Einführung in C++

Arne Schmitt

15. Juli 2006

Inhaltsverzeichnis

Teil I.
Die erste Stunde

Die erste Stunde begint traditionell mit dem „Hello World“-Programm.

```
1 #include <iostream>
2 #include <cstdlib>
3
4 using namespace std;
5
6 int main(int argc, char *argv[])
7 {
8   cout << "Hello, world!" << endl;
9   return EXIT_SUCCESS;
10 }
```

Durch Ausführen dieses Programms erhalten wir:

```
Hello, world!
```

Keine große Überraschung.
Jetzt einmal die einzelnen Elemente des Programms

#include <...> Die Include Anweisung erklärte dem Compiler welche Bibliotheken wir verwenden wollen.

using namespace std; Wählt den Aktuellen Namensraum. Wir ignorien erst einmal was das genau tut.

Jetzt wird es wichtig! Wir kommen zu main-Funktion. Der Startpunkt aller C++ Programme.

```
1 int main(int argc, char *argv[])
2 {
3   ...
4   return EXIT_SUCCESS;
5 }
```

Dieser Teil kommt so in nahezu jedem C++ Programm vor. Zwischen den {} stehen die eigentlichen Befehle. Und für den Anfang werden alle unsere Änderungen hier stattfinden. Den Rest des Programms lassen wir wie er ist.
Jetzt zu unserem ersten Befehl:

```
1 cout << "Hello, world!" << endl;
```

Wie wir bereits wissen gibt er dern Text „Hello, world!“auf dem Bildschirm aus. Texte werden in C++ immmer in ““eingeschloßen, damit der Compiler weiß, das er das was dort steht nicht verstehen muß.

Endl Ist ein vordefinierter Text, ein Steuerzeichen, das dem Computer befiehlt eine neue Zeile anzufangen.

cout Ist der „standart output stream“. Eine hochtrabende Bezeichnung, sie besagt aber nur cout immer gebrauct wrd wenn wir etwas ausgeben wollen.Um jetzt tatsächlich etwas auszugeben, schieben wir unsere Werte nach cout.

Das schreibt sich dann so:

```
1 cout << Wert;
```

oder wenn wir mehrere Werte ausgeben wollen:

```
1 cout << Wert1 << Wert2 << Wert3 ...;
```

Werte können dabei nicht nur Texte sein, sondern genausogut auch Zahlen(int,double),einzelen Buchstaben(char),Ja/Nein-Aussagen(bool) Als Gegenstück zu cout gibt es cin, den “standart input stream„. Ihn verwenden wir wenn wir eine Eingabe vom Benutzer haben wollen. Ähnlich wie bei cout schieben wir die Werte aus cin in eine Variable.

```
1  cout << "Anzahl:";
2  int z;
3  cin >> z;
```

Das bringt uns zu unserem ersten Grundlagenkapitel

1. Variablen

Variablen in einer Programmiersprache sind keine Unbekanten wie in der Mathematik! Eine Variable ist ein reservierter Speicherplatz an dem sich der Computer eine ganz bestimmte Art von Information notieren, man spricht auch von ablegen, kann. Diese bestimmte Art von Information nennt man den Typ eine Variablen. Die Basistypen die wir im Moment verwenden sind:

bool einen Ja/Nein (true/false) information. 1 bit.

int eine ganze Zahl. 32 bit

char ein einzelnes Zeichen (Buchstabe,Ziffer,Sonderzeichen,...) 8 bit

double eine Kommazahl , 128 bit

Um eine Variable zu verwenden muß man sie ersteinmal Deklarieren, also dem Compiler sagen was für einen Typ von Variable wir wollen und wie sie heißen soll.

```
1 Typ Name;
```

Dazu ein paar Beispiele:

```
1 bool ende;
2 double preis;
3 char buchstabe;
4 int zahl;
```

Jetzt müßen wir die Variable initialisieren. Also einen Wert festlegen, den bisher könnte die Variable noch jeden möglichen Wert enthalten und wir haben keine Ahnung welchen. Wenn wir eine Benutzereingabe speichern wollen, können wir uns das aber auch sparen.

```
Name = Wert;
```

Dazu wieder ein paar Beispiele:

```
ende=true;
preis=42.6;
buchstabe='P';
zahl=-5;
```

1.1. Beispiele

Jetzt sind wir soweit ein paar einfache Programme zu schreiben.

