Compilerprojekt: Installation und Verwendung des Frameworks

Der Leistungsnachweis im Modul COBAU ist die Erstellung eines Compilers für die Sprache MiniJ. Diesen Compiler erstellen Sie in einem Projekt bestehend aus vier Meilensteinen. **Für dieses Compilerprojekt müssen Sie das von uns bereitgestellte Framework verwenden.** Dieses Dokument beschreibt in den nachfolgenden Abschnitten schrittweise, wie Sie dieses Framework installieren und verwenden.

1. Java installieren

Wir entwickeln unseren Compiler in Java. Sie sollten *mindestens* die Version 21 installiert haben. Dies ist die gleiche Version, welche Sie auch im Modul OOP verwendet haben. Höhere Java-Versionen sollten jedoch ebenfalls problemlos funktionieren.

Das Java SE Development Kit (JDK) 21 können Sie bei Bedarf unter folgender Adresse beziehen: https://www.oracle.com/java/technologies/downloads/

Um zu testen, ob das aktive JDK mindestens Version 21 hat, geben Sie im Terminal (Linux / Mac OS) oder in der Eingabeaufforderung (Windows) folgendes ein:

```
java -version
```

Sie sollten eine Ausgabe mit der verwendeten Version erhalten, z.B.:

```
openjdk version "21.0.4" 2024-07-16 LTS
OpenJDK Runtime Environment Temurin-21.0.4+7 (build 21.0.4+7-LTS)
OpenJDK 64-Bit Server VM Temurin-21.0.4+7 (build 21.0.4+7-LTS, mixed mode, sharing)
```

⇨ Falls trotz Installation des JDK 21 noch die falsche Version angezeigt wird, liegt dies typischerweise daran, dass die Umgebungsvariable «PATH» nicht korrekt gesetzt ist. Überprüfen Sie diese Umgebungsvariable mittels:
Windows: `echo %PATH%` oder Linux/Mac: `echo $PATH`
In der Umgebungsvariable «PATH» muss der Eintrag zum `bin` Verzeichnis der gewünschten JDK-Installation vor allen anderen JDK-Installationen stehen.

2. Compiler-Framework installieren

Laden Sie das von uns bereitgestellte Compiler-Framework `COBAU-MS1.ZIP` herunter. Sie finden dieses auf ILIAS bei den Materialien zum Compilerprojekt.

Dieses Framework basiert auf dem Gradle-Buildsystem und beinhaltet den Netwide-Assembler (NASM) in der aktuellen Version für die 64-Bit x86 Versionen der folgenden Betriebssysteme:

- Windows
- GNU/Linux
- MacOS

Neuere Macs verwenden ARM-Prozessoren (M1). Der im MacOS eingebaute x64-Emulator «Rosetta 2» kann erfahrungsgemäss mit dem Framework umgehen. Sollten dennoch Probleme auftreten, melden Sie sich bitte beim Dozenten, wir versuchen dann einen x86-Rechner oder eine VM für Sie zu organisieren.

Entpacken Sie `COBAU-MS1.ZIP` in Ihr bevorzugtes Arbeitsverzeichnis. Alle Daten sind im Verzeichnis `CompilerFw` enthalten. Nun haben Sie die Möglichkeit entweder mit der Kommandozeile (Eingabeaufforderung bzw. Terminal) zu arbeiten oder eine Entwicklungsumgebung zu installieren.

3. Ersten Build starten

Das Compiler-Framework ist nach dem Entpacken einsatzbereit. Sie können Ihren ersten Build starten.

Variante A: Kommandozeile

Wechseln Sie in das Verzeichnis `CompilerFw` und führen Sie folgenden Befehl aus:

(Windows) `gradlew build`
(Linux / MacOS) `./gradlew build`

Das Buildsystem wird sich nun selbst konfigurieren und die benötigten Bibliotheken installieren. Anschliessend können Sie Sourcen im Unterverzeichnis `src` mit einem beliebigen Editor erstellen/anpassen/löschen.

Variante B: Entwicklungsumgebung

Im Modul verwenden wir die frei verfügbare Entwicklungsumgebung IntelliJ IDEA Community Edition (https://www.jetbrains.com/idea/download), welche eine gute Integration des Buildsystems «Gradle» bietet und sowohl für ANTLR- als auch für Assembler-Sourcen Syntaxhervorhebung via Plugins bietet.

