Übungsblatt 2

Ralph Gauges Ursula Rost Katja Wegner

31.10.2007

Aufgabe 1:

Schreiben Sie ein Programm, das vom Benutzer eine maximal zweistellige Zahl in Form einer Zeichenkette entgegen nimmt, diese in eine Zahl vom Typ **unsigned int** umwandelt und anschließend den Zahlenwert ausgibt.
Das Programm soll über eine Funktion verfügen, welche überprüft, ob es sich bei der Eingabe um eine Zeichenkette handelt, die eine Zahl darstellt und eine weitere Funktion, welche die Zeichenkette falls möglich in eine Zahl vom Typ **unsigned int** umwandelt.
Handelt es sich bei der eingegebenen Zeichenkette nicht um eine gültige Zahlenzeichenkette, so soll das Programm eine entsprechende Fehlermeldung ausgeben.
Sie können davon ausgehen, daß die eingegebenen Zeichenkette nur Zeichen aus der ASCII Tabelle enthält.

```
1  #include <iostream>
2
3
4  bool istEineZahl(char zeichen1,char zeichen2)
5  {
6    bool result=true;
7    if(zeichen1<48)
8    {
9      result=false;
10   }
11   else
12   {
13     if(zeichen1 > 57)
14     {
15       result=false;
```

```
16      }
17      else
18      {
19        if(zeichen2 < 48)
20        {
21          result=false;
22        }
23        else
24        {
25         if(zeichen2 > 57)
26         {
27           result=false;
28         }
29        }
30      }
31    }
32    return result;
33  }
34
35  unsigned int umwandelnInZahl(char zeichen1,char zeichen2)
36  {
37     unsigned int result=zeichen2 - 48;
38     result+=(zeichen1 - 48) * 10;
39     return result;
40  }
41
42  int main()
43  {
44      std::cout << "Bitte geben Sie eine zweistellige Zahl ein:";
45      char zahl[3];
46      std::cin.getline(zahl,3);
47      if(istEineZahl(zahl[0],zahl[1]) == true)
48      {
49          unsigned int zahlenWert=umwandelnInZahl(zahl[0],zahl[1]);
50          std::cout << "Sie haben die Zahl " << zahlenWert << " eingegeben.\n";
51      }
52      else
53      {
54          std::cout << "Es handelt sich bei der Eingabe nicht um eine Zahl.\n";
55      }
56      return 0;
57  }
```

Zeile 1: Besteht aus einer Präprozessordirektive, die die Datei iostream in das Programm einbindet. Diese enthält die Deklarationen, welche zur Benut-

zung von std::cout und std::cin nötig sind.
Zeile 4 - 33: Deklarieren eine Funktion namens *istEineZahl*, diese hat den Rückgabetyp **bool** und übernimmt zwei Argumente vom Typ **char**. Stellt die Funktion fest, dass es sich bei den übergebenen Zeichen um zwei Zahlenzeichen handelt, liefert sie den Wert **true** zurück, sonst den Wert **false**. Zu Beginn geht die Funktion erst einmal davon aus, dass es sich bei den beiden Variablen *zeichen1* und *zeichen2* tatsächlich um zwei Zahlenzeichen handelt und setzt den Wert der Variablen *result* auf **true**. Nun untersucht die Funktion, ob die zeichen1 und zeichen2 Zahlen im Bereich zwischen 48 und 57 sind. Dieser Bereich entspricht in der ASCII Tabelle den Zeichen '0'-'9'. Ist dies der Fall, so bleibt die Variable *result* auf ihrem ursprünglichen Wert, ist dies nicht der Fall wird *result* auf **false** gesetzt. Von vielen Programmierern wird diese Art die Funktion zu schreiben bevorzugt. Sie ist übersichtlich und es gibt nur eine **return** Anweisung um die Funktion zu verlassen und das Ergebnis zurückzuliefern. Es geht aber auch kürzer:

```
1 bool istEinezahl(char zeichen1,char zeichen2)
2 {
3   if(zeichen1<48) return false;
4   else if(zeichen1 > 57) return false;
5   else if(zeichen2 < 48) return false;
6   else if(zeichen2 > 57) return false;
7   return true;
8 }
```