1.Kreisumfang berechnen

```
double r;
cout << "Radius:" ;
cin >> r;
cout << "Umfang:" << r*3.1415 << endl;
```

2.Zahl der Tage/Stunden/Sekunden in x Jahren berechnen (grob)

```
int jahre;
cout << "Jahre:";
cin >> jahre;
cout >> "in " << jahre << " Jahren sind ca. " \
        << jahre *365.25 << " Tage vergangen" << endl;
cout >> "in " << jahre << " Jahren sind ca. " \
        << jahre *365.25 * 24 << " Stunden vergangen" << endl;
cout >> "in " << jahre << " Jahren sind ca. " \
        << jahre *365.25 * 24 * 3600 << " Sekunden vergangen" << endl;
```

2. Schleifen

Es gibt drei Grundtypen von Schleifen:

1. for
2. while
3. do while

2.1. Die For-Schleife

Die For-Schleife ist Ideal um zu zählen oder eine bekannte Zahl von Wiederholungen zu machen.

2.2. Die While-Schleife

Die While-Schleife wird verwendet wenn man keine Ahnung hat wieviele Wiederholungen es geben wird.

2.3. Die Do-While-Schleife

Die Do-While-Schleife ist der While-Schleife sehr ähnlich. Der Hauptunterschied ist, daß sie mindestens einmal durchläuft bevor sie die Bedingung überprüft.

2.4. Beispiele

1.Zählen von 1 bis z

```
1  int z;
2  cin >> z;
3  for(int i=0;i<z;i++)
4  {
5    cout << i+1 << ",";
6  }
7  cout << endl;
```

2.Zählen der Stellen in einer ganzen Zahl (int)

```
1  int z;
2
3  cout << "Stellenzählen" << endl;
4  cout << "Zahl:";
5  cin >> z;
6
7  int stellen=0;
8  while(z!=0)
9  {
10  stellen++;
11  z = z /10; // Zahl um eine Stelle verschieben
12 }
13 cout << "Es sind " << stellen << " Stellen" << endl;
```

3.Kommazahl aufsummieren bis der Benutzer 0 eingibt

```
1   cout << "Summe" << endl;
2   double summe=0.0,eingabe;
3   do
4   {
5    cin >> eingabe;
6    summe += eingabe;
7   }
8   while(eingabe!=0);
9
10  cout << "Summe: " << summe << endl;
```

Teil II.
Die Zweite Stunde

3. Verzweigungen

Verzweigungen dienen dazu, um je nach situation unterschiedliche Befehle auszuführen.

3.1. If

Die einfachste Verzweigung ist if. Der prinzipielle Aufbau der if-Anweisung lautet:

```
If(Bedingung) Ja-Befehl else Nein-Befehl
```

Bedingung ist irgendetwas, das true oder false liefert. Der Computer interpretiert dabei 0 als false und alles andere als true;
Beispiele:

```
if( x == 1)
{
        cout << "X ist eins\n";
}
else
{
        cout << "X ist nicht eins.\n";
}
if(true) cout << "Hallo\n";
```

3.2. switch-case

Mehrfachverzweigung

```
Switch(x)
{
case 1:
        cout << "Eine Möglichkeit";
        break;
case 2:
        cout << "Eine Möglichkeit";
        break;
case 2:
        cout << "Eine Möglichkeit";
        break;
case 8:
        cout << "Eine Möglichkeit";
```

```
14          break;
15 default:
16          cout << "Irgendwas anderes\n";
17 }
```

Beispiele

4. Arrays

```
1 int zahlen[10];
2 int zahlen2[]={4,8,63,476,3,24,45};
3 zahlen[0]=1;
4 zahlen[5]=34;
5 cout << zahlen[0] << zahlen[1] << zahlen[5];
6 for(int i=0;i<10;i++) zahlen[i]=5*i+3;
7 for(int i=0;i<10;i++) cout << i << '\t' << zahlen[i] << endl;
8 for(int i=0;i<6;i++) cout << i << '\t' << zahlen2[i] << endl;
```

