Lösungen zu den Übungen vom 24.10.2007

Ralph Gauges Ursula Rost Katja Wegner

30. Oktober 2007

Aufgabe 1:

Beim Schreiben von Programmen macht man selbstverständlich auch Fehler und seien es nur Tippfehler. Besonders Sprachen wie C++, welche sehr komplex sind, laden förmlich dazu ein Fehler zu machen. Da sich dies, insbesondere am Anfang, nicht verhindern lässt, ist es umso wichtiger, Fehler schnell zu finden und zu beseitigen. Die Fehlermeldungen des Präprozessors, des Compilers und des Linkers sind hierbei eine große Hilfe. Man sollte sich deshalb frühzeitig angewöhnen, genau auf solche Fehlermeldungen zu achten. Mit der Zeit lernt man dann welche Fehlermeldungen für welche Fehler typisch sind und kann diese schneller beseitigen.

Es ist ebenfalls immer ratsam, die Option -Wall beim kompilieren zu verwenden. Die -Wall Option veranlasst den Compiler zusätzliche Warnungen auszugeben, falls er auf Programmcode trifft, welcher zwar gültiges C++ darstellt, aber potentiell zu Problemen führen kann. Auch diese Warnungen sollte man ernst nehmen und die Ursachen nach Möglichkeit beseitigen.

Geben Sie folgende drei Versionen des "HelloWorld"Programms in den Texteditor Ihrer Wahl ein und übersetzen Sie sie. Jedes dieser Programme enthält einen Fehler und für jede Version des Programme unten kommt die Fehlermeldung von einem anderen Programm. Versuchen Sie herauszufinden welches Programm bzw. welche Programmkomponente (Präprozessor, Compiler, Linker) den Fehler meldet.

```
1: // HelloWorld 1
2: #include <iostream>
3:
4: int main()
5: {
6:   std::cout << "Hello World!\n";
7:   return 0
8: }
```

Zeile 7: Das Semikolon nach return 0 fehlt! Immer darauf achten, dass jede Kommandozeile mit einem Semikolon abgeschlossen wird. Die Fehlermeldung kommt entsprechend vom Compiler, da es sich um ungültigen C++ Code handelt.

```
1: // HelloWorld 2
2: #include <isostream>
3:
4: int main()
5: {
6:   std::cout << "Hello World!\n";
7:   return 0;
8: }
```

Zeile 2: Die Datei, in der die Funktionen und Objekte deklariert sind welche man für die Textausgabe in der Konsole benötigt, heisst iostream und nicht isostream. In diesem Fall kommt die Fehlermeldung vom Präprozessor. Er bemängelt zurecht, dass er die Datei isostream nicht finden kann.

```
1: // Hello World 3
2: #include <iostream>
3:
4: int moin()
5: {
6:   std::cout << "Hello World!\n";
7:   return 0;
8: }
```

Zeile 4: Die Funktion in der ausführbare Programme beginnen, wenn man sie startet, heisst main und nicht moin! Wir haben hier zwar absolut korrekten C++ Code, da wir jedoch ein ausführbares Programm erzeugen möchten, sucht der Linker in der kompilierten Objektdatei noch dem Symbol main. Da wir die Funktion aber moin genannt haben, sagt uns der Linker, dass er das Symbol main nicht finden kann.

Aufgabe 2:

Schreiben Sie ein Programm mit zwei Variablen a und b vom Typ Integer. Initialisieren Sie a mit dem Wert 10 und b mit dem Wert 15. Tauschen Sie dann die Werte von a und b und geben das Ergebnis aus. Die Ausgabe des Programms sollte folgendermaßen aussehen:

```
a: 15
b: 10
```

```
#include <iostream>

int main()
{
  unsigned int a = 10;
  unsigned int b = 15;
  unsigned int help = a;
  a = b;
  b = help;
  std::cout << "a: " << a << "\n";
  std::cout << "b: " << b << "\n";
  return 0;
}
```

Das Programm initialisiert die Variablen a und b vom Typ int mit den Werten 10 und 15 respektive, wie in der Aufgabe verlangt. Anschließend wird eine dritte Hilfsvariable von selben Typ deklariert und mit dem Wert von a initialisiert. Nun da der Wert von a gesichert ist, kann man a den Wert von b zuweisen und anschließend wird noch b der ursprünglich Wert von a zugewiesen, der aber nun in der Hilfsvariablen help steht. Zum Schluss folgt dann wie in der Aufgabe verlangt die Ausgabe der Werte in a und b.

