

## Programmierung(2)

# Agenda

- **Funktions-Prototypen**

- Definition
- Motivation
- Beispiel

- **Eigene Bibliotheken**

- Motivation
- Beispiel

- **Fachpraktische Anwendungen**

# Funktions-Prototyp – Definition

- Ein Funktions-Prototyp erinnert an eine Variable, die zwar bereits deklariert und definiert worden ist, der aber bisher noch kein Wert zugewiesen wurde.
- Ähnlich verhält es sich nun auch mit einem Funktions-Prototypen, bei dem zwar bereits ...
  - der **Name** der Funktion
  - die **Typen** aller Parameter
  - der **Typ** des Rückgabewertes... festgelegt wurden, bei dem aber noch **keine Definition des Funktionsrumpfes** stattgefunden hat.

# Funktions-Prototyp – Motivation

- Funktions-Prototypen können eingesetzt werden, wenn mehrere Programmierer an einem gemeinsamen **Projekt** arbeiten. Auf diese Weise kann dann bereits zu Beginn eine Team-übergreifende Einigung bzgl. Aufbau und Namensgebung von Funktionen erzielt werden.
- Funktions-Prototypen können aber auch der **Geheimhaltung** dienen, falls der Code des Funktionsrumpfes nicht veröffentlicht werden soll.
- Ferner können Funktions-Prototypen auch technisch notwendig sein, falls sich zwei (oder mehrere) Funktionen **gegenseitig aufrufen**. (Dieser Fall wird als „**Mutual Recursion**“ bezeichnet und wird Gegenstand unseres Beispiels sein.)
- Entsprechend bietet sich der Einsatz von Prototypen in all jenen Fällen an, in denen eine solche **gegenseitige Abhängigkeit** von Funktionen nicht ausgeschlossen werden kann, oder aber zumindest durch zukünftige Änderungen am Quellcode eintreten könnte.

## Hinweis

Der von uns verwendete Compiler (**gcc**) ist „schlauer“ als verlangt.  
Als sogenannter „**Multi-Pass-Compiler**“ (*Kompilierung geschieht mittels **mehrfachem** Durchlaufen des Quellcodes*) erlaubt er uns, **Mutual Recursion** auch ohne Verwendung von Prototypen zu realisieren.

Der Allgemeingültigkeit halber wollen wir uns darauf aber nicht verlassen.  
Stattdessen soll unser Quellcode auch für einen „**Single-Pass-Compiler**“ verständlich sein.  
Daher werden wir vor der Definition der Funktionsrumpfe die entsprechenden Prototypen einführen.

# Funktions-Prototyp – Beispielaufgabe

- Das Hauptprogramm besteht lediglich aus dem Aufruf der Funktion „tueWas“. Da tueWas allerdings eine andere Funktion aufruft, die ihrerseits wieder tueWas aufrufen wird, entsteht eine Rekursion, die (in diesem Fall) eine Zählschleife ersetzt. Genauer gilt:

Funktionsname: ..... **tueWas**

Übergabewerte: ..... 2 Integer zaehler und max

Funktionalität: ..... Gibt den aktuellen Zählerwert auf der Konsole aus und ruft anschließend die Funktion „erhoeheZaehler“ auf (mit den identischen Übergabewerten)

Rückgabewert: ..... keiner

Funktionsname: ..... **erhoeheZaehler**

Übergabewerte: ..... 2 Integer zaehler und max

Funktionalität: ..... Erhöht den Zählerwert um 1  
falls neuer Zählerwert  $\leq$  max, so wird die Funktion „tueWas“ aufgerufen  
(mit dem neuen Wert von zaehler und dem unveränderten von max)

Rückgabewert: ..... keiner

# Funktions-Prototyp – Beispielaufgabe

```
#include<stdio.h>

void erhoeheZaehler(int, int);
void tueWas(int, int);

void erhoeheZaehler(int zaehler, int max)
{
    zaehler++;
    if(zaehler<=max)
tueWas(zaehler,max);
}

void tueWas(int zaehler, int max)
{
    printf("Aktueller Wert: %d\n",zaehler);
    erhoeheZaehler(zaehler,max);
}

int main(void)
{
    tueWas(1,10);
    return 0;
}
```

# Funktions-Prototyp – Beispielaufgabe

```
#include<stdio.h>

void erhoeheZaehler(int, int);
void tueWas(int, int);

void erhoeheZaehler(int zaehler, int max)
{
    zaehler++;
    if(zaehler<=max)
        tueWas(zaehler,max);
}

void tueWas(int zaehler, int max)
{
    printf("Aktueller Wert: %d\n",zaehler);
    erhoeheZaehler(zaehler,max);
}

int main(void)
{
    tueWas(1,10);
    return 0;
}
```

Die **Syntax** eines Prototypen entspricht im Wesentlichen dem Kopf der zugehörigen Funktion.

