Thema, Ziele: Libraries erstellen und verwenden, C++ Auffrischung

Für die folgenden Aufgaben lohnt sich eine saubere Ordner-Struktur. Das jeweilige Library Projekt sollte sich in einem anderen Ordner befinden als das Executable Projekt.

Aufgabe 1: Static Library erstellen

Verpacken Sie die Klasse Stack vom Ordner Vorlage/Stack in eine static Library mit dem Namen libStack.a.

Aufgabe 2: Static Library nutzen

Verwenden Sie die von Aufgabe 1 erstellte Library *libStack.a*. Erstellen Sie ein *main.cpp* File in welchem Sie auf die *Stack* Library zugreifen und zugleich testen.

Welche Dateien brauchen Sie dazu?

Achten Sie beim Erstellen des *main.cpp* Files darauf, dass dieses in einem anderen Ordner als die *Stack* Library liegt.

Aufgabe 3: Shared Library erstellen

Verpacken Sie die Klasse Stack analog der Aufgabe 1 in eine shared Library mit dem Namen libStack.so.

Aufgabe 4: Shared Library nutzen

Verwenden Sie die in Aufgabe 3 erstellte shared Library analog Aufgabe 2, diesmal jedoch mit der in Aufgabe 3 erstelle shared Library *libStack.so*.

Mit dem Linux Befehl `ldd` können die verwendeten shared Libraries in einem ausführbaren Programm angezeigt werden.

Welche shared Libraries werden im soeben erstellten Programm verwendet?

Aufgabe 5: Static Library mit Eclipse erstellen

Erstellen Sie aus der Stack Klasse im Vorlageordner eine statische Library *libStack.a*. Dazu soll ein neues C++Projekt in Eclipse erstellt werden. In Abbildung 1 sind die benötigten Einstellung beim Erstellen eines neuen C++ Projekts ersichtlich. Die Ordnerstruktur wird wie in Abbildung 4 dargestellt empfohlen.

