FH-OÖ Hagenberg/HSD
Betriebssysteme 3, WS 2025
Übung 2

Name:	**Marco Söllinger**	Aufwand in h:	**4**
Mat.Nr:	**s2410306011**	Punkte:	
Übungsgruppe:	**Gruppe 1**	korrigiert:	

Arbeiten mit dem GCC (24 Punkte)

Machen Sie sich mit den grundlegenden Eigenschaften der Programmiersprache C vertraut. Entpacken Sie die für diese Übung bereit gestellten Dateien (*C-Dateien für Übung2*) in ein eigenes Verzeichnis und analysieren Sie den C-Code. Versuchen Sie die Dateien zu einem ausführbaren Programm ***ctest*** zu übersetzen, indem Sie alle Fehler und Warnungen beseitigen. Kommentieren Sie die beseitigten Fehler, Warnungen und Codeverbesserungen im Quellcode entsprechend!

Benutzen Sie zur Verbesserung des Codes die Fehler-Checkliste auf der Kommunikationsplattform.

Verwenden Sie zur Übersetzung der Dateien den gcc von der Kommandozeile aus und bearbeiten Sie die C-Dateien in einem einfachen Editor (z.B. notepad++.exe)!

Analysieren Sie weiters den Ablauf des Programmes, kommentieren Sie alle Funktionen um den Ablauf entsprechend erklären zu können und machen Sie das Programm lauffähig! Nutzen Sie dazu auch die zur Verfügung gestellten Unterlagen und die Linux-Manpages (*Manual Page - Handbuch-Seite*).

`http://de.wikipedia.org/wiki/Manpage`

Beantworten Sie weiters folgende Fragen:

- Warum sind einige Funktionen als `static` deklariert und andere nicht?
- Welches Programmier-Prinzip lässt sich mit Hilfe von `static`-Funktionen in C realisieren?

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!

Beispiel 1

Es wurde wie in der Aufgabenstellung verlangt das C Programm korregiert, sodass es ohne Warnungen kompiliert.
Fuer das compilieren wurde folgendes Kommando verwendet:
gcc main.c Print.c Test.c -o cTest -Wall -Wextra -Werror

Die Aenderungen wurden in dem Code kommentiert.

1.1 Code

main.c

```c

#include "Print.h" // Add the Header for TestFormatIO
#include "Test.h"  // Add the Header for function declarations

int main() {

  // Formatierte Ein- und Ausgabe: scanf, printf
  PrintResult("Format IO Test", TestFormatIO());

  // Stringoperationen
  PrintResult("String Test", TestString());

  // dynamischer Speicher
  PrintResult("Memory Test", TestDynMem());

  // Deklaration und Verwendung von Strukturen
  PrintResult("Struct Test", TestStruct());

  // Deklaration und Verwendung eines Feldes
  PrintResult("Array Test", TestArray());

  // Zeiger auf Funktionen
  PrintResult("FuncPtr Test", TestFuncPtr());

  return 0;
}
```

Print.h

```c
#ifndef PRINT_H
#define PRINT_H

void PrintResult(char const *const text, unsigned const errorCode);

void PrintHeader(char const header[]);

void PrintStrArr(char const *const *const arr, unsigned const len);

void PrintIntArr(int const *const arr, int len);

#endif
```

Print.c

```c
#include <stdio.h>  // include printf, putc
#include <stdlib.h> // include malloc, free
#include <string.h> // include strlen, strcpy, strcat

static char const *mErrorText[] = {"OK", "NOK"};

// Builds and prints a result string
// but output is always "OK" since the testfunction always returns 0
// an enum would be better here for the return codes
void PrintResult(char const *const text, unsigned const errorCode) {
  // +2 because of space and null terminator
  char *out = (char *)malloc(strlen(text) + strlen(mErrorText[errorCode]) + 2);
  strcpy(out, text);
```

```
  strcat(out, " "); // Appends to out a space
  strcat(out, mErrorText[errorCode]);
  out[strlen(out)] = '\0'; // Null terminator
  printf("%s\n\n", out);
  free(out);
}

void PrintHeader(char const header[]) {
  unsigned headLen = 0;

  printf("\n%s\n", header);
  headLen = strlen(header);
  while (headLen > 0) {
    putc('=', stdout);
    headLen--;
  }
  printf("\n");
}

// Prints an array of strings
void PrintStrArr(char const *const *const arr, unsigned const len) {
  size_t i = 0;
  for (; i < len; i++) {
    printf("%s\n", arr[i]);
  }
}

// Prints an array of integers
void PrintIntArr(int const *const arr, int len) {
  for (int i = 0; i < len; ++i) {
    printf("%d\n", arr[i]);
  }
  return;
}
```