Diese Version macht genau das Gleiche wie die lange Version oben. Die einzigen Unterschiede sind, dass die ganzen Klammern weggelassen wurden, was erlaubt ist, da der Codeblock in jeder Klammer nur aus einer Zeile bestand (Siehe dazu auch die Erläuterungen zu Aufgabe 2). Zweitens verwendet die neue Version nicht nur eine **return** Anweisung am Ende, sondern liefert sofort an der Stelle das Ergebnis zurück an der es feststeht. Später wenn wir noch mehr der C++ Operatoren behandelt haben, lässt sich die ganze Funktion auch in einer Zeile (ohne unübersichtlich zu werden) schreiben.
Zeile 35 - 40: Hier wird die Funktion *umwandelnInZahl* definiert. Diese hat den Rückgabewert **unsigned int** und übernimmt zwei Argumente von Type **char**. Die beiden Argumente zeichen1 und zeichen2 sind dabei die beiden Zeichen aus denen die Zahlenzeichenkette zusammengesetzt ist, wobei zeichen1 die Zehnerstelle der Zahl darstellt und zeichen2 die Einerstelle. Um den ASCII Wert eines Zahlenzeichens in die entsprechende Zahl umzuwandeln, muss man den Index des Zahlenzeichens $'0' = 48$ vom Wert des Zahlenzeichens ab-

ziehen. Um den Zahlenwert der zweistelligen Zahl zu berechnen muss man den Zahlenwert der Zehnerstelle ($zeichen1 - 48$) mit 10 multiplizieren und zum Zahlenwert der Einerstelle ($zeichen2 - 48$) addieren. Das Ergebnis wird in der Variablen *result* vom Typ **unsigned int** gespeichert und am Ende der Funktion zurückgegeben.
Wiederum hätte man das Ganze natürlich etwas kürzer gestalten können:

```
1 unsigned int umwandelnInZahl(char zeichen1,char zeichen2)
2 {
3     return (zeichen1 - 48) * 10 + zeichen2 - 48;
4 }
```

Wir verzichten einfach darauf das Ergebnis in einer Variablen zwischenzuspeichern, sondern geben das Ergebnis der Berechnung direkt mit **return** zurück.
Zeile 42 - 57: Die *main* Funktion fordert den Benutzer zuerst auf, eine zweistellig Zahl einzugeben. Als nächstes Deklarieren wir eine Variable *zahl* welche ein Feld vom Typ **char** ist. Das Feld muss die Länge 3 besitzen, da eine Zeichenkette in C immer mit einer 0 abgeschlossen wird, d.h. wir müssen zwei Zahlenzeichen speichern und die abschließende Null. In der nächsten Zeile wird mit der *getline* Methode von *std::cin* die Zahlenzeichenkette in die Variable zahl eingelesen. Auch der *getline* Methode muss man als zweites Argument den Wert 3 übergeben, da sie weiß, dass es sich bei *zahl* um eine Zeichenkette handelt und die 0 anhängt. Nun wird die Funktion *istEineZahl* verwendet, um zu überprüfen ob es sich bei den beiden ersten Zeichen in *zahl* um Zahlenzeichen handelt. Wenn ja, dann benutzen wir die Funktion *umwandelnInZahl* um die beiden Zeichen in einen Zahlenwert umzuwandeln und den Zahlenwert auszugeben. Falls es sich jedoch nicht um eine gültige Zahlenzeichenkette handelt, wird eine Fehlermeldung ausgegeben.
Eine Sache haben wir in diesem Programm noch nicht berücksichtigt. Gibt der Benutzer eine Zahlenzeichenkette ein, die länger ist als zwei Zeichen, so werden nur die ersten zwei Zeichen in einen Zahlenwert umgerechnet und da ist in diesem Fall auch OK. Was passiert jedoch, wenn der Benutzer lediglich eine einstellige Zahl eingibt? In diesem Fall wäre das Programm der Meinung, es würde sich nicht um eine gültige Zahl handeln und eine entsprechende Meldung ausgeben. Dies ließe sich durch einige wenige Änderungen am Programm beheben. Versuchen Sie es doch einmal bevor Sie sich die Lösung zu Aufgabe 2 ansehen.
Es ist in diesem Fall auch möglich, das Zeichenkettenfeld den Funktionen als solches zu übergeben. Da wir aber noch nicht alle Details der Argumen-

tenübergabe an Funktionen behandelt haben, würde ich im Moment noch davon Abstand nehmen, komplexe Datentypen an eine Funktion zu übergeben.