Es fehlen allerdings die geschwungenen Klammern. Stattdessen wird die Notation eines Funktions-Prototypen mit einem **Semikolon** abgeschlossen.

Ferner müssen den **Parametern keine Namen** zugewiesen werden. (Dies ist zwar nicht explizit verboten, wäre aber bedeutungslos).

# Funktions-Prototyp – Beispielaufgabe

```
#include<stdio.h>

void erhoeheZaehler(int, int);
void tueWas(int, int);

void erhoeheZaehler(int zaehler, int max)
{
    zaehler++;
    if(zaehler<=max) tueWas(zaehler,max);
}

void tueWas(int zaehler, int max)
{
    printf("Aktueller Wert: %d\n",zaehler);
    erhoeheZaehler(zaehler,max);
}

int main(void)
{
    tueWas(1,10);
    return 0;
}
```

Die erste Funktion **erhöht zunächst den Zähler** und ruft dann die Funktion „**tueWas**“ auf, falls der aktuelle Zählerwert kleiner oder gleich **max** ist.



# Funktions-Prototyp – Beispielaufgabe

```
#include<stdio.h>

void erhoeheZaehler(int, int);
void tueWas(int, int);

void erhoeheZaehler(int zaehler, int max)
{
    zaehler++;
    if(zaehler<=max) tueWas(zaehler,max);
}

void tueWas(int zaehler, int max)
{
    printf("Aktueller Wert: %d\n",zaehler);
    erhoeheZaehler(zaehler,max);
}

int main(void)
{
    tueWas(1,10);
    return 0;
}
```

Die zweite Funktion gibt den **aktuellen Zählerwert** auf der Konsole aus. Anschließend wird die Funktion „erhoeheZaehler“ aufgerufen.

# Funktions-Prototyp – Beispielaufgabe

```
#include<stdio.h>

void erhoeheZaehler(int, int);
void tueWas(int, int);

void erhoeheZaehler(int zaehler, int max)
{
    zaehler++;
    if(zaehler<=max)
        tueWas(zaehler,max);
}

void tueWas(int zaehler, int max)
{
    printf("Aktueller Wert: %d\n",zaehler);
    erhoeheZaehler(zaehler,max);
}

int main(void)
{
    tueWas(1,10);
    return 0;
}
```

Der Aufruf von tueWas(1,10) führt zum Start einer „**Mutual Recursion**“ mit 10 Durchläufen.

**Es erscheint die folgende Ausgabe:**

```
Aktueller Wert: 1
Aktueller Wert: 2
Aktueller Wert: 3
Aktueller Wert: 4
Aktueller Wert: 5
Aktueller Wert: 6
Aktueller Wert: 7
Aktueller Wert: 8
Aktueller Wert: 9
Aktueller Wert: 10
```

# Eigene Bibliotheken – Motivation

- Wir haben bereits eigene Funktionen erstellt, die wir im besten Falle auch bei zukünftigen Programmen **wiederverwenden** können.
- Mit zunehmender Anzahl solcher Funktionen wäre es allerdings **nicht praktikabel**, diese stets oberhalb des main (und unterhalb der Präprozessor-Anweisungen) einzufügen.
- Günstiger wäre es, wenn wir diese Funktionen in einer **externen Datei** auslagern und bei Bedarf mit Hilfe der include-Anweisung ins Programm einbinden könnten.
- Genau dies wird uns mit **eigenen Bibliotheken** („Libraries“, „Header-Dateien“) gelingen.
- Solche Bibliotheken könnten ausschließlich aus **Funktions-Prototypen** bestehen, während die Definition jeder einzelnen Funktion jeweils in einer eigenen Datei vorgenommen wird.
- Wir werden allerdings darauf verzichten, und eine zunehmend beliebter werdende Methode vorstellen, die als „**Header-only**“ bezeichnet wird. Bei dieser besteht die Bibliothek bereits aus den vollständig definierten Funktionen.
- Sofern diese Datei mittels include im main-Quellcode eingebunden ist, wird sie automatisch mit kompiliert. (Voraussetzung ist allerdings, dass Programm und Bibliothek im **selben Ordner** liegen).

