C/C++ Materialpaket (Level C)
11a_PUTT – Putting it all together (Aufgaben)

Prof. Dr. Carsten Link

Zusammenfassung

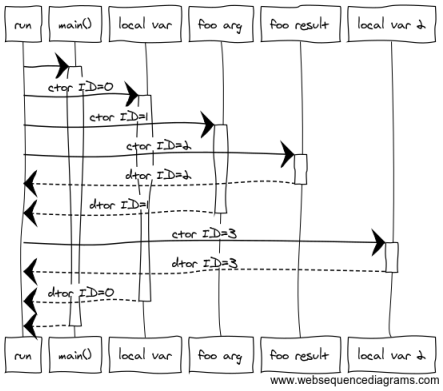


Abbildung 1: Noch kein Piktogram vorhanden

Inhaltsverzeichnis

1 Kompetenzen und Lernegebnisse

Durch das Bearbeiten dieses Materialpaketes erwerben Sie diese Kompetenzen (Wissen, Fähigkeiten, Fertigkeiten zur Problemlösung):

Sie können die isoliert vorgestellten Inhalte der vorangegangenen Materialpakete in Kombination anwenden.

Die oben genannten Kompetenzen erwerben Sie, indem Sie Lernziele erreichen, welche sich prüfen lassen. Lernegebnisse: Sie können nachweislich[1]:

- Aufgaben, wie sie in einer Prüfung gestellt werden können und die alle vorangegangenen Themenbereiche betreffen, erfolgreich bearbeiten
- Fragen, wie sie in einer Prüfung gestellt werden können und die alle vorangegangenen Themenbereiche betreffen, richtig beantworten

2 Konzepte

Im Folgenden wird kein neuer Inhalt dargestellt. Vielmehr sollen Sie alles bereits Gelernte miteinander kombinieren, um les- und wartbare, effiziente C++-Programme entwickeln zu können.

3 Prüfungsvorbereitung

In diesem Materialpaket finden sich Aufgaben und Verständnisfragen, die über jene hinausgehen, die sich in den vorangegangenen Materialpaketen befinden, da sie Wissen und Fertigkeiten benötigen, die über das im jeweiligen Materialpaket Vorgestellte hinausgehen.

3.1 Aufgaben zu 01_ENV

3.1.1 Bauen eines fremden Projektes

Inspizieren Sie den Ordner `src_cpp_student/11_PUTT` und erstellen eine Datei `build.sh`, welche die Testprogramme baut.

3.1.2 (Level C) Toolchain mit dem Portable C Compiler

Der gesamte Übersetzungsvorgang soll anhand eines kleinen C-Programmes illustriert werden. Zur Übersetzung wird `pcc` (The Portable C Compiler[2]) verwendet,

[1] Sie können das Erzielen der einzelnen Lernergebnisse beispielsweise bei einem Testat im Praktikum oder einer Aufgabe in der Modulprüfung nachweisen

[2] Portable C Compiler http://pcc.ludd.ltu.se

da dieser einfache Compiler sehr übersichtlichen Assemblercode erzeugt.

Wichtig: im folgenden Abschnitt sollen sie die Werkzeuge kennen lernen und eine Vorstellung davon entwickeln, welche Art von Zuständigkeiten diese jeweils haben. Es ist nicht nötig, jedes Detail zu verstehen – ein grober Überblick reicht völlig aus.