Installieren Sie IntelliJ IDEA Community Edition und anschliessend die folgenden Plugins (File -> Settings -> Plugins):

- ANTLR v4 grammar plugin
- NASM Assembly Language

Bei Verwendung von IntelliJ können Sie das Verzeichnis `CompilerFw` wie ein normales IntelliJ-Projekt öffnen. IntelliJ wird sich dann selbst konfigurieren.

4. Verwendung des Frameworks

Das Framework folgt den Konventionen des Gradle-Buildsystems. Die Gradle-Ansicht im IntelliJ gibt eine gute Übersicht über die möglichen Aktionen (in Gradle-Terminologie: Tasks). Typsicherweise verwenden Sie folgende Tasks:

- `assemble` – Erstellt alle Artefakte (Binaries, Klassen und Jars).
- `check` – Führt die Systemtests aus.
- `build` – Erstellt alle Artefakte und führt danach die Systemtests aus.
- `makeSubmission` – Erstellt ein ZIP-File, welches Sie als Abgabe ins ILIAS hochladen.

Sie können aber auch selektiv nur einzelne Teilaktionen ausführen. Eine Übersicht über die einzelnen Aktionen finden Sie in der «Gradle View» (IntelliJ -> View -> Tool Windows -> Gradle).

Das Compiler-Framework weist folgende Ordnerstruktur auf:

- `build` – Binaries (z.B. div2.exe) sowie class-Files und jar-Files.
- `reports` – HTML-Reports der Systemtests.
- `src/<name>/asm` – Alle Quellen (asm-Files) des Assembler-Binaries <name>. Nur relevant für den ersten Meilenstein. Bei Build werden alle Files in einem Verzeichnis assembliert und verlinkt. Das Binary <name> finden Sie dann direkt im `build`-Verzeichnis.
- `src/main/java` – Alle Java Quellen des MiniJ-Compilers.
- `src/main/antlr` – Alle ANTLR-Quelle des MiniJ-Compilers.

Wichtig: Machen Sie keine Veränderung am Buildsystem. Ändern Sie ausschliesslich Dateien im Verzeichnis `src`.

5. Systemtests

Im Verlauf dieses Modules schreiben Sie ausschliesslich Programme, welche auf der Kommandozeile arbeiten (keine grafische Oberfläche). Ein Hauptfeature des Compiler-Frameworks sind Systemtests, welche direkt in den Build integriert sind (Gradle-Targets: check oder build) und die von Ihnen geschriebenen Kommandozeilenprogramme mittels Eingabe und erwarteter Ausgabe überprüfen.

Die Resultate der Systemtests finden Sie im Verzeichnis `reports`. Diese Reports sind im HTML-Format. Abbildung 1 zeigt ein Beispiel eines solchen Reports.

Test results (score: 10.0 of 10.0):

Test case	Input	Expected Output	Actual Output	Score	Result	Time [ms]	Timeout [ms]
single digit (zero)	0	0	Bitte geben Sie eine Zahl ein: Das Resultat ist: 0	2.0	Passed	0	500
single digit (uneven)	3	1	Bitte geben Sie eine Zahl ein: Das Resultat ist: 1	2.0	Passed	0	500
two digits (even)	10	5	Bitte geben Sie eine Zahl ein: Das Resultat ist: 5	1.0	Passed	0	500
two digits (uneven)	42	21	Bitte geben Sie eine Zahl ein: Das Resultat ist: 21	1.0	Passed	0	500
text	10	Bitte geben Sie eine Zahl ein: Das Resultat ist: 5	Bitte geben Sie eine Zahl ein: Das Resultat ist: 5	1.0	Passed	0	500
negative	-1234	-617	Bitte geben Sie eine Zahl ein: Das Resultat ist: -617	1.0	Passed	0	500
large (positive)	12345678900	6172839450	Bitte geben Sie eine Zahl ein: Das Resultat ist: 6172839450	0.5	Passed	0	500
large (negative)	-12345678900	-6172839450	Bitte geben Sie eine Zahl ein: Das Resultat ist: -6172839450	0.5	Passed	0	500
64-bit (maximal)	9223372036854775807	4611686018427387903	Bitte geben Sie eine Zahl ein: Das Resultat ist: 4611686018427387903	0.5	Passed	0	500
64-bit (minimal)	-9223372036854775808	-4611686018427387904	Bitte geben Sie eine Zahl ein: Das Resultat ist: -4611686018427387904	0.5	Passed	0	500

Disclaimer: The calculated points serve only as an indication of the achieved points. You muss check yourself for obvious errors in the evaluation (e.g., test shows green although feature has not been implemented).Also keep in mind that your submission will be tested against a similar test set with minor differences to check for hard coded results.