# Eigene Bibliotheken – Beispielaufgabe

- Das Programm soll der Reihe nach die drei folgenden Funktionen aufrufen, die alle in der Bibliothek „meineBib.h“ ausgelagert sind.

Funktionsname: ..... **deutscherZeichensatz**

Übergabewerte: ..... keine

Funktionalität: ..... Setzt den deutschen Zeichensatz und löscht den Bildschirm

Rückgabewert: ..... keiner

Funktionsname: ..... **schreibeText**

Übergabewerte: ..... 1 String

Funktionalität: ..... Gibt den String auf der Konsole aus

Rückgabewert: ..... keiner

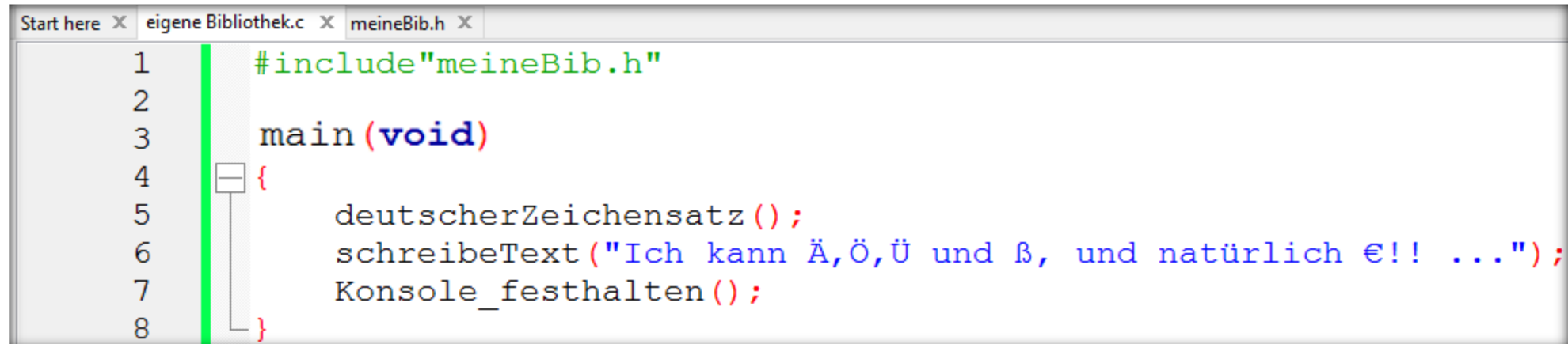
Funktionsname: ..... **Konsole\_festhalten**

Übergabewerte: ..... keine

Funktionalität: ..... Setzt 4 Umbrüche und hält die Konsole bis zu einem Tastendruck fest

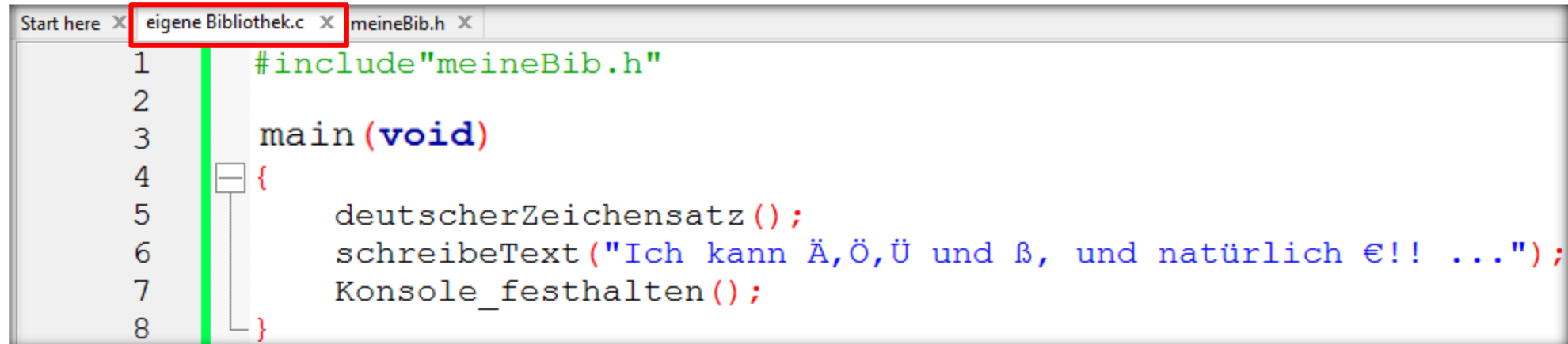
Rückgabewert: ..... keiner

# Eigene Bibliotheken – Beispielaufgabe



```
Start here x eigene Bibliothek.c x meineBib.h x
1  #include "meineBib.h"
2
3  main(void)
4  {
5      deutscherZeichensatz();
6      schreibeText("Ich kann Ä, Ö, Ü und ß, und natürlich €!! ...");
7      Konsole_festhalten();
8  }
```