Aufgabe 2:

Erweitern Sie das Programm aus Aufgabe 1 um eine Funktion "fakultaet", welche die Fakultät einer Zahl zwischen 1 und 20 berechnet und das Ergebnis als Wert vom Typ **double** zurückliefert.
Liegt die eingegebene Zahl außerhalb dieses Bereiches soll das Programm dem Benutzer eine entsprechende Fehlermeldung ausgeben.
Anstatt den eingegebenen Zahlenwert auszugeben wie in Aufgabe 1, soll das Programm nun das Ergebnis der Fakultätsberechnung ausgeben.

```
1  #include <iostream>
2
3
4  bool istEineZahl(char zeichen1,char zeichen2)
5  {
6    bool result=true;
7    if(zeichen1<48)
8    {
9      result=false;
10   }
11   else
12   {
13     if(zeichen1 > 57)
14     {
15       result=false;
16     }
17     else
18     {
19       if(zeichen2 != 0)
20       {
21         if(zeichen2 < 48)
22         {
23           result=false;
24         }
25         else
26         {
27           if(zeichen2 > 57)
28           {
29             result=false;
30           }
```

```
31          }
32        }
33      }
34    }
35    return result;
36  }
37
38  unsigned int umwandelnInZahl(char zeichen1,char zeichen2)
39  {
40     if(zeichen2==0)
41     {
42        zeichen2=zeichen1;
43        zeichen1='0';
44     }
45     unsigned int result=zeichen2 - 48;
46     result+=(zeichen1 - 48) * 10;
47     return result;
48  }
49
50  double fakultaet(unsigned int n)
51  {
52    double result=1.0;
53    unsigned int x;
54    for(x=2; x<=n; ++x)
55    {
56      result*=x;
57    }
58    return result;
59  }
60
61  int main()
62  {
63    std::cout << "Bitte geben Sie eine zweistellige Zahl ein:";
64    char zahl[3];
65    std::cin.getline(zahl,3);
66    if(istEineZahl(zahl[0],zahl[1]) == true)
67    {
68      unsigned int zahlenWert=umwandelnInZahl(zahl[0],zahl[1]);
69      if(zahlenWert==0)
70      {
71        std::cout << "Bitte geben Sie eine Zahl zwischen 1 und 20 ein.\n";
72      }
73      else if(zahlenWert > 20) // falscher Wert
74      {
75        std::cout << "Bitte geben Sie eine Zahl zwischen 1 und 20 ein.\n";
```

```
76      }
77      else
78      {
79        std::cout <<  "Fakultaet von " << zahlenWert;
80        std::cout << " ist " << fakultaet(zahlenWert) << ".\n";
81      }
82    }
83    else
84    {
85      std::cout << "Es handelt sich bei der Eingabe nicht um eine Zahl.\n";
86    }
87    return 0;
88  }
```

Das Programm ist dem Programm aus Aufgabe 1 sehr ähnlich, deshalb gehe ich hier nur noch auf die Unterschiede ein.

Zeile 4 - 36: Es handelt sich wieder um die Funktion, welche überprüft, ob zwei übergebene Variablen vom Typ **char** Zahlenzeichen sind. In **Zeile 19** gibt es jedoch einen kleinen Unterschied gegenüber der Lösung aus Aufgabe 1. Hier wird abgefragt, ob die Variable zeichen2 den Wert 0 enthält. Dies ist der Fall, wenn der Benutzer anstatt zwei oder mehr Zeichen einzugeben, nur ein Zeichen eingegeben hat und dann die RETURN Taste gedrückt hat. Auch in diesem Fall kann es sich um eine gültige Zahl handeln, sie besteht aber diesmal lediglich aus zeichen1. Konsequenterweise müssen wir zeichen2 nicht weiter überprüfen.