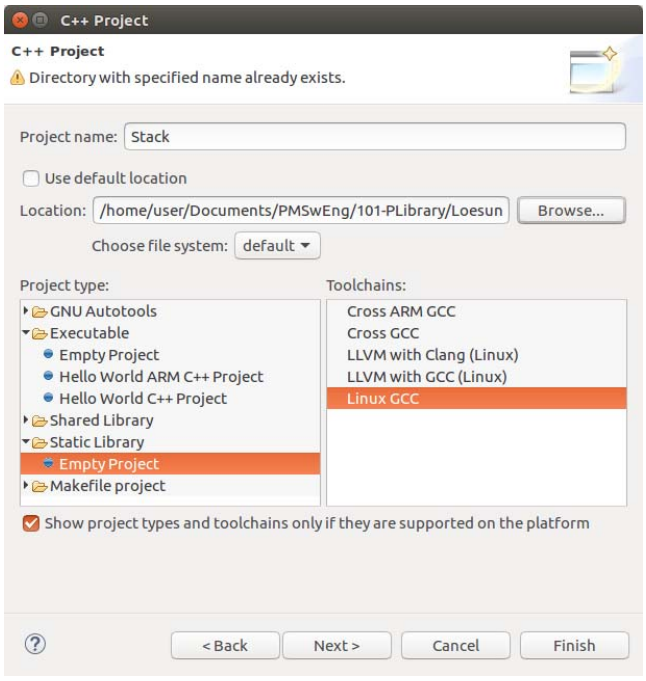

Abbildung 1 C++ Einstellungen für ein neues static Library Projekt

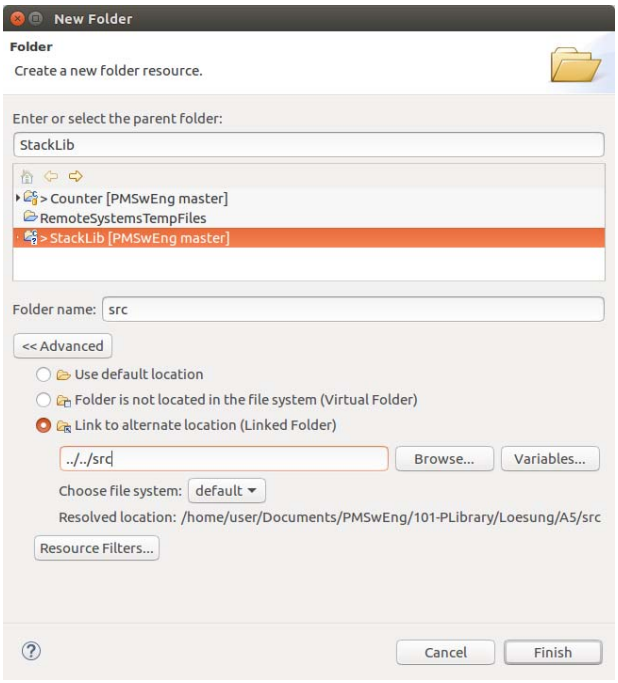

Abbildung 2 Verklinken von *src* Ordner

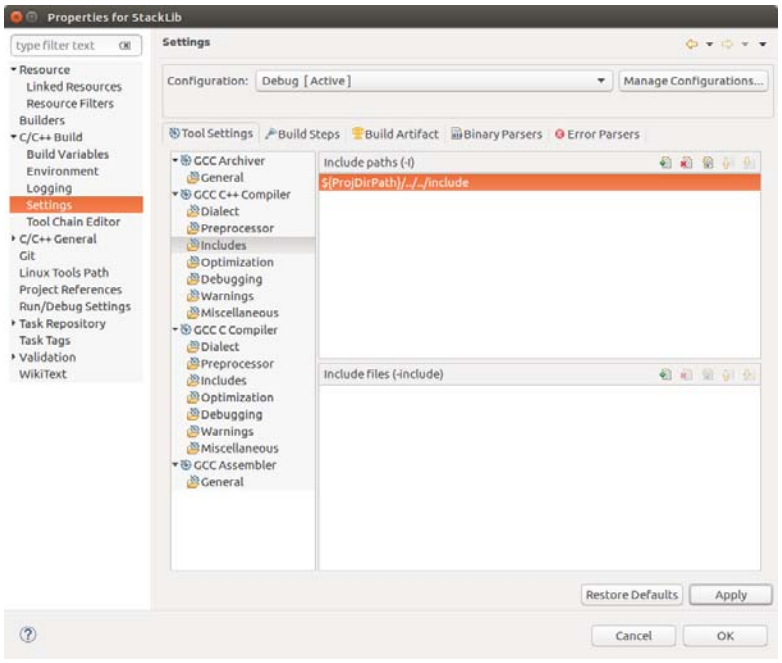

Abbildung 3 Projekteinstellung für *Include* Pfad

```
A5/
├── include
│   └── Stack.h
├── lib
│   └── libStack.a
├── prj
│   └── Eclipse
│       ├── .cproject
│       ├── Debug
│       │   ├── makefile
│       │   ├── objects.mk
│       │   ├── sources.mk
│       │   └── src
│       │       └── subdir.mk
│       ├── .project
│       └── .settings
│           └──
└── src
    └── Stack.cpp
```

Abbildung 4 Ordner Struktur

Das neu zu erstellende Projekt sollte also im *prj/Eclipse* Pfad abgespeichert werden.

Da Eclipse nur die Dateien im gleichen Ordner oder darunter im Projekt darstellt, müssen die *src* und *include* Ordner verlinkt werden. Wählen Sie dazu *File* → *New* → *Folder* aus. Nun kann der src Ordner wie in Abbildung 2 verlinkt werden. Der gleiche Schritt kann für den *include* Ordner wiederholt werden.

Wie in den vorhergehenden Aufgaben muss auch hier der *include* Ordner angegeben werden, damit der Compiler die Header Datei findet. Der Pfad kann in den Projekteinstellungen angegeben werden (siehe Abbildung 3).