Test.h

```
#ifndef TEST_H
#define TEST_H

// Tests Input methode and prints them back to stdout
// remove extern
int TestFormatIO();

// Copies an string into a char var. the
// prints the length
// Then shifts an text and prints it
int TestString();

int TestDynMem();

int TestStruct();

int TestArray();

int TestFuncPtr();

#endif
```

Test.c

```
#include <stdio.h>  // include printf, fgets, scanf
#include <stdlib.h> // include malloc, free
#include <string.h> // include strlen, strcpy

#include "Print.h" // include PrintHeader, PrintStrArr, PrintIntArr"

#define MAX 100
#define BUFFER_LEN 1024

// Move Prototypes to top of file
static void PrintLength(const char buf[]);
static void Shift(char v[]);

// Tests Input methode and prints them back to stdout
int TestFormatIO() {
```

```c
  // change to const where possible
  const int i = 1;
  const int j = 65;
  const char *pCh = 0;
  const double pi = 3.1415;

  char str[BUFFER_LEN] = "";

  // Remove unused variables
  // int Arr[2][3] = {{1, 2, 3}, {4, 5, 6}};
  // int h = 0x10; Remove unused variable

  PrintHeader("Test Format IO");

  // i++; Directly initialize i with 1
  printf("i: %d ", i);
  // Don't know what was expected here, Now print the address of i
  printf("Address: %p\n", (void *)&i);

  printf("Bitte eingeben: ");
  fgets(str, BUFFER_LEN, stdin);
  printf("%s", str);
  printf("\n");

  // Remove unneeded fgets
  // fgets(str, BUFFER_LEN, stdin);

  int val = 0;
  // Add text before scanf
  printf("Geben sie einen Integer ein:");
  scanf("%4d", &val);
  printf("%d\n", val);

  pCh = (char *)&j;
  // Print the Pointer of the char
  printf("%p \n", (void *)pCh);
  printf("%c \n", *pCh);

  printf("PI: %f \n", pi);

  return 0;
}

// Copies an string into a char var. the
// prints the length
// Then shifts an text and prints it
int TestString() {
  char buffer[MAX] = "";
  char text[] = "ABC";

  PrintHeader("Test Strings");

  strcpy(buffer, "Das ist ein C-String");
  PrintLength(buffer);

  printf("Buffer: %s \n", buffer);

  printf("Text vorher : %s \n", text);
  Shift(text);
  printf("Text nachher: %s \n", text);

  return 0;
}

// Prints a string and its length
static void PrintLength(const char buf[]) {
  // strlen return unsigned long, but %d was used
  printf("Length of \"%s\" is %lu chars\n", buf, strlen(buf));
}

// Implement real shifts
// pushes chars to the right
// pushed out element get added on the left side
static void Shift(char v[]) {
  size_t len = strlen(v);
  if (len < 2)
```

```
92      return; // empty or 1-char strings: nothing to do
93
94    char last = v[len - 1]; // save last character
95    for (size_t i = len - 1; i > 0; --i) {
96      v[i] = v[i - 1]; // shift right
97    }
98    v[0] = last;
99  }
100
101 // creates dynamic memory for a string,
102 // writes something to it and prints it
103 int TestDynMem() {
104   char *Buf = 0;
105
106   PrintHeader("Test dynamic Memory");
107
108   Buf = (char *)malloc(100);
109   if (Buf != 0) {
110     strcpy(Buf, "Hello World!");
111     printf(" -> %s \n", Buf);
112
113     free(Buf);
114   } else {
115     printf("Speicher konnte nicht reserviert werden!");
116     return 1;
117   }
118   return 0;
119 }
120
121 // defines a struct Person,
122 // fills two instances and prints them
123 int TestStruct() {
124   struct Person {
125     char name[100];
126     unsigned weight;
127   } max;
128
129   struct Person moritz;
130
131   PrintHeader("Test Structs");
132
133   strcpy(moritz.name, "Moritz Mustermann");
134   moritz.weight = 80;
135
136   // reset all bytes of max to 0
137   memset(&max, 0, sizeof(struct Person));
138
139   // Add nhull terminator
140   memcpy(max.name, "Max Mustermann", 14 + 1);
141   max.weight = moritz.weight;
142
143   printf("Person: %s hat %d kg\n", max.name, max.weight);
144   printf("Person: %s hat %d kg\n", moritz.name, moritz.weight);
145   return 0;
146 }
147
148 // Tests array initialization with memset
149 // int arr[10] = {0}; also possible
150 // memset with 1 does not work as intended since it fills every byte with 1
151 int TestArray() {
152   int arr[10];
153
154   memset(arr, 0, sizeof(arr)); // Remove & because arr is already a pointer
155   PrintHeader("initialized array with 0:");
156   PrintIntArr(arr, sizeof(arr) / sizeof(int)); // Add len parameter
157
158   // Writes 1 in every byte of the array bacause of that output 16843009
159   memset(arr, 1, sizeof(arr)); // Remove & because arr is already a pointer
160   PrintHeader("initialized array with 1:");
161   PrintIntArr(arr, 10); // Remove & because arr is already a pointer
162
163   return 0; // Add 0, but why an return value at all? on the test functions?
164 }
165
166 // reverses an string
167 static void PrintBackward(const char str[]) {
```

```c
  int i = 0;
  int maxInx = strlen(str) - 1;
  for (i = maxInx; i >= 0; --i) {
    printf("%c", str[i]);
  }
  printf("\n");
}

static int comp(void const *const str1, void const *const str2) {
  return strcmp(*(char **)str1, *(char **)str2);
}

typedef void (*TFunc)(const char arr[]);

// Executes a function pointer for each element of an array
static void CallFuncPointer(TFunc func, const char *arr[]) {
  unsigned i = 0;
  for (; i < 4; i++) {
    func(arr[i]);
  }
}

// Tests function pointers
int TestFuncPtr() {

  const char *unsorted[] = {(char *)"Hello", (char *)"Martha", (char *)"Anton",
                            (char *)"Berta"};

  void (*func)(const char arr[]) = PrintLength;

  PrintHeader("Test Function Pointers");

  // Sorts length
  // sorts using qsort with the comp function as comparator
  qsort(unsorted, 4, sizeof(char *), comp);
  // Cast so constness is preserved
  PrintStrArr(unsorted, 4);

  CallFuncPointer(PrintBackward, unsorted);

  // give callback as variable
  CallFuncPointer(func, unsorted);

  return 0;
}
```