Aufgabe 3:

Schreiben Sie ein Programm, welches die Zahlen von 1-10 multipliziert und das Ergebnis ausgibt. Überprüfen Sie das Ergebnis mit einem Taschenrechner.

```
#include <iostream>

int main()
{
  int result = 1;
  for (int i = 1; i<= 10; ++i)
  {
    result *= i;
  }
  std::cout << result << "\n";
  return 0;
}
```

Das Programm multipliziert die Zahlen von 1-10 und gibt das Ergebnis (3628800) aus. Im Grunde könnte man den Schleifenzähler i auch von $i = 2$ bis 2 <= 10 laufen lassen und sich so einen Schleifendurchgang sparen, da im ersten Durchgang nur 1 mit sich selbst multipliziert wird. Das Ergebnis dieses Programms sollte mit dem Ergebnis, wie es ein Taschenrechner liefert, übereinstimmen.

Aufgabe 4:

Schreiben Sie ein Programm, welches die Zahlen von 1-15 multipliziert und das Ergebnis ausgibt. Überprüfen Sie das Ergebnis mit einem Taschenrechner.

```
#include <iostream>

int main()
{
  int result = 1;
  for (int i = 1; i <= 15; ++i)
  {
    result *= i;
  }
  std::cout << result << "\n";
  return 0;
}
```

Gegenüber der Lösung aus Aufgabe 4 hat sich nur die Bedingung in der for Schleife geändert. Der Schleifenzähler läuft nun nicht mehr von 1 bis 10, sondern von 1-15. Das Programm gibt (wenn es mit GCC 4.0 auf einer Intel CPU kompiliert wurde) die Zahl 2004310016 aus. Wenn man jedoch das Ergebnis mit einem Taschenrechner überprüft, so wird man feststellen, dass der Taschenrechner als Ergebnis $1.307674368 * 10^{12}$ liefert. Das liegt daran, dass Integer Werte bei GCC4 auf aktuellen Intel CPUs eine Länge von 32 Bit besitzen, dies reicht aber lediglich aus, um im Falle eines unsigned int Zahlen bis zu einer Grösse von ca. $4 * 10^9$ darzustellen. Im Falle eines signed int, welches im Programm verwendet wird sind es die Zahlen von ca. $-2 * 10^9$ bis $+2 * 10^9$. Die richtige Lösung passt in diesem Fall also nicht in den von uns gewählten Datentyp, und das Ergebnis, welches das Programm liefert ist falsch. Wie man feststellt, welchen Wertebereich ein bestimmter Datentyp abdeckt, bzw. wie viele Bytes der Compiler für die jeweiligen Datentypen reserviert, wird in einer der nächsten Stunden behandelt.

Aufgabe 5:

Schreiben Sie ein Programm, welches die Zahlen von 1-10 multipliziert und das Ergebnis ausgibt jedoch ohne dabei eine Form der Multiplikation (*, *=) zu benutzen.

```
#include <iostream>

int main()
{
  int result = 1;
  for (int i = 1; i <= 10; ++i)
  {
      int help = result;
      for (int j = 1; j < i; ++j)
      {
                result += help;
      }
  }
  std::cout << result << "\n";
  return 0;
}
```

Als Grundlage für das Programm dient die Lösung von Aufgabe 3. Da die Aufgabe von uns verlangt, keine Multiplikation zu verwenden, müssen wir die Multiplikation durch etwas äquivalentes ersetzen. Eine Multiplikation ist nichts anders als eine Serie von Additionen. Wir können also die Multiplikation durch eine Serie von Additionen ersetzen, welche wir in einer inneren for Schleife ausführen. Dabei sind zwei Dinge zu beachten. Erstens müssen wir uns den Wert von result in einer Hilfsvariablen merken, da wir ja durch die Addition in der inneren Schleife result selbst verändern. Zum zweiten sollten man darauf achten, dass der innere Schleifenzähler j für eine Multiplikation mit n nur von 1 bis $n-1$ läuft, da result zu Beginn der inneren Schleife ja schon einen Wert besitzt.