Zeile 38 - 48: Auch hier handelt es sich wieder um die Funktion aus Aufgabe 1, welche die beiden übergebenen Zeichen in einen Zahlenwert umwandeln. Nur berücksichtigt die Funktion nun auch, dass der Benutzer möglicherweise nur eine Ziffer eingegeben hat und nicht zwei. Hat *zeichen2* den Wert 0, besteht die zahl aus nur einer Ziffer, da sich die Einerstelle der Zahl aber nun in *zeichen1* befindet, müssen wir der Wert von *zeichen1* nach *zeichen2* kopieren und *zeichen1* auf die '0' oder 48 setzen, gerade so als hätte der Benutzer eine zweistellige Zahl mit führender 0 eingegeben.

Zeile 50 - 59: Die Funktion namens *fakultaet* übernimmt ein Argument vom Typ **unsigned int** und liefert einen **double** Wert zurück. Zu Beginn der Funktion wird eine Variable namens *result* vom Type **double** deklariert und mit dem Wert 1.0 initialisiert, anschließend wird in einer **for** Schleife die Fakultät der Zahl berechnet, die der Funktion als Argument übergeben wurde. In der Schleife wird eine Zahl vom Type **double** mit einer Zahl vom Type **unsigned int** multipliziert. Das Ergebnis, wiederum eine Zahl vom

Type **double**, wird der Variablen *result* zugewiesen. Dies funktioniert, da C++ gewisse Regeln festlegt nach denen Datentypen automatisch ineinander umgewandelt werden können. Wird eine Operation mit zwei verschiedenartigen Datentypen durchgeführt (z.B. Rechenoperationen), so hat das Ergebnis meist denselben Datentyp wie der Operator mit dem größeren Umfang.
Zeile 61 - 88: Auch die *main* Funktion ist der aus Aufgabe 1 sehr ähnlich. Es wird vom Benutzer die Eingabe einer zweistelligen Zahl erbeten, welche mit *getline* in die Variable *zahl* eingelesen und anschließend in den entsprechenden Zahlenwert umgewandelt wird.
Da wir nur Zahlen zwischen 1 und 20 für die Berechnung der Fakultät haben möchten, müssen wir den Zahlenwert daraufhin prüfen und gegebenenfalls eine Fehlermeldung ausgeben. Ist der Zahlenwert zwischen 1 und 20, berechnen wir mit der Funktion *fakultaet* die Fakultät der Zahl und geben das Ergebnis direkt aus (**Zeilen 79 + 80**), ohne es in einer Variablen zu speichern.
Aufmerksamen Lesern ist vermutlich die etwas andere Notation der verschachtelten if...else Anweisungen in **Zeile 66 bis 81**.
Anstatt der gewohnten Verschachtelung:

```
1  if(BEDINGUNG1)
2  {
3     ...
4  }
5  else
6  {
7     if(BEDIENGUNG2)
8     {
9        ...
10    }
11    else
12    {
13       ...
14    }
15 }
```

verwenden wir die etwas übersichtlichere Schreibweise:

```
1  if(BEDINGUNG1)
2  {
3     ...
4  }
5  else if(BEDIENGUNG2)
6  {
7     ...
8  }
```

```
else
{
  ...
}
```

Beide Schreibweisen sind absolut äquivalent, die zweite ist jedoch in der Regel etwas übersichtlicher, da nicht ganz so viele Klammern und Ebenen auftreten.

Aufgabe 3:

Schreiben Sie ein Programm, das ein zweidimensionales Feld mit dem kleinen Einmaleins füllt.
Das Programm soll anschließend die Quadrate, d.h. das Ergebnis der Rechnung 1x1, 2x2, 3x3 usw., aus der Feld auslesen und ausgeben.

```
#include <iostream>

int main()
{
  unsigned int einmaleins[10][10];
  unsigned int i;
  for(i=1; i<=10; ++i)
  {
    unsigned int j;
    for(j=1; j<=10; ++j)
    {
      einmaleins[i-1][j-1]=i*j;
    }
  }
  for(i=1; i<=10; ++i)
  {
    std::cout << "Das Quadrate von " << i;
    std::cout << " ist " << einmaleins[i-1][i-1] << ".\n";
  }
  return 0;
}
```

In der ursprünglichen, unkorrigierten Aufgabenstellung hieß es, man solle eine Funktion schreiben, welche ein Feld mit dem kleinen Einmaleins füllt. Je nachdem, wie man dies implementiert, kann es zu Problemen kommen, da wir noch nichts über den Gültigkeitsbereich von Variablen wissen. Aus diesem Grund haben wir die Aufgabenstellung so abgeändert, dass alles in der *main* Funktion erledigt werden soll.