4.1. Beispiele

5. Pointer

Pointer(Zeiger) enthalten die Speicheradresse einer Information.

```
1 Typ *name;
```

Ist also ein Zeiger auf eine Information vom Typ Typ mit dem Namen name Um an den Wert zu kommen auf den der Pointer zeigt muß an in dereferenzieren:

```
1 cout << *name; // Schreibt den Wert auf den name Zeigt
2 cout << name; // Schreibt die Addresse an der die Information steht
```

5.1. Beispiele

```
1 int zahl=2;
2 int zahl2= 15;
3 int zahlen[5]={3,5,2,4,0};
4 int *zeiger=&zahl;
5 cout << zeiger << " " << *zeiger << "\n";
6 *zeiger = 5;
7 cout << zeiger << " " << *zeiger << "\n";
8 zeiger = &zahl2;
9 cout << zeiger << " " << *zeiger << "\n";
10 zeiger = zahlen;
11 while(*zeiger)
```

```
12 {
13         cout << zeiger << "␣" << *zeiger << "\n";
14         zeiger++;
15 }
```

Teil III.
Die dritte Stunde

Heute beschäftigen wir uns mit einem praktischen Beispiel, um das Zusammenspiel der Grundelemente zu demonstrieren.

6. Die Notenverwaltung v1

6.1. Anforderungen

Unser Programm soll die folgenden Funktionen beherschen:

1. Bis zu 10 Schulnoten (1-6) speichern
2. Eingabe von Noten
3. Ausgabe der Noten
4. Den Durchschnitt der Noten berechnen
5. Das Programm beenden

6.2. Der Entwurf - Analyse

Ein guter Ansatzpunkt für den Entwurf eines Programms ist es, sich erst einmal zu überlegen, welche Art Daten man verarbeiten will. In unserem Beispiel sind das Schulnoten. Dafür gibt es verschiedene Darstellungsarten.

1. 1,2,3,4,5,6
2. 1,1-,1-2,2+,2,....
3. 1,1.25,1.5,....
4. A,B,C,D,E,F

Die einfachste Möglichkeit scheit die 3. zu sein, den wir haben ja bereits etwas das Kommazahlen speichern kann. Der Typ double. Davon brauchen wir jetzt 10 Stück. Hier bietet sich ein Area an. Jetzt müssen wir uns allerdings auch noch merken, wo die nächste Note hin soll. Sonst würden wir die gespeichete Note ja jedesmal ersetzen. Da wir die Noten in einem Area speichern, kann man die position als ganze Zahl angeben, also als int.

```
1 double Noten[10];
2 int position=0;
```

Als nächstes sollte man einen Blick auf den Ablauf des Programms werfen. In unserem Fall kennen wir den Ablauf nicht. Der Benutzer muß entscheiden was er als nächstes tun will. In so einem Fall verwendet man gerne ein Menu. D.H, man presentiert dem Nutzer die Liste der möglichen Funktionen und läst ihn auswählen.

```
cout << "\n\tMenu\n";
cout << "1: Eingeben\n";
cout << "2: Ausgeben\n";
cout << "3: Durchschnitt\n";
cout << "0: Beenden\n";
cout << ":";
int eingabe;
cin >> eingabe;
switch(eingabe)
{
        ...
}
```

Und jetzt zu den einzelnen Funktionen:

Eingabe einer Note
Als erstes sollten wir den Nutzer darüber informieren was wir von im erwarten.

```
cout << "Bitte geben sie eine Note (1-6) ein:";
```

Anschließend müssen wir die Eingabe einlesen und prüfen ob es wirklich eine Note zwischen 1 und 6 ist. Außerdem sollten wir kontrollieren ob wir noch Platz für eine weitere Note haben. Jetzt können wir die Note am richtigen Platz abspeichern und sind damit fertig.

```
double note;
cin >> note;
if(position>=10) break; // Leider kein Platz mehr
if(note > 6 || note < 1  )
  cout << "Falsche Eingabe\n";
else
{
  Noten[position] = note;
  position += 1;
}
```