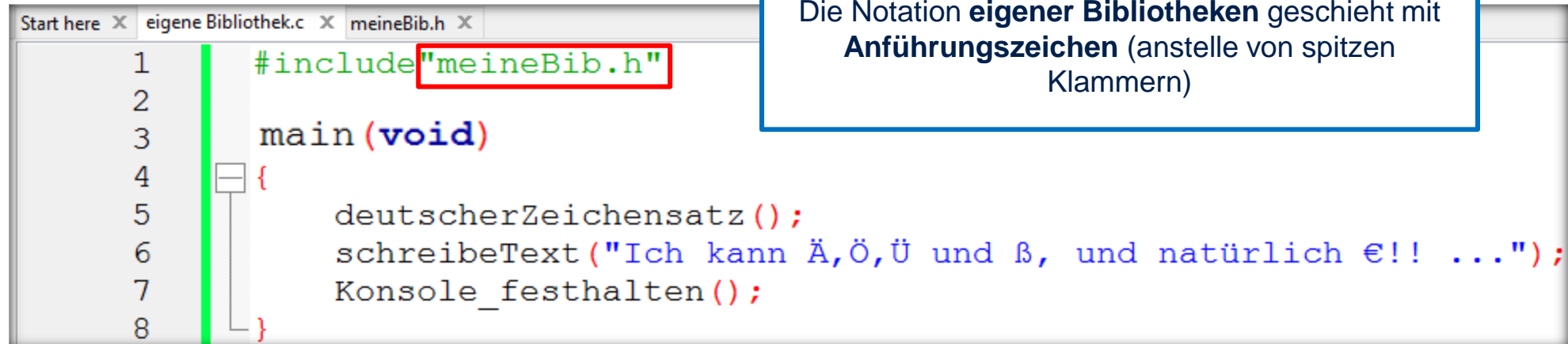
# Eigene Bibliotheken – Beispielaufgabe



The screenshot shows a code editor with two tabs: 'eigene Bibliothek.c' (highlighted with a red box) and 'meineBib.h'. The code in 'eigene Bibliothek.c' is as follows:

```
1  #include "meineBib.h"
2
3  main(void)
4  {
5      deutscherZeichensatz();
6      schreibeText("Ich kann Ä, Ö, Ü und ß, und natürlich €!! ...");
7      Konsole_festhalten();
8  }
```

# Eigene Bibliotheken – Beispielaufgabe



```
Start here X eigene Bibliothek.c X meineBib.h X
1  #include "meineBib.h"
2
3  main(void)
4  {
5      deutscherZeichensatz();
6      schreibeText("Ich kann Ä,Ö,Ü und ß, und natürlich €!! ...");
7      Konsole_festhalten();
8  }
```