Zeile 1 ist die übliche **#include** Direktive mit der die Deklarationen eingebunden werden, die man in C++ für die Ein- und Ausgabe von Text auf der Konsole (**cin**,**cout**) benötigt.
Zeile 5: Die *main* Funktion deklariert zu Anfang ein Feld mit der Dimension 2 und der Große 10 in jeder Dimension. Das ist Platz für 10*10 Werte vom Typ **unsigned int**, genauso die Zahl, die wir benötigen, um die Ergebnisse des kleinen Einmaleins zu speichern. Man kann ein zweidimensionales Feld auch einfach als Tabelle mit n Spalten und m Reihen betrachten, wobei m und n die Größen der Dimensionen des Feldes sind. D.h. unser Feld hier hat 10 Reihen mit je 10 Spalten.
Zeile 6 - 14: Nun müssen wir jede Zahl von 1 bis 10 mit jeder Zahl von 1 bis 10 multiplizieren. Dies erreicht man am besten mit einer verschachtelten **for** Schleife bei der die unterschiedlichen Schleifenzähler jeweils von 1 bis 10 laufen. Im Körper der inneren Schleife wird das Ergebnis der Multiplikation der beiden Schleifenzähler an einem der Plätze in der Tabelle (Feld) abgelegt. Der Schleifenzähler der äußeren Schleife (i) gibt dabei die Reihe in der Tabelle an, der Zähler der inneren Schleife (j) die Spalte. Da die Indices in C++ Felder bei 0 beginnen, muss man noch jeweils den Wert 1 von jedem Schleifenzähler abziehen, um die Indices in der Tabelle zu erhalten.
Zeile 15 - 21: Hier geben wir in einer weiteren **for** Schleife das Ergebnis der Quadrate aus. Die Quadrate liegen auf der Diagonalen der Tabelle, d.h. jeweils bei den Indices i,i. D.h. wir benötigen hier nur eine einfache Schleife und können den Schleifenzähler i von weiter oben wiederverwenden. Wiederum muss man darauf achten, dass das Ergebnis der Berechnung $1 * 1$ im Feld an Position $(0, 0)$ liegt, da die Indices bei 0 beginnen.
Es mag dem ein oder anderen aufgefallen sein, dass wir die Zählervariablen für die Schleifen jeweils vor der **for** Anweisung deklarieren und in der **for** Anweisung nur noch initialisieren. Dies tun wir lediglich aus Kompatibilitätsgründen, da einige ältere Microsoft Compiler die Deklaration in der **for** Anweisung nicht verarbeiten können. Aktuelle Compiler haben dieses Problem vermutlich nicht mehr.

Aufgabe 4

Schreiben Sie eine Funktion, welche als Argument ein **unsigned int** entgegennimmt und den Wert als Binärzahl (Dualsystem) ausgibt.
Wer nicht weiß, was Binärzahlen sind findet einen entsprechenden Artikel