Das nachfolgende Programm verfügt über die Hauptfunktion `main()` (welche vom Betriebssystem im Zusammenspiel mit den Standardbibliotheken aufgerufen wird), die Funktion `sum()`, sowie eine globale Variable `int global`.

```c
#include <stdio.h>

int global = 8150;

int sum(int a, int b){
  int result = 451;
  result = a + b;
  return result;
}

int main(int argc, char **argv)
{
  int local=4711;
  printf("Hello, world!\nglobal=%d local=%d\n", global, local);
  local = sum(global, local);
  return local;
}
```

Das obige Programm ist in der Datei `main.c` gespeichert. Übersetzt wird es mit der Datei `build.sh`:

```
#!/bin/sh
# generate main.s, main.o, b.out
pcc -Wmissing-prototypes -O0 -S main.c
as -o main.o main.s
pcc -o b.out main.o
# generate a.out
pcc -O0 -g main.c
# generate assembly intermixed with source code
objdump -S a.out > objdump-S_a.out.txt
```

Das oben angegeben Shell Script startet mehrere Werkzeuge um diverse Übersetzungen zu erhalten.

- `pcc -O0 -S main.c` erstellt eine Assemblerdatei `main.s`, wobei `-Wmissing-prototypes` dafür sorgt, dass der Compiler eine Warnung ausgibt, sofern Funktionen ohne zugehörige Deklaration benutzt werden
- `as -o main.o main.s` übersetzt `main.s` in die Objektdatei main.o. Diese

Objektdatei enthält die ausführbaren Funktionen aus `main.c`, jedoch fehlen Funktionen aus den Standardbibliotheken (Zur Ausgabe mit `printf` und zum Starten und Beenden des Programms)

- `pcc -o b.out main.o` nutzt `pcc` als Treiber (driver), um vom Linker `ld` das vollständige (ausführbare) Programm `b.out` erstellen zu lassen.
- `pcc -O0 -g main.c` nimmt nicht den Umweg über eine Assemblerdatei; `a.out` (default name) wird direkt mittels Assembler und Linker erstellt (pcc als Treiber). Die Option `-g` sorgt dafür, dass die Ausgabe `a.out` mit Debug-Informationen versehen wird; die Option `-O0` sorgt dafür, dass der Compiler keine Optimierungen vornimmt

Bei der Entwicklung eines Programmes kann der Programmierer oft auf die Dateien der Zwischenstufen Präprozessor und Compiler verzichten, so dass aus jeder C- oder C++-Datei direkt eine `.o`-Datei erzeugt wird, die dann zusammen an den Linker übergeben werden, um die ausführbare Datei `a.out` zu erzeugen (beispielsweise `pcc -c a.c`, `pcc -c b.c`, `pcc a.o b.o`).

Im Folgenden wird gezeigt, wie sich der C-Code im generierten Maschinencode der ausführbaren Datei widerspiegelt. Das Kommando `objdump -S a.out` stellt den Assemblercode wieder her und mischt diesen mit dem ursprünglichen Quelltext (falls debug info in der ausführbaren Datei vorhanden ist).

```
08048468 <sum>:
#include <stdio.h>

int global = 8150;

int sum(int a, int b){
 8048468:   c8 08 00 00             enter  $0x8,$0x0
  int result = 451;
 804846c:   c7 45 fc c3 01 00 00    movl   $0x1c3,-0x4(%ebp)
  result = a + b;
 8048473:   8b 45 08                mov    0x8(%ebp),%eax
 8048476:   03 45 0c                add    0xc(%ebp),%eax
 8048479:   89 45 fc                mov    %eax,-0x4(%ebp)
  return result;
 804847c:   8b 45 fc                mov    -0x4(%ebp),%eax
 804847f:   89 45 f8                mov    %eax,-0x8(%ebp)
 8048482:   eb 00                   jmp    8048484 <sum+0x1c>
}
 8048484:   8b 45 f8                mov    -0x8(%ebp),%eax
 8048487:   c9                      leave
 8048488:   c3                      ret
 8048489:   8d 76 00                lea    0x0(%esi),%esi

0804848c <main>:
```

```
int main(int argc, char **argv)
{
 804848c:   c8 08 00 00             enter  $0x8,$0x0
  int local=4711;
 8048490:   c7 45 fc 67 12 00 00    movl   $0x1267,-0x4(%ebp)
  printf("Hello, world!\nglobal=%d local=%d\n", global, local);
 8048497:   ff 75 fc                pushl  -0x4(%ebp)
 804849a:   ff 35 1c a0 04 08       pushl  0x804a01c
 80484a0:   68 58 85 04 08          push   $0x8048558
 80484a5:   e8 06 fe ff ff          call   80482b0 <printf@plt>
 80484aa:   83 c4 0c                add    $0xc,%esp
  local = sum(global, local);
 80484ad:   ff 75 fc                pushl  -0x4(%ebp)
 80484b0:   ff 35 1c a0 04 08       pushl  0x804a01c
 80484b6:   e8 ad ff ff ff          call   8048468 <sum>
 80484bb:   83 c4 08                add    $0x8,%esp
 80484be:   89 45 fc                mov    %eax,-0x4(%ebp)
  return local;
 80484c1:   8b 45 fc                mov    -0x4(%ebp),%eax
 80484c4:   89 45 f8                mov    %eax,-0x8(%ebp)
 80484c7:   eb 00                   jmp    80484c9 <main+0x3d>
}
 80484c9:   8b 45 f8                mov    -0x8(%ebp),%eax
 80484cc:   c9                      leave
 80484cd:   c3                      ret
 80484ce:   66 90                   xchg   %ax,%ax
```