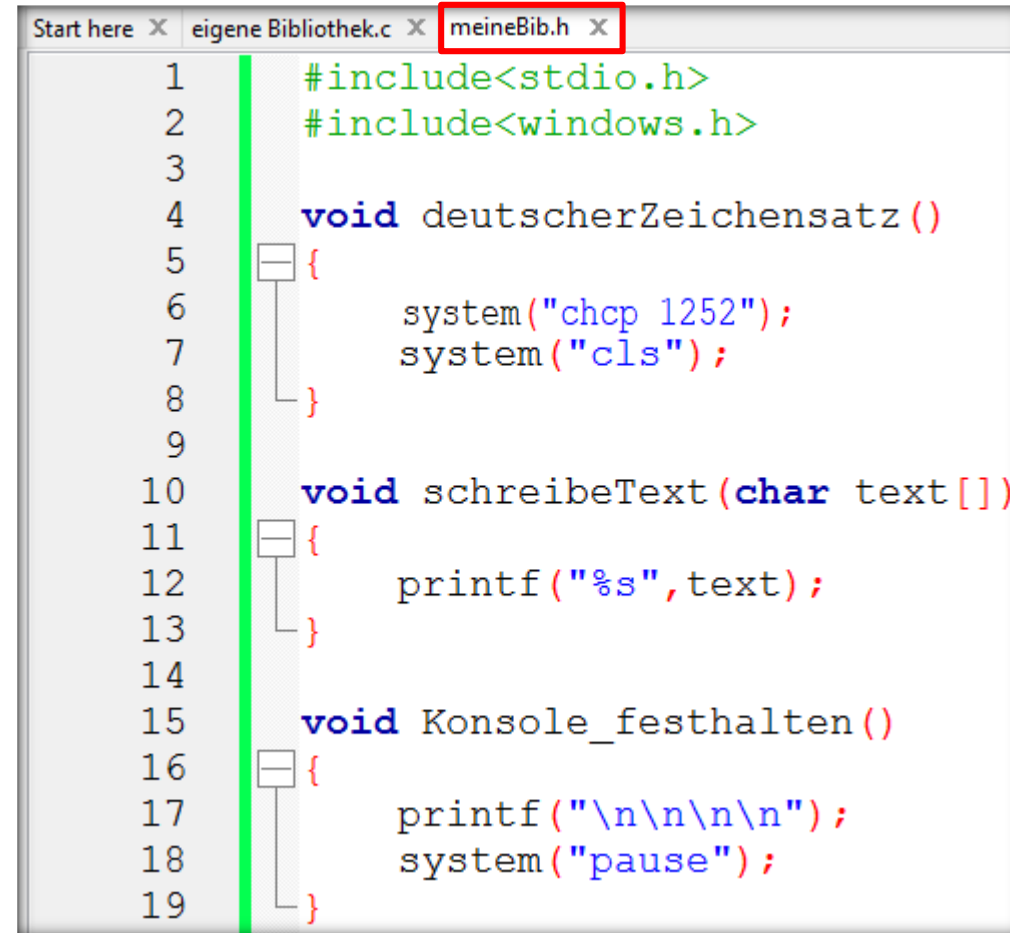
Die Notation **eigener Bibliotheken** geschieht mit **Anführungszeichen** (anstelle von spitzen Klammern)

# Eigene Bibliotheken – Beispielaufgabe

```
Start here x eigene Bibliothek.c x meineBib.h x
1  #include<stdio.h>
2  #include<windows.h>
3
4  void deutscherZeichensatz()
5  {
6      system("chcp 1252");
7      system("cls");
8  }
9
10 void schreibeText(char text[])
11 {
12     printf("%s", text);
13 }
14
15 void Konsole_festhalten()
16 {
17     printf("\n\n\n\n");
18     system("pause");
19 }
```



# Eigene Bibliotheken – Beispielaufgabe



```
1  #include<stdio.h>
2  #include<windows.h>
3
4  void deutscherZeichensatz()
5  {
6      system("chcp 1252");
7      system("cls");
8  }
9
10 void schreibeText(char text[])
11 {
12     printf("%s", text);
13 }
14
15 void Konsole_festhalten()
16 {
17     printf("\n\n\n\n");
18     system("pause");
19 }
```

Nachdem alle Funktionen in die Bibliothek eingetragen wurden, muss diese Datei **gespeichert** werden.