unter http://de.wikipedia.org/wiki/Dualsystem.

```
1 #include <iostream>
2
3 unsigned int wertVonBit(unsigned int bitNummer)
4 {
5   unsigned int result=1;
6   unsigned int i;
7   for(i=1; i < bitNummer; ++i)
8   {
9     result*=2;
10  }
11  return result;
12 }
13
14 void binaerzahlAusgeben(unsigned int n)
15 {
16  unsigned int sizeInBits=(sizeof n) * 8;
17  unsigned int i;
18  for(i=sizeInBits; i>=1; --i)
19  {
20    unsigned int bitWert=wertVonBit(i);
21    if(n>=bitWert)
22    {
23      std::cout << "1";
24      n-=bitWert;
25    }
26    else
27    {
28      std::cout << "0";
29    }
30  }
31 }
32
33 int main()
34 {
35  std::cout << 0 << "   in Binaerschreibweise ist ";
36  binaerzahlAusgeben(0);
37  std::cout << "\n";
38  std::cout << 1 << "   in Binaerschreibweise ist ";
39  binaerzahlAusgeben(1);
40  std::cout << "\n";
41  std::cout << 2 << "   in Binaerschreibweise ist ";
42  binaerzahlAusgeben(2);
43  std::cout << "\n";
44  std::cout << 4 << "   in Binaerschreibweise ist ";
```

```
45   binaerzahlAusgeben(4);
46   std::cout << "\n";
47   std::cout << 8 << "   in Binaerschreibweise ist ";
48   binaerzahlAusgeben(8);
49   std::cout << "\n";
50   std::cout << 16 << "  in Binaerschreibweise ist ";
51   binaerzahlAusgeben(16);
52   std::cout << "\n";
53   std::cout << 32 << "  in Binaerschreibweise ist ";
54   binaerzahlAusgeben(32);
55   std::cout << "\n";
56   std::cout << 64 << "  in Binaerschreibweise ist ";
57   binaerzahlAusgeben(64);
58   std::cout << "\n";
59   std::cout << 128 << " in Binaerschreibweise ist ";
60   binaerzahlAusgeben(128);
61   std::cout << "\n";
62   std::cout << 256 << " in Binaerschreibweise ist ";
63   binaerzahlAusgeben(256);
64   std::cout << "\n";
65   std::cout << 512 << " in Binaerschreibweise ist ";
66   binaerzahlAusgeben(512);
67   std::cout << "\n";
68   std::cout << 48 << "  in Binaerschreibweise ist ";
69   binaerzahlAusgeben(48);
70   std::cout << "\n";
71   std::cout << 255 << " in Binaerschreibweise ist ";
72   binaerzahlAusgeben(255);
73   std::cout << "\n";
74   std::cout << 62 << "  in Binaerschreibweise ist ";
75   binaerzahlAusgeben(62);
76   std::cout << "\n";
77   std::cout << 21 << "  in Binaerschreibweise ist ";
78   binaerzahlAusgeben(21);
79   std::cout << "\n";
80   return 0;
81 }
```

Zeile 1 ist die übliche **#include** Direktive mit der die Deklarationen eingebunden werden, die man in C++ für die Ein- und Ausgabe von Text auf der Konsole (**cin**,**cout**) benötigt.
Zeile 3 - 12: Die Funktion *wertVonBit* nimmt ein Argument *bitNummer* vom Typ **unsigned int** und gibt ebenfalls einen **unsigned int** Wert zurück. Die Funktion berechnet den Wert von $2^{b}itNummer$. Dies wird dadurch erreicht,

dass man die Potenz durch mehrfache Multiplikation in einer **for** Schleife nachbildet.
Zeile 15 - 32: Die Funktion *binaerzahlAusgeben* ist das eigentliche Herzstück des Programms. Sie übernimmt als Argument eine Variable n vom Typ unsigned int. Sie besitzt keinen Rückgabewert. Als erstes bestimmt die Funktion wie viele Bit eine Variable vom Typ **unsigned int** besitzt. Dies ist notwendig, da der Datentyp **unsigned int** je nach System und Compiler unterschiedlich groß sein kann. Das Ergebnis des **sizeof** Operators muss man dann noch mit 8 Multiplizieren um die Anzahl der Byte in die zahl der Bit umzurechnen. Die Variable *sizeinBits* speichert die Größe einer **unsigned int** Variablen in Bit und somit wieviel Bit die Funktion ausgeben muss. In der anschließenden **for** Schleife wird der Schleifenzähler i mit dem Wert von sizeInBits initialisiert und am Ende jedes Schleifendurchgangs um eins erniedrigt. So lange i größer oder gleich 1 ist. Der Schleifenzähler i ist gleichzeitig der Index des nächsten Bit, das ausgegeben werden soll. D.h. wir geben die Bits vom höchstwertigen Bit zum niedrigwertigsten Bit, also von links nach rechts, aus.
Um festzustellen, ob wir für ein Bit eine 1 (gesetzt), oder ein 0 (ungesetzt) ausgeben müssen, berechnen wir den Zahlenwert den das Bit repräsentiert mit der Funktion *wertVonBit* (**Zeile 21**) und weisen das Ergebnis der Variablen *bitWert*. Ist der Wert von n nun größer als der Wert der Variablen *bitWert*, so ist das Bit gesetzt und wir geben eine "1" aus (**Zeile 24**), anschließend müssen wir noch den Wert der Variablen *bitWert* von n abziehen, damit der Vergleich beim nächsten Schleifendurchgang überhaupt funktionieren kann. Tun wir dies nicht, wäre der Vergleich in jedem Schleifendurchgang nachdem man ein gesetztes Bit gefunden hatte wahr, und das Programm würde nur noch Einsen ausgeben. Ist der Vergleich nicht wahr, geben wir eine 0 aus, da das Bit nicht gesetzt ist.
Zeile 34 - 82: Ist die *main* Funktion welche einige **unsigned int** Werte mithilfe der Funktion *binaerzahlAusgeben* ausgibt.
Beim Schreiben des Programms muss man darauf achten, dass die Funktion *wertVonBit* deklariert werden muss, bevor man diese in der Funktion *binaerzahlAusgeben* verwenden kann. Dasselbe gilt für die Funktion *binaerzahlAusgeben*, diese muss deklariert werden, bevor man sie in der *main* Funktion verwenden kann.
Will man z.B. die *main* Funktion am Anfang des Programms haben, so muss man die Deklaration der *binaerzahlAusgeben* Funktion vor die *main* Funktion stellen, die Deklaration der *binaerzahlAusgeben* kann im Anschluss an die *main* Funktion erfolgen. Entsprechendes gilt für die Deklaration der *wertVon-*