Es ist zu sehen, dass beide Funkionen `main()` und `sum()` mit `enter` beginnen und mit `leave` enden. Dies dient dem Auf- bzw. Abbau des Aktivierungsrecords[3]. Dadurch erhalten lokale Variablen Speicherplatz und rekursive Aufrufe sind möglich, ohne Daten von anderen Ausprägungen der jeweiligen Funktionen zu überschreiben. Funktionsaufrufe werden mit `call` umgesetzt, die aufgerufene Funktion lässt die CPU mit `ret` zum Aufrufer zurückspringen. Hierzu hat `call` die Adresse des nachfolgenden Befehls auf den Call Stack gelegt und ret lädt diesen in den Instruction Pointer.

Aufgaben:

1. Erweitern Sie das oben angegebene Programm um eine lokale Variable `int lineLocator` und weisen in möglichst vielen Quelltextzeilen dieser Variable den Macro-Wert `__LINE__` zu. Übersetzen Sie mit `pcc -O0 -S` und inspizieren die Ausgabe.
2. Welche Veränderungen ergeben sich, wenn Sie lokale Variablen hinzufügen? Verwenden Sie `meld <file 1> <file 2>`, `diff` oder `Diffuse Merge Tool`, um Änderungen hervorheben zu lassen.

[3] https://en.wikipedia.org/wiki/Function_prologue

3. **Level C:** Was ändert sich im Assemblercode (`objdump -S`), wenn Sie einer Funktion eine lokale Variable hinzufügen?
4. Welche Folgen kann es haben, wenn eine Funktion aus einer anderen Übersetzungseinheit (`.c` zu `.o`) aufgerufen wird, ohne dass der Compiler vorher die dazugehörige Deklaration gesehen hat (warning: implicit declaration / missing prototype)?

3.2 Aufgaben zu 02_DATA

1. Bauen Sie eine `struct TinyAsciiString`, welche Strings aus 16 ASCII-Zeichen aufnehmen kann. Da ASCII-Zeichen nur sieben Bit benötigen, können Sie auf das Längenfeld verzichten und die für die Länge benötigten vier Bit in den Zeichen des gespeicherten String unterbringen
2. Laden sie das virtuelle LC-Display[4] herunter und experimentieren Sie damit. Erstellen Sie ein Programm, welches einen `int` hochzählt und diesen mittig auf dem Display darstellt.
3. Erstellen Sie eine `struct IpV4Address`, welche IPv4-Adressen aufnehmen kann. Intern wird ein 32-Bit-`int` verwendet. Es sollen die Methoden `fromString()`, `toString()` sowie der subscript-Operator implementiert werden (alle sollen unabhängig von der byte order der CPU arbeiten)
4. Erstellen Sie eine Funktion `std::string toPrettyString(double)`, welche Ausgaben erzeugt, wie Sie sie von Taschenrechnern gewophnt sind
5. Erstellen Sie Funktionen oder eine Klasse mit Methoden und operatoren, mit der Sie Zugriff auf ein 3-dimensionales Array erlauben. Intern soll jedoch ein 1-dimensionales Array verwendet werden. Die Größe wird bei der Initialisierung angegeben.
6. Implementieren Sie die Berechnung sowie die Prüfung von ISBN-10-Prüfziffern