Ausgabe der Noten
Das Ausgeben besteht eigentlich nur aus einer Schleife um alle Note aufzuzählen.

```
cout << "\n\tNoten:\n";
for(int i=0; i<position;i++)
  cout << i+1 << ": " << Noten[i] << "\n";
```

Durchschnitt berechnen
d Durchschnitt

N_i Die $i$te Note

n Die Anzahl der Noten

$$d = \frac{\sum_i N_i}{n}$$

in C++:

```
1 double sum=0.0; // Irgendwo müssen wir uns da die Summe merken
2 for(int i=0; i<position;i++)
3   sum += Noten[i];
4 cout << "\tDurchschnitt:" << sum / position << endl;
```

6.3. Der Quelltext

Und hier sind jetzt alle Teile zu einem funktionirenden Programm zusammengesetzt.

```
1 #include <iostream>
2 #include <cstdlib>
3
4 using namespace std;
5
6 int main(int argc, char *argv[])
7 {
8   cout << "Notenverwaltung" << endl;
9   bool beenden=false;
10  double Noten[10];
11  int position=0;
12
13  do
14  {
15          cout << "\n\tMenu\n";
16          cout << "1: Eingeben\n";
17          cout << "2: Ausgeben\n";
18          cout << "3: Durchschnitt\n";
19          cout << "0: Beenden\n";
20          cout << ":";
21          int eingabe;
22          cin >> eingabe;
23          double sum=0.0;
24          switch(eingabe)
25          {
26                  case 1:
```

```
27                              // Eingeben
28                              cout << "Bitte Note eingeben:";
29                              double note;
30                              cin >> note;
31                              if(position>=10) break;
32                              if(note > 6 || note < 1   )
33                                      cout << "Falsche Eingabe\n";
34                              else
35                              {
36                                      Noten[position] = note;
37                                      position += 1;
38                              }
39                              break;
40                      case 2:
41                              // Ausgeben
42                              cout << "\n\tNoten:\n";
43                              for(int i=0; i<position;i++)
44                                      cout << i+1 << ": " << Noten[i] << "\n";
45                              break;
46                      case 3:
47                              // Durchschnitt
48
49                              for(int i=0; i<position;i++)
50                                      sum += Noten[i];
51                              cout << "\tDurchschnitt:" << sum / position << endl;
52                              break;
53                      case 0:
54                              beenden = true;
55                              break;
56                      default:
57                              cout << "unbekannte Eingabe\n";
58              }
59      }
60      while(!beenden);
61
62      return EXIT_SUCCESS;
63  }
```

6.4. Erweiterungen

Zum üben bieten sich jetzt einige Erweitungen an.

1. Weitere Statistikfunktionen (z.B. Standartabweichung) hinzufügen
2. Noten löschen (Alle auf einmal)

3. Noten löschen (einzeln)
4. Mit diesem Programm als Vorlage, ein Telefonbuch schreiben.

Teil IV.
Die vierte Stunde

Wie wir gesehen haben, wird es schnell unübersichtlich wenn man alles ineinander schachtelt. Außerdem wäre es unheimlich nützlich wenn der Computer sich die eingegebenen Daten auch zwischen den Programmstarts merken könnte.

7. Funktionen

Als Antwort auf das erste Problem, wurden Funktionen eingeführt. Funktionen sind mehr oder weniger unabhänige Programmteile, die bei Bedarf ausgeführt (aufgerufen) werden.

Eine Funktion findet man in jedem Programm. Die Main-Funktion.

```
1 int main(int argc, char *argv[])
2 {
3         ...
4   return EXIT_SUCCESS;
5 }
```

Wir können aber auch eigene Funktionen bauen. Z.B. eine Durchschnitt-Funktion

```
1 double durchschnitt()
2 {
3         double sum=0.0;
4         for(int i=0; i<position;i++)
5           sum += Noten[i];
6         return sum / position;
7 }
8
9 cout << "Durchschnitt:" << durchschnitt() << endl;
```