Bit Funktion in Relation zur Definition der *binaerzahlAusgeben* Funktion.

Aufgabe 5

Gegeben ist folgender Programmschnipsel:

```
1 int x=4;
2 int y=(++++x)+++(++++x)++;
```

Welche Werte haben x und y nachdem die beiden Programmzeilen ausgeführt wurden und warum?
Versuchen Sie die Frage zu beantworten ohne den Compiler zu bemühen.
Anschließend bauen Sie die beiden Zeilen in ein Programm ein, daß die Werte von x und y am Ende ausgibt, und überprüfen ob Ihre Annahme richtig war. Falls nicht, versuchen Sie anhand der Ergebnisse herauszufinden warum das Ergebnis so aussieht.
Die Aufgabe ist sehr schwierig, es wäre trotzdem gut, wenn man zumindest versuchen würde Sie zu lösen.
(Bitte schreiben Sie niemals solchen Code und wenn es doch einmal sein muss, um z.B. aus einem Programm das letzte Quäntchen Geschwindigkeit herauszuholen, dokumentieren Sie den Code ausreichend.)
Zeile 1: Sollte klar sein, es wir eine Variable x vom Typ **int** deklariert und mit dem Wert 4 initialisiert.
Zeile 2: Es handelt sich hier um mehrere Operationen. Am besten versucht man die einzelnen Operationen zu identifizieren und auf mehrere Zeilen aufzuteilen.
Wie wir wissen, hat der Klammeroperator die höchste Priorität, d.h. die beiden Ausdrücke in runden Klammern werden zuerst ausgewertet. Es handelt sich dabei um zwei Increment Operatoren in der Präfix Notation. D.h. x wird jeweils zwei mal um eins erhöht.

```
1 int x=4;
2 ++x;
3 ++x;
4 ++x;
5 ++x;
6 int y=(x)+++(x)++;
```

Zeile 6: Die Ausdrücke in den runden Klammern sind nun ausgewertet und wir könnten die Klammern auch weglassen. Wenn wir die Zeile jetzt noch etwas auseinanderziehen erhalten wir folgendes:

```
1 int x=4;
2 ++x;
3 ++x;
4 ++x;
5 ++x;
6 int y=x++ + x++;
```

Zeile 6: Man sieht nun, dass es sich um zwei weitere Increment Operatoren und eine Addition handelt. Diesmal sind es jedoch die Increment Operatoren in der Postfix Notation, d.h. sie haben einen niedrigere Priorität als die Addition. Es wird also zuerst die Addition der beiden Werte von x vorgenommen und erst dann wird x erneut zwei mal um eins erhöht:

```
1 int x=4;          // x=4
2 ++x;              // x=5
3 ++x;              // x=6
4 ++x;              // x=7
5 ++x;              // x=8
6 int y=x + x;      // y=16
7 x++;              // x=9
8 x++;              // x=10
```

x hat somit anschließend den Wert 10 und y den Wert 16.