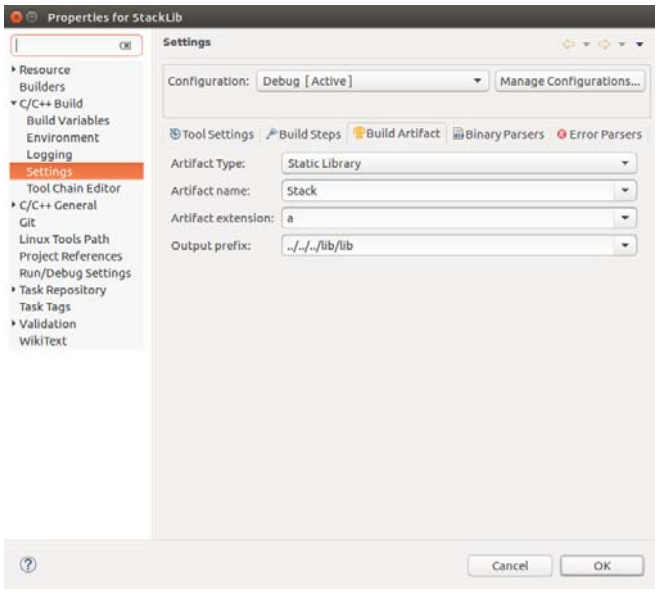

Abbildung 5 Definition des Output Ordners

Eclipse erstellt in einem *managed make* Projekt (typisches Projekt) die generierten *object* und *exectuable* Files unter dem Eclipse Projektdateien Ordner ab. Falls das generierte Library-File in den *lib* Ordner abgelegt werden soll wie bei den Aufgaben 1 und 3, kann dies in den Projekteinstellungen gemäss Abbildung 5 einstellen.

Aufgabe 6: Static Library mit Eclipse verwenden

Erstellen Sie wie vom ProgCPP Modul gewohnt ein neues Eclipse Projekt. Beachten Sie die Ordner-Struktur ähnlich der Struktur von Aufgabe 5 (siehe Abbildung 7).

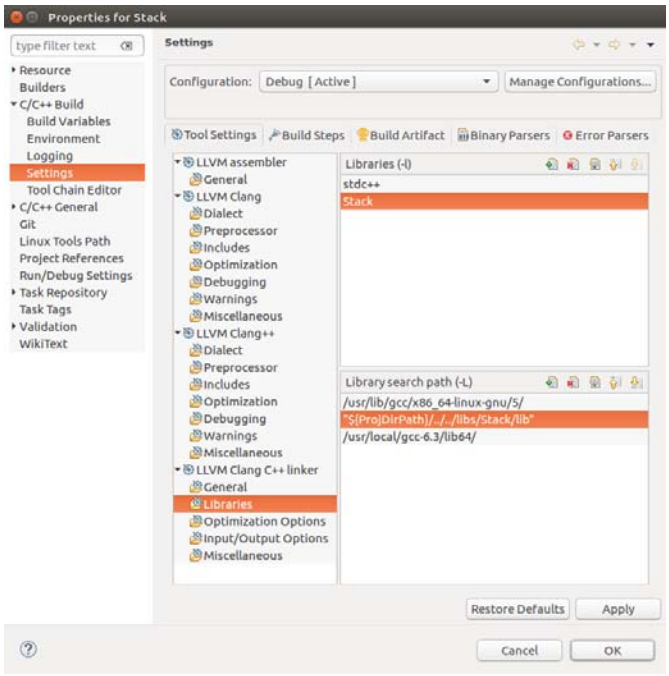

Abbildung 6 Angabe von Library Pfad und Library Name für Linker

```
A6/
├── libs
│   └── Stack
│       ├── include
│       │   └── Stack.h
│       └── lib
│           └── libStack.so
├── prj
│   └── Eclipse
└── src
    └── main.cpp
```

Abbildung 7 Ordnerstruktur Eclipse Executable

Vorgehen:

1. Speichern Sie die Projektdateien im Verzeichnis *prj/Eclipse* ab.
2. Verlinken Sie den src Ordner wie in Abbildung 2 dargestellt.
3. Fügen Sie den Includepfad *lib/include* hinzu (ähnlich Abbildung 3, aber anderer Pfad)
4. Fügen Sie den Library-Pfad hinzu sowie die Stack Library (siehe Abbildung 6)

Aufgabe 7: Klasse Counter erstellen

Erstellen Sie eine Klasse die eine Counter Funktionalität abbildet. Folgende API soll zur Verfügung stehen:

```
Counter();
int getValue() const;
void incValue();
void decValue();
void setValue(int value);
void clearValue();
```

Im Vorlage Ordner Vorlage/Counter befindet sich das Testprogramm *main.cpp* für ihre *Counter* Klasse. Die Klasse muss nicht in eine Library verpackt werden.