3.3 Aufgaben zu 03_FLOW_a

1. Implementieren Sie den FloodFill-Algorithmus rekursiv. Erstellen Sie hierzu z. B. ein zweidimensionales `char`-Array (ggf. `std::array<>`), welches mit Leerzeichen, einem Rahmen und Hindernissen gefüllt ist.
2. Implementieren Sie den FloodFill-Algorithmus iterativ. Hinweis: Sie werden einen Zwischenspeicher benötigen.
3. Tragen Sie die durchnummerierten Schritte für den Aufruf `recurse(4)` farbig in diesen Code ein:

```
53
54  void recurse(int turns){nop;
55    colorOffset++; nop;
56    nop; if(turns>0) { nop;
57      recurse(turns - 1);
```

[4] http://www.technik-emden.de/~clink/projects/2016w-ProjGrp/05_Website_ohne_Quellcodes/11_Dokumente/13_Doxygen/doc/html/index.html

```
58      nop;}
59    colorOffset--; nop;
60    nop;}
61
```

5. Betrachten den nachfolgenden Code und beantworten diese Fragen:
   - Welche Ausgabe ergibt sich bei `exec state 2` für den Aufruf `insertionSort("7456")` ?
   - Welche Ausgabe ergibt sich bei `exec state 3` für den Aufruf `insertionSort("7456")` ?

```
// based on https://www.geeksforgeeks.org/insertion-sort/
std::string insertionSort(std::string chars){
  char currentChar;
  size_t i, k;
  for (i = 1; i < chars.length(); i++){
    currentChar = chars[i];
    println("exec state ", i, " : currentChar = '", currentChar,
            "' chars = \"", chars, "\"");

    // Move elements of "chars[0..i-1]", that are
    // greater than "currentChar", to one position ahead
    // of their current position (move ahead == move right).
    // starting in the middle of "chars" going downwards (i.e.
    // chars[i-1] .. chars[0]).
    k = i - 1;
    while (k >= 0 && chars[k] > currentChar){
      chars[k + 1] = chars[k];
      k = k - 1;
    }
    chars[k + 1] = currentChar;
  }
  println("exec state ", i, " : currentChar = '", currentChar,
          "' chars = \"", chars, "\"");
  return chars;
}
```

6. Verwenden Sie Teile von `std::filesystem`, um eine gegebene Verzeichnishierarchie rekursiv abzuwandern.
7. Verwenden Sie Teile von `std::filesystem` und `std::deque`, um eine gegebene Verzeichnishierarchie iterativ abzuwandern.

3.4 Aufgaben zu 03_FLOW_c

1. Rechner für geklammerte Ausdrücke ohne Rekursion: Implementieren Sie den Rechner für geklammerte boolsche Ausdrücke ohne Rekursion. Benut-

zen Sie stattdessen `std::verctor<>`, um zwei Stacks zu realisieren.

3.5 Aufgaben zu 04_UDEF

1. Fügen Sie der Klasse `RgbColor` den Subscript-Operator hinzu, so dass über die Indizes `0` bis `2` auf die RGB-Werte zugegriffen werden kann. Achten Sie auf den Rückgabewert der Operatorenüberladung! Experimentieren Sie mit verschiedenen Möglichkeiten. welche Auswirkungen hat dies jeweils?

