Installationsanleitung für die Programmierung in Pascal

mit Debugging in Visual Studio Code

Allgemeines

hier folgt nun eine Anleitung für die Einrichtung der IDE (Visual Studio Code) zum Programmieren mit Pascal. Dieser ist besonders Attraktiv wegen den vielen AddOns und weil es nunmal so gut aussieht und zum Programmieren so fast alles enthält, was man braucht.

WICHTIG: Diese Anleitung ist unter OS X geschrieben worden. Für alle anderen Betriebssysteme müsst Ihr euch dann selbst mit den Tools und Installationen auseinander setzen.

Anforderungen

Für den Anfang sind folgende Werkzeuge nötig:

- Pascal-Compiler (z.B. fpc in https://www.freepascal.org/download.var)
- Visual Studio Code (https://code.visualstudio.com/Download)

Ihr müsst nun auf eurem OS entsprechend einen FPC-Compiler installieren. FPC steht hierbei für FreePascal. Das Ergebnis ist, dass ihr später im Terminal den FPC-Befehl wie folgt erreichen könnt:

```
datas-MacBook-Pro:pascal dataclub$ fpc
Free Pascal Compiler version 3.0.2 [2017/02/12] for i386
Copyright (c) 1993-2017 by Florian Klaempfl and others
fpc [options] <inputfile> [options]
```

Den Installationspfad findet man mit dem Befehl:

```
datas-MacBook-Pro:pascal dataclub$ which fpc
/usr/local/bin/fpc
```

Damit müsst ihr nun einen Pascal-Quellcode mit dem untenstehenden Befehl kompilieren können:

```
datas-MacBook-Pro:pascal dataclub$ fpc -g ab.pas
Free Pascal Compiler version 3.0.2 [2017/02/12] for i386
Copyright (c) 1993-2017 by Florian Klaempfl and others
Target OS: Darwin for i386
Compiling ab.pas
ab.pas(19,3) Note: Local variable "Feld1" not used
ab.pas(20,3) Note: Local variable "Feld2" not used
ab.pas(21,3) Note: Local variable "ErgebnisFeld" not used
ab.pas(22,3) Note: Local variable "i" not used
ab.pas(23,3) Note: Local variable "j" is assigned but never used
ab.pas(24,3) Note: Local variable "k" not used
Assembling (pipe) ab.s
Linking ab
33 lines compiled, 0.4 sec
6 note(s) issued
```

Code-Formatting & Code-Highlighting

Bei der Programmierung gibt es Formatierungsregeln zu beachten. Diese braucht man jedoch nicht per Hand einrücken, sondern kann alles von der IDE übernehmen lassen. Wie gut, dass unsere IDE (VSC) eine entsprechende Extension enthält, welche man einfach über den Punkt “Erweiterungen” installieren. Dieser nennt sich “Language Pascal”. Sucht einfach danach und installiert diesen über VSC.
(https://marketplace.visualstudio.com/items?itemName=alefragnani.pascal)

Nun noch entsprechend die settings.json in .vscode-Ordner des Projektverzeichnisses erweitern mit:
settings.json

```
    "pascal.formatter.engine": "ptop",
    "pascal.formatter.enginePath": "/usr/local/bin/ptop",
```

Nach dem Neustart vom VSC müsste es dann möglich sein, den Quellcode zu formatieren, wenn man diesen Selektiert und über die rechte Maustaste auf “Format Document” drückt.

Code-Running

Es gibt eine Möglichkeit den Code automatisch über die IDE laufen zu lassen. Das ist interessant, wenn man nicht ständig zwischen der IDE und dem Terminal wechseln möchte und die ganzen Aufgaben (zum Ordner wechseln, Kompilieren & Ausführen) von der IDE übernehmen lassen möchte.

Dafür gibt es den Code-Runner, den man auch einfach über eine Extension installiert und über die Conf einstellen kann. Dieser lautet: “Code Runner” und ist über die URL erreichbar: (https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner)

Danach noch die settings.json einstellen:
settings.json

```
    "code-runner.executorMap": {
        "build": "cd $dir && fpc -g $fileName",
        "run": "cd $dir && fpc -g $fileName && ./$fileNameWithoutExt"


    },
    "code-runner.clearPreviousOutput": true,
    "code-runner.saveFileBeforeRun": true,
    "code-runner.runInTerminal": true,
```

Im settings.json-Datei habe ich noch zusätzlich paar weitere Sachen gemacht, die die Arbeit erleichtern.

