

FH-OÖ Hagenberg/HSD
Betriebssysteme 3, WS 2025
Übung 6



Name: _____ Aufwand in h: _____

Mat.Nr: _____ Punkte: _____

Übungsgruppe: _____ korrigiert: _____

Kommando-Interpreter (24 Punkte)

Ein Kommando-Interpreter, auch als Shell oder Befehlsinterpretierer bezeichnet, ist eine Softwareanwendung in einem Betriebssystem, die Benutzern die Interaktion mit dem System durch die Eingabe von Befehlen ermöglicht. Der Kommando-Interpreter interpretiert die eingegebenen Befehle und führt die entsprechenden Aktionen aus, indem er Systemressourcen verwaltet, Programme ausführt oder andere Aufgaben erledigt.

Dieses Hilfsmodul erlaubt die Verwaltung von Funktionen, zugehöriger Hilfetexte, den Aufruf einzelner Funktionen samt Parameterübergabe und deren Exekution und Bewertung von Rückgabewerten.

Um den Implementierungsaufwand in Grenzen zu halten, wird ein vorgefertigtes Code-Template zur Verfügung gestellt. Es besteht aus den Modulen *CommandInterpreter* und *CommandTable*.

a) **Modul: *CommandInterpreter***

Die Modul-Schnittstelle des Kommando-Interpreters ist in der Header-Datei *CommandInterpreter.h* bereits vorgegeben.

Damit das Modul unabhängig von der verfügbaren Zeichenausgabe ist, erfolgt diese ausschließlich über eine konfigurierbare Funktion, die jeweils nur ein Zeichen übernimmt:

```
int PutChar(int ch);
```

Die `Print(...)`-Funktion in der Schnittstelle verwendet genau diese Funktion zur Ausgabe. Alle anderen Ausgabefunktionen dürfen nicht verwendet werden.

Initialisierung:

Das Modul bietet eine Initialisierungsfunktion: Parameter sind die Liste der verfügbaren Kommandos `TCmdHndTable` (inkl. Zusatzinformationen) und der Funktionszeiger der Ausgabefunkti-

on (z.B. `int putchar(int c)` aus `<stdio.h>`). Diese Daten werden in modulinternen Variablen verwaltet. Ein Begrüßungstext (z.B. "Welcome Command-Interpreter") und die Ausgabe des Promptzeichens (z.B. ">") signalisieren die Bereitschaft zur zeilenweisen Eingabe von Kommandos.

Kommandoeingabe:

Die Modulbenutzer rufen mit jedem Eintreffen eines Zeichens (`int getchar(void)`) eine Funktion `Process` auf, der als Parameter das Zeichen übergeben wird. Der Rückgabewert dieser Funktion gibt Aufschluss, ob die "Escape" Taste gedrückt wurde und somit die Kommandoeingabe beendet werden soll.

Die einzelnen Zeichen werden in einen modulinternen Puffer kopiert und zusätzlich als Echo über die Ausgabefunktion ausgegeben. Sobald das Zeichen LF ("\") oder CR ("\") detektiert worden ist, erfolgt die Analyse, ob im eingegebenen Text ein Kommando aus der definierten Liste enthalten ist. Je Eingabezeile ist ein Kommando erlaubt, das entweder direkt durch LF | CR oder durch ein Leerzeichen begrenzt ist. Wurde das Kommando in der Liste gefunden, erfolgt der Aufruf der zugehörigen Kommandofunktion und die Anzeige dessen Rückgabewertes. Wurde ein ungültiges Kommando eingegeben (nicht in der Liste enthalten), wird dies entsprechend angezeigt. Als Abschluss erfolgt wiederum die Ausgabe des Promptzeichens.

b) Modul: *CommandTable*

Die Schnittstelle ist bereits vorgegeben unb besteht aus dem Typ der Kommando-Tabelle `TCmdHndTable` und einer Zugriffsfunktion `GetCmdHndTable()`, die einen Zeiger auf die Tabelle liefert.

Das Modul beinhaltet intern die vordefinierte Kommando-Tabelle mit allen Kommandos, wobei die internen Kommando-Funktionen bereits vordeklariert sind, die vollständige Implementierung fehlt und ist fertigzustellen.

Eine Kommandofunktion ist folgendermaßen definiert:

```
int Command(void);
```

Jedes Kommando liefert entsprechend seines Bearbeitungsergebnisses einen Rückgabewert: 0... fehlerhaft, sonst OK.

Kommandooverarbeitung:

Ein Kommando kann getrennt durch ein (oder mehrere) Leerzeichen von einem oder mehreren jeweils wieder durch Leerzeichen getrennte Parametern gefolgt werden. Hierfür bietet das Modul Funktionen in der `TCmdHndTable`

```
1 int CmdParamInt() { ... }
2 int CmdParamFloat() { ... }
3 int CmdParamString() { ... }
4 int CmdParamAddInt() { ... }
```

zur fortlaufenden Bewertung der Parameter als Integer (int), Fließkomma (float) oder Zeichen-

kette (string) an. Die Anzahl der zu lesenden Parameter ergibt sich aus dem Bedürfnis jedes einzelnen Kommandos, d.h. überzählige Parameter werden ignoriert.

Die einzelnen Kommando-Funktionen sind ebenfalls entsprechend der `TCmdHndTable` in diesem Modul implementiert und haben folgende Eigenschaften:

"help", "?" geben jeweils einen Hilfetext entsprechend obigem Vorbild aus.
"int" ein Parameter wird als Integer-Wert interpretiert und angezeigt.
"float" ein Parameter wird als Fließkomma-Wert interpretiert und angezeigt.
"string" ein Parameter wird als Zeichenkette interpretiert und angezeigt.
"addint" zwei Integer-Werte werden addiert und angezeigt.

c) Tests

Ausgabe: Ausgabe nach Aufruf des Kommandos **help**:

```
CmdHnd: Welcome!
>help
help
(press <ESC> to exit)
available commands:
help show help text
? show help text
int [value] first parameter as integer
float [value] first parameter as float
string [value] first parameter as string
addint [value1] [value2] adds two integer values
OK
>
```

Ausgabe nach Aufruf des Kommandos **int** mit Parameter 123:

```
>int 123
int 123
Parameter: 123
OK
>
```

Ausgabe nach Aufruf des Kommandos **float** mit Parameter 3.14:

```
>float 3.14
float 3.14
Parameter: 3.14
OK
>
```

Ausgaben nach Aufruf des Kommandos **string** mit Parameter "Donald":

```
>string Donald
string Donald
Parameter: Donald
OK
>
```

Ausgaben nach Aufruf des Kommandos **float** mit Parameter "Donald":

```
>float Donald
float Donald
ERROR
>
```

Ausgaben nach Aufruf des Kommandos **addint** mit den Parametern 3 und 4:

```
>addint 3 4
addint 3 4
Result: 3 + 4 = 7
OK
>
```

Versehen Sie Ihren Quelltext mit entsprechenden Kommentaren.

Allgemeine Hinweise: Legen Sie bei der Erstellung Ihrer Übung großen Wert auf eine **saubere Strukturierung** und auf eine **sorgfältige Ausarbeitung!** Dokumentieren Sie alle Schnittstellen und versehen Sie Ihre Algorithmen an entscheidenden Stellen ausführlich mit Kommentaren! Testen Sie ihre Implementierungen ausführlich! Geben Sie den **Testoutput** mit ab!