Fakultät IV - Elektrotechnik und Informatik
Einführung in die Programmierung WS 2017/18
Feldmann / Foucard / Lichtblau / Streibelt / Krüger / Enghardt / Palmer

Aufgabenblatt 8

letzte Aktualisierung: 26. Oktober, 13:24 Uhr

(cf2848c986fbff15bbcef2931dc19bc6a040e443)

Ausgabe: Donnerstag, 26.10.2017
Abgabe: keine Abgabe

Thema: *Debugging*

Abgabemodalitäten

1. Die Aufgaben des C-Kurses bauen aufeinander auf. Versuche daher bitte Deine Lösung noch am gleichen Tag zu bearbeiten und abzugeben.
2. Alle abzugebenden Quelltexte müssen ohne Warnungen und Fehler auf den Rechnern des tubIT/eecsIT mittels `gcc -std=c99 -Wall` kompilieren.
3. Die Abgabe erfolgt ausschließlich über unser SVN im Abgaben-Ordner. Nur wenn ein Test in Osiris angezeigt wird ist sichergestellt, dass die Abgabe erfolgt ist.
4. Du kannst bis zur Abgabefrist beliebig oft neue Versionen abgeben.
5. Die Abgabe erfolgt in folgendem Unterordner:
   `ckurs-wise1718/Studierende/<L>/<tubIT-Login>/Abgaben/Blatt0<X>`
   wobei `<L>` durch den ersten Buchstabe des TUBIT-Logins und `<X>` durch die Nummer des Aufgabenblattes zu ersetzen sind. Die Ordner werden automatisch angelegt sobald die Abgabe freigeschaltet wird.
6. Benutze für alle Abgaben soweit nicht anders angegeben das folgende Namensschema: `ckurs_blatt0<X>_aufgabe0<Y>.c` wobei `<X>` und `<Y>` entsprechend zu ersetzen sind. Gebe für jede Unteraufgabe genau eine Quellcodedatei ab.
7. Du darfst den Abgabeordner für das Blatt nicht selbst erstellen, das machen wir jeden Morgen kurz nach 8 Uhr!
8. Du musst aber den Befehl `svn up` auf der obersten Verzeichnisebene des Repositories (also in `ckurs-wise1718`) ausführen um alle Änderungen vom Server abzuholen.
9. Im Abgaben-Ordner gelten einige restriktive Regeln. Dort ist nur das Einchecken von Dateien mit den in der Aufgabe vorgegebenen Namen erlaubt, ausserdem werden die Abgabefristen vom Server überwacht. Beachte eventuelle Fehlermeldungen beim SVN-Commit. Lade nur Dateien hoch, die Du selbst bearbeiten sollst, insbesondere also keine Vorgaben.
10. Es gibt einen Ordner 'Arbeitsverzeichnis', in dem Du Dateien für Dich ablegen kannst.
11. Die Ergebnisse der automatischen Tests kannst Du auf OSIRIS einsehen:
    https://teaching.inet.tu-berlin.de/services/osiris-wise1718/

Hinweis: Alle Aufgaben auf diesem Übungsblatt sind unbewertet. Ziel dieses Übungszettels ist es vorzustellen wie in `C` mithilfe `printf` und `gdb` die Funktionsweise von Programmen nachvollzogen und Fehler effektiv gefunden werden. Es gibt für diese Aufgaben keinerlei automatische Tests. Diese Aufgaben müssen nicht in das SVN geladen werden.

1. Aufgabe: Erste Schritte mit `gdb` (unbewertet)

Listing 1: ckurs_blatt08_aufgabe01.c

```
1  #include<stdio.h>
2
3  int main()
4  {
5      // Gib die Fakultät von n aus
6      int i,n = 10;
7      int produkt = 1;
8      for(int i = 2; i < n; ++i);
9      {
10         produkt = produkt * i;
11     }
12     printf("Die␣Fakultät␣von␣n␣=␣%d␣und␣damit␣das␣Produkt␣der␣Zahlen␣([%d,
           ↪ ␣...,␣%d])␣ist␣%d\n", n, 1, n, produkt);
13 }
```

Das in Listing 1 dargestellte Programm soll die Fakultät von `n` berechnen. Seltsamerweise kommt ein unerwartetes Ergebnis. Überlege dir:

1. Was für ein Ergebnis hast du erwartet?
2. Kompiliere das Programm und führe es aus. Was für ein Ergebnis bekommst du?
3. Wo würdest du am Besten ein `printf` einfügen um, die Funktionsweise des Programms nachzuvollziehen? Welche Variablen würdest du ausgeben? (Vielleicht hast du schon eine Idee, was das Problem sein könnte. Versuche deine Idee direkt am Programm zu testen, indem du ein oder mehrere `printf` einfügst.)
4. Hast du die Lösung gefunden? Kannst du dir vorstellen, wie das Hinzufügen von `printf` gerade für größere Programme zur Unübersichtlichkeit führt?

Anstelle von `printf` wollen wir `gdb` verwenden. Compiliere den Code dabei mit der zusätzlichen `-g` Option wie in Listing 2. Ignoriere erstmal die Warnung des Compiler.

Listing 2: Compilieren mit Debugoption

```
> gcc -std=c99 -Wall -g ckurs_blatt08_aufgabe01.c -o
    ↪ ckurs_blatt08_aufgabe01
...
```

Jetzt können wir das Programm mit `gdb` aufrufen wie in Listing 3 gezeigt.

Listing 3: Starten von GDB

```
> gdb ./ckurs_blatt08_aufgabe01
GNU gdb (GDB) 7.9.1
Copyright (C) 2015 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.
    ↪ html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-unknown-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./ckurs_blatt08_aufgabe01...done.
(gdb)
```

Wenn alles gut geht siehst du nun ein `(gdb)` in deiner Eingabezeile deinem Terminal. (Wenn du ein `Command not found` bekommst, musst du `gdb` erst installieren. Auf den Uni Rechner sollte `gdb` vorinstalliert sein.)