# Eigene Bibliotheken – Beispielaufgabe

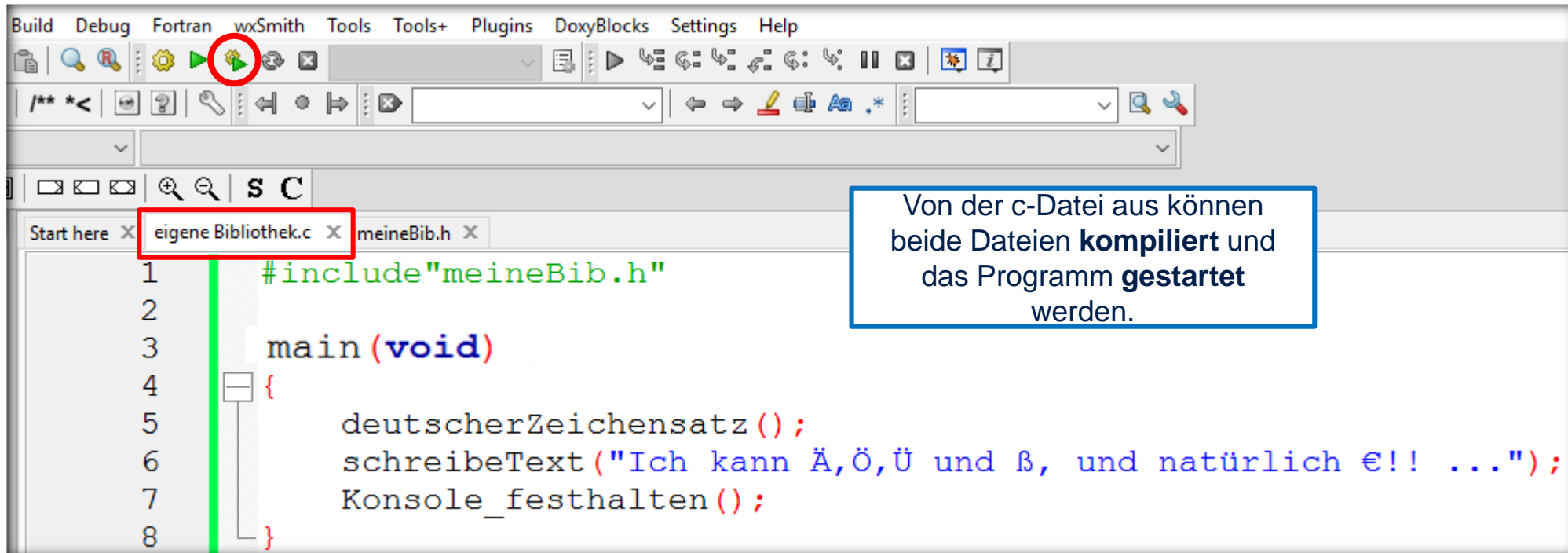
```
Start here x eigene Bibliothek.c x meineBib.h x
1  #include<stdio.h>
2  #include<windows.h>
3
4  void deutscherZeichensatz()
5  {
6      system("chcp 1252");
7      system("cls");
8  }
9
10 void schreibeText(char text[])
11 {
12     printf("%s", text);
13 }
14
15 void Konsole_festhalten()
16 {
17     printf("\n\n\n\n");
18     system("pause");
19 }
```

Auch innerhalb eigener Bibliotheken können bei Bedarf **weitere Bibliotheken** eingebunden werden.

**Hinweis:**

Falls es sich dabei um Bibliotheken handeln sollte, die bereits im Hauptprogramm eingebunden wurden, so ist dies unproblematisch!

# Eigene Bibliotheken – Beispielaufgabe



```
1  #include "meineBib.h"
2
3  main(void)
4  {
5      deutscherZeichensatz();
6      schreibeText("Ich kann Ä,Ö,Ü und ß, und natürlich €!! ...");
7      Konsole_festhalten();
8  }
```

Von der c-Datei aus können beide Dateien **kompiliert** und das Programm **gestartet** werden.

"C:\Users\Administrator\Desktop\C - Programme\test.exe"

```
Ich kann Ä,Ö,Ü und ß und natürlich €!! ...

Drücken Sie eine beliebige Taste . . .
```

# Prototypen, Bibliotheken und Mutual Recursion – Übung A\_06\_02\_01



## Aufgabe\_06\_02\_01

Schreiben Sie bitte Funktionen, mit deren Hilfe Sie mittels **Mutual Recursion** die Ausgabe eines Strings ermöglichen. Lagern Sie bitte diese Funktionen in einer selbst erstellten Headerdatei aus.

### Aufgabenstellung:

Erstellen Sie hierzu bitte einen geeigneten **Quellcode** und die entsprechende **Headerdatei**.

WBS TRAINING AG  
Lorenzweg 5  
D-12099 Berlin  
Amtsgericht Berlin HRB 69531  
Sitz der Gesellschaft: Berlin

Vorstand:  
Heinrich Kronbichler,  
Joachim Giese  
Aufsichtsrat (Vorsitz): Dr. Daniel Stadler  
USt-IdNr.: DE 209 768 248

GLS Gemeinschaftsbank eG  
IBAN: DE18 4306 0967 1146 1814 00  
BIC: GENODEM1GLS



GLS zertifiziert nach  
DIN EN ISO 9001:2015 Reg. Nr. 01114410415  
Zertifizierung nach ISO 14001 Reg. Nr. 01114410415

**VIELEN DANK  
FÜR IHRE  
AUFMERKSAMKEIT!**