelt werden (`e2c.util.ArgsHandler`) und wie der Übersetzungsvorgang gesteuert wird (`e2c.util.CompilerOptions`).

Hinweis: Verstehen Sie die Vorlage als eine Art „Starthilfe“, die Ihnen den Einstieg erleichtert und einige Arbeit abnimmt. Sie dürfen Teile der Vorlage ändern – achten Sie dann jedoch darauf, dass *alle* Testklassen weiterhin übersetzen. Dies können Sie beispielsweise mit Hilfe des Gradle-Task „`compileAll`“ überprüfen. Fügen Sie *keine* fremden Bibliotheken hinzu und ändern Sie *nicht* die Versionsnummern der vorgegebenen Bibliotheken.

Viel Spaß bei der Bearbeitung der Projektaufgaben ☺

Projektübung 1 (Abgabe bis 19.11.2017)

Hinweis: Zur Bearbeitung dieser Aufgabe benötigen Sie Wissen aus der *zweiten* Vorlesung. Sie sollten deshalb ggf. mit der Bearbeitung bis nach der zweiten Vorlesung warten.

Im Rahmen dieser Projektübung vervollständigen Sie die Grammatik sowie die Konstruktion des Abstrakten Syntaxbaums (AST). Des Weiteren implementieren Sie eine Ausgabe des ASTs im Dot-Format.

[2] Konfiguriert wird das Gradle-Projekt über die Datei `build.gradle`. Im Normalfall müssen Sie diese Datei *nie* ändern.

a) In der vorgegebenen *Antlr*-Grammatik (`src/main/antlr/e2c/antlr/E2.g4`) fehlen noch manche Produktionen, sodass manche Sprachkonstrukte noch nicht geparst werden können:

- Für `return`-Anweisungen und Deklarationen von Funktionsparametern existieren bereits AST-Klassen (`ReturnStatement` bzw. `ParameterDeclaration`), aber die Regeln fehlen in der Grammatik.
- Für `while`-Schleifen und Array-Zugriffe fehlen sowohl die Regeln in der Grammatik als auch Klassen zur Repräsentation dieser Konstrukte im AST (`WhileStatement` bzw. `ArrayAccess`).

Ergänzen Sie die fehlenden Teile. Achten Sie darauf, dass Sie die fehlenden Klassen wie angegeben benennen und in diesen insbesondere auch die Methode `getChildren()` korrekt implementieren (für die Tests zum ersten Meilenstein relevant). Durch das Hinzufügen der neuen Produktionen müssen Sie auch die Klasse `e2c.frontend.parser.builder.ASTBuilder` erweitern, in der die Knoten des ASTs erzeugt werden. Orientieren Sie sich dabei an den bereits implementierten Methoden für die übrigen Produktionen.

b) Das Interface `e2c.frontend.visitors.ASTVisitor` definiert die Schnittstelle für das später zu verwendende *„Besucher“*-Entwurfsmuster. Erweitern Sie diese für die in Teilaufgabe a) neu angelegten AST-Klassen.

c) Die Klasse `e2c.frontend.visitors.BaseASTVisitor` dient als Basis für die später zu implementierenden Besucher und traversiert die Knoten des ASTs, ohne dabei weitere Aktionen auszuführen. Passen Sie diese Klasse an die von Ihnen hinzugefügten und erweiterten (!) Klassen an. Orienteren Sie sich an den bereits implementierten Methoden und verwenden Sie die Methoden `prolog()`, `epilog()` und `visitChild()`.

d) Zur Visualisierung eines ASTs bietet sich das Programm `dot` an.[3] Implementieren Sie dazu einen Besucher `DotGenerator` im Paket `e2c.frontend.ast.visitors`, der zu einem gegebenen AST eine Dot-Beschreibung generiert und in eine Datei schreibt. Die Ausgabe soll nur dann passieren, wenn die Kommandozeilenoption „`--dot`“ gesetzt ist (die Behandlung der Kommandozeilenoption ist bereits implementiert, siehe `e2c.util.ArgsHandler` und `e2c.util.CompilerOptions`).

Hinweis: Fügen Sie Ihrer Dot-Beschreibung folgende Zeile hinzu, damit die Reihenfolge der Kinder in der Darstellung der Reihenfolge im AST entspricht:

```
graph[ordering="out"];
```

Tipp: Die Klasse `e2c.util.FileUtil` stellt einige nützliche Methoden zum Öffnen, Schreiben und Lesen von Dateien zur Verfügung.

e) **Optional**: Schreiben Sie einen Besucher `PrettyPrinter`, der zu einem gegebenen AST wieder `e2`-Code generiert und in eine Datei schreibt. Fügen Sie eine zusätzliche Kommandozeilenoption hinzu, um die Ausgabe mit Hilfe des Besuchers zu aktivieren.

Hinweis: Achten Sie darauf, dass bei komplexen (arithmetischen und logischen) Ausdrücken an den richtigen Stellen Klammern verwendet werden (Stichwort: Operatorpräzedenz)!

Wenn Sie die Implementierung dieser Projektaufgabe abgeschlossen haben, sollten alle Beispielprogramme im Verzeichnis `examples/` erfolgreich geparst werden können. Des Weiteren sollten alle zum ersten Meilenstein gehörenden Tests erfolgreich ausgeführt werden können (siehe unten).

Meilenstein 1 (Abgabe bis 19.11.2017)

Stellen Sie sicher, dass Ihr Übersetzer alle zum ersten Meilenstein gehörenden Testfälle korrekt behandelt, indem Sie den Gradle-Task „`milestone1`“ ausführen und kontrollieren, dass am Ende die Ausgabe „`BUILD SUCCESSFUL`“ erscheint. Versehen Sie den finalen Stand mittels „`git tag`“ mit der Markierung „`milestone1`“ und pushen Sie Ihre Änderungen auf den `git`-Server des Lehrstuhls.

[3] Hinweise zur Verwendung von `dot` und Beispiele, wie eine Ausgabe aussehen könnte, finden Sie auf der Internetseite zur Übung.