Du kannst jetzt hinter dem Text `(gdb)` Komandos eingeben. Du beendest `gdb` jederzeit, indem du `quit` eingibst. Als nächstes kannst du:

1. Mehrfach `list` eintippen um dir den Code inklusive Zeilennummern auf dem Terminal auszugeben.
2. Suche dir die Zeile indem `produkt` erhöht wird. (Ungefähr Zeile 10.)
3. Teile `gdb` mit, dass du an dieser Stelle anhalten willst, indem du `break` gefolgt von einem Leerzeichen und der Zeilennummer eingibst.

```
(gdb) break 10
Breakpoint 1 at 0x400531: file ckurs_blatt08_aufgabe01.c, line 10.
```

4. Führe das Programm mit `run` aus. Es hält an der angegebenen Stelle.

```
(gdb) run
Starting program: ...

Breakpoint 1, main () at ckurs_blatt08_aufgabe01.c:10
10          produkt = produkt * i;
```

5. Du kannst dir nun mit `print` den Inhalt der Variablen ausgeben. Überlege welche Variable für dich interessant ist.

```
(gdb) print produkt
$2 = 1
```

6. Mithilfe von `continue` springst du zum nächste Mal, in dem der Code aufgerufen wird. Wie oft klappt das?
7. Statt `continue` kannst du mit `next` das Programm bis zum Ende Zeile zu Zeile durchgehen.
8. Wenn du am Ende angekommen bist wird dir das `gdb` mitteilen:

```
(gdb) continue
Continuing.
Die Fakultät von n = 10 und damit das Produkt der Zahlen ([1, ...,
    ↪ 10]) ist -6752
[Inferior 1 (process 18142) exited normally]
```

9. Du kannst nun:
   - Den Punkt an dem `gdb` das Programm anhält mit `clear` löschen und einen neuen mit `break` definieren.
   - Das Programm neu mit `run` ausführen.
   - `gdb` beenden mit `quit`

Das war es auch schon, damit bist du bestens ausgerüstet um die folgenden Programme zu debuggen. Aber du wirst merken, dass `gdb` viele kleine Abkürzungen und noch viel mehr Funktionalität bereithält. Informationen zu den Befehlen (z.B. `break`) bekommst du via `help break`.

2. Aufgabe: Weiter mit gdb und warum Einrücken hilft (unbewertet)

Listing 4: ckurs_blatt08_aufgabe02.c

```
1  #include<stdio.h>
2
3  // Vergleicht die beiden übergebenen Zahlen und liefert 1 zurück
4  // falls beide Zahlen gleich sind
5  int equals(int a, int b)
6  {
7  if(a < b){
8  return 0;
9  if(a > b)
10 return 0;}
11 else{
12 return 1;
13 }
14 }
15
16 int main()
17 {
18     int n = 10;
19 for(int i = 1; i <= n; ++i)
20 {
21 for(int j = 1; j <= n; ++j)
22 {
23 if(equals(i,j))
24 printf("%2d == %2d : TRUE\n", i,j);
25 else
26 printf("%2d == %2d : FALSE\n", i,j);
27 }
28 }
29
30 }
```

Das Programm soll ausgeben, ob die Zahl `n` gerade oder ungerade ist. Was macht diese Aufgabe so unübersichtlich? Wie könnte man sie übersichtlicher gestalten?

Kannst du dir vostellen wo das Problem liegt? Teste deine Idee mit `printf` oder `gdb`, wie oben beschrieben.

3. Aufgabe: Verständnis vereinfachen, die Kraft der Kommentare (unbewertet)

Listing 5: ckurs_blatt08_aufgabe03.c

```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int n = 100;
    // Es ist hier wichtig genau n+1 viele chars zu reservieren, da
        ↪ strings mit
    // einer '\0' abgeschlossen werden müssen
    char* string = malloc( sizeof(char) * (n+1));

    int i = 0;
    for(; i < n; ++i)
    {
        if(i%2 == 0)
        {
            string[i] = 'z';
        }
        else
        {
            string[i] = 'Z';
        }
        if(n = i)
        {
            string[i] = '\0';
        }
    }
    printf("Das passiert, wenn ich schlafe:\n%s", string);

}
```

Das Programm soll einen String der Länge 100 ausgeben, der aus `zZ` Buchstaben besteht. Verstehst Du was das Programm macht? Kannst du mit Kommentaren das Programm einfacher verständlich machen?

Find den Fehler indem du dichj mit `printf` oder `gdb` Schritt für Schritt durch das Programm arbeitest.

4. Aufgabe: Kryptische Variablennamen (unbewertet)

Listing 6: ckurs_blatt08_aufgabe04.c

```c
#include<stdio.h>

// Gibt die Zahl x^x für alle Zahlen zwischen 1 und n aus
//
// 1^1 = 1 = 1
// 2^2 = 2*2 = 4
// 3^3 = 3*3*3 = 27
// 4^4 = 4*4*4*4 = 256
// ...
// n^n
int main()
{
    int n = 20;

    for(int x = 1; x <= n; ++x)
    {
        int f_x = 0;
        int benoetigte_multiplikationen = f_x - 1;
        while(benoetigte_multiplikationen > 0)
        {
            int f_x = x;
            f_x *= f_x;
        }
        printf("%d^%d = %d\n", x,x,x,f_x);
    }

}
```

Dieses Programm soll für alle Zahlen 1 bis n die Zahl n^n ausgeben. Kannst du bessere Variablennamen finden? Teste dann deine Hypothese wieder mit `gdb` oder `printf`.