1.2 Test

Terminal Output

```
flashfish@fedora ~/D/R/F/B/U/c_testprog2 (main)> ./cTest

Test Format IO
==============
i: 1 Address: 0x7fff22a531ec
Bitte eingeben: 1
1

Geben sie einen Integer ein:2
2
0x7fff22a531e8
A
PI: 3.141500
Format IO Test OK


Test Strings
============
Length of "Das ist ein C-String" is 20 chars
Buffer: Das ist ein C-String
Text vorher : ABC
Text nachher: CAB
String Test OK


```

```
Test dynamic Memory
===================
 -> Hello World!
Memory Test OK


Test Structs
============
Person: Max Mustermann hat 80 kg
Person: Moritz Mustermann hat 80 kg
Struct Test OK


initialized array with 0:
=========================
0
0
0
0
0
0
0
0
0
0

initialized array with 1:
=========================
16843009
16843009
16843009
16843009
16843009
16843009
16843009
16843009
16843009
16843009
Array Test OK


Test Function Pointers
======================
Anton
Berta
Hello
Martha
notnA
atreB
olleH
ahtraM
Length of "Anton" is 5 chars
Length of "Berta" is 5 chars
Length of "Hello" is 5 chars
Length of "Martha" is 6 chars
FuncPtr Test OK
```

Beispiel 2

1 Frage: Warum sind einige Funktionen als static deklariert?

Das keyword static bei Funktionen sorgt dafuer, dass der Linker weiss, dass diese Funktion nur in der jeweiligen .c sichbar sein soll.
Es sichert also, internal linkage zu. Wenn man dies nicht macht, koennte man mit dem keyword extern in anderen .c Dateien auf diese Funktion zugreifen.
Static verhindert also Namenskonflikte und sorgt fuer eine bessere Kapselung.

Generell kann man sagen, dass alle Funktionen, die nicht in der .h Datei deklariert sind, static sein sollten.

2 Frage: Welches Programmier-Prinzip laesst sich mit Hilfe von static-Funktionen umsetzen?

Mit static Funktionen laesst sich also Information Hiding umsetzen, bzw man kann damit private Funktionen von der Objektorientierung nachstellen