3.6 Aufgaben zu 05_OO_a

1. (Level C) Führen Sie zwischen der Klasse `Shape` und den davon abgeleiteten Klassen eine Klasse `ColoredShape` ein, welche die Farbverwaltung übernimmt. Mittels der Methode `int ColoredShape::amountPaintNeeded()` kann ermittelt werden, wie viele Zeichen gemalt werden würden. Die Methode `amountPaintNeeded()` soll nicht virtuell sein, kann aber virtuelle Methoden verwenden

3.7 Aufgaben zu 05b_OO_ENTVAL

3.7.1 Copy on Write

Um Werte in einem Programm zu Speichern, welche sehr viel Speicherplatz benötigen, bietet sich das Idiom bzw. Design Pattern *copy on write* an. Hierbei verwenden mehrere Kopien ein und denselben Speicherplatz; allerdings nur so lange, bis eine Änderung (*write*) durchgeführt wird: dann wird für das Modifizierte Objekt eine eigene Kopie erstellt, welche dann verändert wird.

Aufgabe: nehme Sie die Dateien `OneByOneMatrix.h`, `OneByOneMatrix.cpp` und `main.cpp` im Verzeichnis `11_PUTT/CopyOnWrite/` als Vorlage.

Compilieren Sie nun mit den `typedef`s `int` und `OneByOneMatrix` und lassen das Programm jeweils laufen.

```
1 typedef int NumberType;
2 //typedef OneByOneMatrix NumberType;
3 //typedef LargeCowMatrix NumberType;
```

Compilieren Sie nun mit dem `typedef LargeCowMatrix`. Verändern Sie den Code, so dass keine Fehler beim Übersetzen auftreten und keine Fehler zur Laufzeit auftreten.

3.8 Aufgaben zu 05c_OO_CYCL

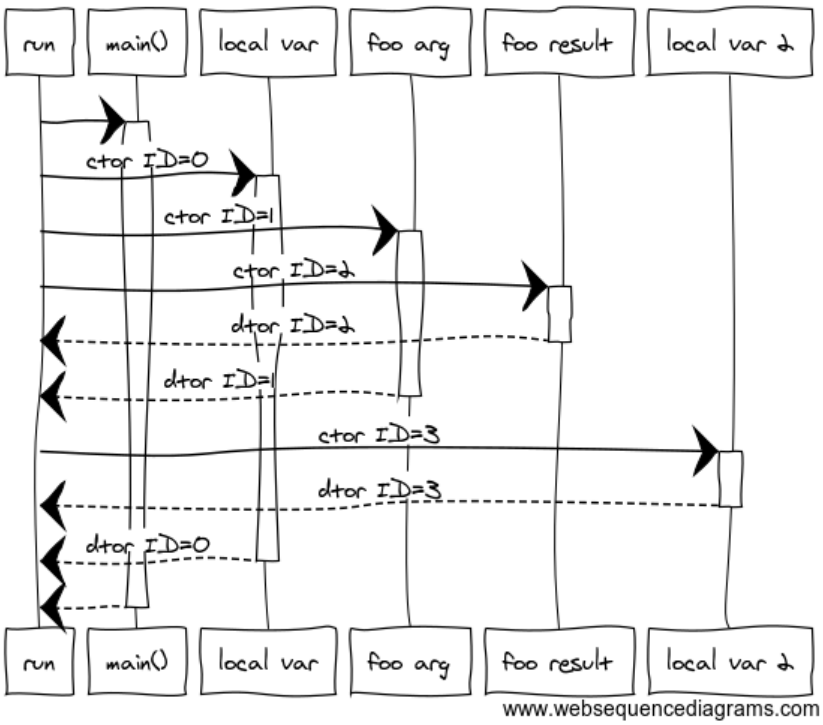
1. Betrachten Sie folgenden Code:

```
1 struct A{
2 };
3
```

```
4  struct B{
5  };
6
7
8  class K {
9   A a;
10 };
11
12 class M : public K {
13  B b;
14 };
15
16 M * m =  new M();
17 delete m;
```

In welcher Reihenfolge werden welche special member functions aufgerufen?