Abbildung 1: Beispiel einer erfolgreichen Testdurchführung.

Jeder Systemtest besteht aus mehreren Testfällen. Pro Testfall sehen Sie jeweils:

- **Input:** Eingabe an Ihr Programm.
- **Expected Output:** Die vom Test erwartete Ausgabe.
- **Actual Output:** Die effektive Ausgabe Ihres Programmes.
- **Expected Exitcode:** Der vom Test erwartete Exitcode.
- **Actual Exitcode:** Der effektive Exitcode Ihres Programmes.
- **Score:** Die Anzahl Punkte, welche Sie bei Bestehen des Testfalls erhalten.
- **Result:** Das Resultat des Testlaufs.
  - **Passed:** Das Programm gibt das erwartete Resultat aus.
  - **Wrong Answer:** Das Programm gibt nicht das erwartete Resultat aus.
  - **Wrong Exitcode:** Falscher Rückgabewert.
  - **Program Error:** Das Programm kann nicht gestartet werden (typischerweise konnte das Programm nicht gefunden werden, da nicht erfolgreich kompiliert).
  - **Timeout:** Zeitüberschreitung bei der Ausführung (*).
  - **Abort:** Abbruch durch den Benutzer.
- **Time:** Die Zeit, welche Ihr Programm benötigte.
- **Timout:** Das Zeitlimit.

(*) Es gibt kein fixes Zeitlimit für Ihren Compiler. Die Überprüfung auf Zeitüberschreibung dient lediglich dazu Endlosschleifen abzufangen. Sollte Ihr Compiler mehr Zeit benötigen, steht Ihnen frei das Zeitlimit zu erhöhen.

Tipp: Sollte die Ausführung der Tests sehr viel Zeit in Anspruch nehmen, können Sie anstelle eines vollständigen Builds mit Tests (`build`) auch einen Build ohne Tests (`assemble`) durchführen.

Regeln zu den Systemtests

- Die Tests sind eine Hilfestellung zur Erfüllung der Aufgabe und geben das Punkteraster vor. **Wir schauen uns Ihre Abgabe trotzdem an!**
- Festkodierte Ausgaben wie im folgenden Beispiel geben keine Punkte:

```
if (input.equals("abc") {
    System.out.println("edf");
} else if (...) { ...
```

  ⇨ **Wir testen noch mit einem zweiten Testset mit leicht anderen Werten!**
- Sollte das Framework trotz offensichtlichem Fehler (z.B. entsprechende Funktionalität gar nicht implementiert, kein Output wird angezeigt, Exception wird angezeigt, etc.) grün anzeigen: Dafür gibt es ebenfalls keine Punkte. **Bitte melden Sie solche Fälle umgehend per Email!**

6. Abgabe eines Meilensteins

Zur Abgabe eines Meilensteins gehen Sie wie folgt vor:

1. Erstellen Sie eine ZIP-Datei mit dem aktuellen Stand, d.h., der Verzeichnisse `src` und `reports` innerhalb des Verzeichnisses `CompilerFw`. Sie können dazu den Gradle-Task `makeSubmission` aufrufen. Dieser Task erzeugt eine Datei mit dem Namen `COBAU-MSx-Gyy.zip`, welche Sie bitte entsprechend umbenennen: `x` entspricht dem abzugebenden Meilenstein und `yy` Ihrer Gruppennummer. **Die Angabe von Gruppe und Meilenstein im Dateinamen ist wichtig und dient dazu Verwechslungen vorzubeugen.**

   Beispiel: Würde Gruppe 42, also den Meilenstein 3 einreichen, wäre der Name der Datei: `COBAU-MS3-G42.zip`.

2. Reichen Sie die ZIP-Datei im ILIAS unter dem entsprechenden Meilenstein ein (Compilerprojekt -> Abgabeordner -> Meilenstein x).

   Wichtig: Bei mehreren Uploads wird der letzte Upload (gemäss ILIAS) vor Ablauf des Abgabezeitpunkts bewertet.