Um den Runner auszuführen, drückt man auf F1-Taste, gibt in der oberen Leiste in das Eingabefeld “Run By Language” ein und dann “build” oder “run” ein.

Debugging

Jetzt kommen wir zum interessanten Teil der Geschichte. Debugging ist nötig, da man sonst “blind” programmiert. Naturlich geht es auch ohne, nur wer drauf Lust hat, kann auch gerne ohne Debugger arbeiten, jedoch ist es ein MUSS.

Kurze Erklärung was ein Debugger ist:
Man kann einen Quellcode debuggen lassen, in dem man über die breakpoints den Code einzeln durchlaufen lassen kann, um zu sehen, was im Code tatsächlich geschieht. So kann man Fehler schneller beheben und deren Ursache leichter rausfinden.

Um Debuggen zu können braucht ihr dafür:

- einen Debugger (z.B. gdb oder lldb-mi)
- einen Adapter für Visual Studio Code, der sich zwischen Debugger und Code anhängt

GDB Installation

Jetzt wird es besonders frickelig. Um mir die Schreiberei zu ersparen habe ich unten ein Youtube-Video angehängt wie man den GDB-Debugger installiert.

https://www.youtube.com/watch?v=QE_IAjkXEIg

Wenn alles erfolgreich war, müsstet Ihr gdb über Terminal erreichen können:

```
datas-MacBook-Pro:pascal dataclub$ which gdb
/usr/local/bin/gdb
```

WICHTIG: Ich hoffe, dass ihr auch ein GDB-Zertifikat erstellt und richtig erlaubt habt. Ggf. noch das System neustarten, um sicher zu sein.

Quellcode im VSC-IDE mit GDB debuggen

Dazu benötigt Ihr die Extension “Native Debug”. Diesen Installiert Ihr einfach über die Extensions. (https://marketplace.visualstudio.com/items?itemName=webfreak.debug)

Nun kommt die Konfiguration. Dazu passt Ihr launch.json-Datei unter .vscode-Ordner im Projektverzeichnis an.

launch.json

```
{
    "version": "0.2.0",
    "configurations": [

        {
            "type": "lldb-mi",
            "request": "launch",
            "name": "LLDB-Debug",
            "target": "${fileDirname}/${fileBasenameNoExtension}",
            "cwd": "${workspaceRoot}",
            "lldbmipath": "/usr/local/bin/lldb-mi"
        },
        {
            "type": "gdb",
            "request": "launch",
            "name": "GDB-Debug",
            "target": "${fileDirname}/${fileBasenameNoExtension}",
            "cwd": "${workspaceRoot}",
            "gdbpath": "/usr/local/bin/gdb",
            "preLaunchTask": "build"

        }
    ],
    "compounds": []
```

```
}
```

Danach kann man die Quellcode-Datei mit F5 debuggen. Davor muss der Quellcode vorher kompiliert werden, erst dann kann nun ein Breakpoint gesetzt und der Code im VSC mit dem Play-Button gedebugt werden!

Task für die Kompilierung und Debugging

Das Ziel ist es nun, nur noch den Play-Button in Visual Studio Code zu drücken, sodass dann automatisch der Code kompiliert und dann hinterher gedebugt wird. Dafür muss man ein Task anlegen. Um die oben genannte Arbeit komplett abnehmen zu lassen, könnt Ihr ein Task wie folgt anlegen. Im .vscode-Order einen tasks.json-Datei anlegen und mit folgendem Inhalt befüllen.

tasks.json

```
{
    "version": "0.1.0",
    "command": "fpc",
    "isShellCommand": true,
    "args": ["-g", "${fileDirname}/${fileBasenameNoExtension}"],
    "showOutput": "always",
    "tasks": [
        {
            "taskName": "build",
            "suppressTaskName": true
        }
    ]
}
```

Viel Spass und falls noch fragen sind, in discord anschreiben.

Noch paar Links für euch, was ich alles insgesamt durchgegangen bin:
https://code.visualstudio.com/docs/editor/debugging
http://lldb.llvm.org/build.html#BuildingLldbOnMacOSX
http://kirste.userpage.fu-berlin.de/chemnet/use/info/gdb/gdb_6.html

https://marketplace.visualstudio.com/items?itemName=webfreak.debug
https://marketplace.visualstudio.com/items?itemName=formulahendry.code-runner