2. Auf der Web-Seite https://www.websequencediagrams.com lassen sich UML-Sequenzdiagramme erstellen, welche in Textform definiert sind. Diese lassen sich zweckentfremden, um die Lebensdauer von Objekten zu visualisieren (siehe nachfolgende Abbildung Websequence-Diagrams). Erstellen Sie eine Klasse `SequenceDiagramCreator`, welche im Konstruktor und Destruktor dafür sorgt, dass Text ausgegeben wird, der von `websequencediagrams.com` interpretiert werden kann.



Experimentieren Sie anschließend mit globalen, lokalen und statischen Variablen. Ebenso sollten Sie die Parameterübergabe erkunden.

3. Kann eine Klasse `X` folgendes enthalten (ja/nein; mit Begründung; falls ja: was ist zu beachten?; falls nein: warum nicht?):
   - ein Objekt der Klasse `X`
   - eine Referenz auf ein Objekt der Klasse `X`
   - ein Pointer auf ein Objekt der Klasse `X`

3.9 Aufgaben zu 06_BIND

1. Erstellen Sie jeweils ein kurzes einfaches Beispielprogramm für
   a. ad-hoc polymorphism
   b. subtyping polymorhism
   c. parametric polymorphism
2. Betrachten Sie folgende Klassendeklarationen:

```
1 class TimePiece {
2 public:
3   virtual ~TimePiece();
4   void foo();
5   virtual std::string toString();
6 };
```

```
7
8  class AnalogWatch : public TimePiece {
9  public:
10   void foo();
11   void bar();
12   virtual std::string toString();
13 };
14
15 class DigitalWatch : TimePiece {
16 public:
17   void foo();
18   void bar();
19   virtual std::string toString();
20 };
```

Was geschieht bei den folgenden Anweisungen?

```
1  TimePiece * ta = new AnalogWatch();
2  TimePiece * td = new DigitalWatch();
3  TimePiece * tp = ta;
4
5  tp->toString();
6  tp->foo();
7  ta->foo();
8  td->foo();
9  td->bar();
10 tp->bar();
```

3. Entwickeln Sie ein kleines Programm, welches Funktionszeiger verwendet, um die Häufigkeit von Buchstaben, Ziffern, und sonstigen Zeichen in einem `std::string` zu ermitteln. Sie können beispielsweise ein Array anlegen, in dem für jedes Zeichen ein Funktionszeiger hinterlegt ist (die Funktionen hinter den Zeigern könnten Variablen hochzählen).

3.10 Aufgaben zu 07_STD

1. Verwenden Sie `IntStack` oder `std::vector<>`, um den Callstack nachzubilden. Das heißt: erstellen Sei einige `void/void`-Funktionen (ohne Parameter und Rückgabewert), welche sich gegenseitig aufrufen (gern auch rekursiv, ggf. indirekt) und die Parameter und Rückgabewerte über eine Instanz von `IntStack` oder `std::vector<>` austauschen.

4 Nützliche Links

- Stack Overflow: http://stackoverflow.com
- C++ reference: http://en.cppreference.com/w/

- C++ Referenz: http://de.cppreference.com/w/
- Programmieraufgaben: https://adventofcode.com
- Programmieraufgaben: https://www.codechef.com
- Programmieraufgaben: https://leetcode.com
- Programmieraufgaben: https://www.codewars.com

5 Literatur

- [PPP] Stroustrup, Bjarne: Programming - Principles and Practice using C++
- [TCPL] Stroustrup, Bjarne: The C++ Programming Language, Fourth Edition
- [CUEB] U. Kirch, P. Prinz: C++ das Übungsbuch, Testfragen und Aufgaben mit Lösungen