7.1. Die Notenverwaltung v2

Hier ist unsere Notenverwaltung jetzt auf mehrere Funktionen aufgeteilt.

```
1 #include <iostream>
2 #include <cstdlib>
3
4 using namespace std;
5
6 // Damit unsere Funktionen die Noten sehen können,
7 // müssen wir sie hier oberhalb der Funktionen declarieren
```

```
8  double Noten[10];
9  int position=0;
10
11 void NoteEingeben()
12 {
13         // Eingeben
14         cout << "Bitte Note eingeben:";
15         double note;
16         cin >> note;
17         if(position>=10)
18         {
19                 cout << "Kein Platz mehr\n";
20                 return;
21         }
22         if(note > 6 || note < 1  )
23                 cout << "Falsche Eingabe\n";
24         else
25         {
26                 Noten[position] = note;
27                 position += 1;
28         }
29 }
30
31 void NotenAusgeben()
32 {
33         // Ausgeben
34         cout << "\n\tNoten:\n";
35         for(int i=0; i<position;i++)
36                 cout << i+1 << ": " << Noten[i] << "\n";
37 }
38
39 double durchschnitt()
40 {
41         // Durchschnitt
42         double sum=0.0;
43         for(int i=0; i<position;i++)
44                  sum += Noten[i];
45         return sum/position;
46 }
47
48 void DurchschnittAusgeben()
49 {
50         cout << "\tDurchschnitt:" << durchschnitt() << endl;
51 }
52
```

```
53 int main(int argc, char *argv[])
54 {
55   cout << "Notenverwaltung" << endl;
56   bool beenden=false;
57
58   do // Hauptschleife
59   {
60           cout << "\n\tMenu\n";
61           cout << "1: Eingeben\n";
62           cout << "2: Ausgeben\n";
63           cout << "3: Durchschnitt\n";
64           cout << "0: beenden\n";
65           cout << ":";
66           int eingabe;
67           cin >> eingabe;
68           double sum=0.0;
69           switch(eingabe)
70           {
71                   case 1:
72                           NoteEingeben();
73                           break;
74                   case 2:
75                           NotenAusgeben();
76                           break;
77                   case 3:
78                           DurchschnittAusgeben();
79                           break;
80                   case 0:
81                           beenden = true;
82                           break;
83                   default:
84                           cout << "unbekannte Eingabe\n";
85           }
86   }
87   while(!beenden);
88
89   return EXIT_SUCCESS;
90 }
```

8. Dateien

Die Lösung für das zweite Problem, das Informationen mit den des Progamms verloren sind, heißt Dateien. In C++ werden Dateien meißt als streams behandelt. Also also Datenströme aus denen man nacheinder Daten lesen oder in die man nacheinander

Daten schreiben kann.

Mit zwei Streams hatten wir bereits zutun:

cin Der Inputstrean. Meist die Tastatureingaben des Benutzers.

cout Der Outputstream. Der Inhalt erscheint normalerweiße auf dem Bildschrim.

Wir können num eigene Streams definieren, die ihre Daten z.B. auf Festplatte,Diskette oder USB-Stick speichern und von dort auch wieder lesen.

Um Dateien verwenden zu können müssen wir eine neue Includezeile hinzufügen.

```
#include <fstream>
```

jetzt können wir ein Datei zum Schreiben

```
ofstream output("Testdatei");
output << "Das hier steht gleich auf der Festplatte" << endln;
output.close();
```

oder auch zum lesen öffnen.

```
ifstream input("Testdatei");
string s;
input >> s;
input.close();
cout << s;
```

8.1. Die Notenverwaltung v3

Damit bekommt unsere Notenverwaltung ein Gedächnis:

```
void Schreiben(char* filename)
{
        ofstream out(filename);
        out << "#Notenverwaltung_v3\n";
        for(int i=0;i<position;i++)
                out << Noten[i] << "\n";
        out.close();
}

void Lesen(char* filename)
{
        ifstream in(filename);
        string s;
        in >> s;
        position =0;
        while(!in.eof() && position<10)
        {
                double note;
```

```
19                  in >> note;
20                  Noten[position] = note;
21                  position++;
22          }
23  }
```

9. Strukturen und Klassen

Wenn wir schon dabei sind, unser Programm übersichtlicher zu machen, führen wir gleich noch zwei weitere Konzepte ein.

9.1. struct

Eine C++ Struktur (struct) fast zusammengehörende Daten zusammen.

```
1  struct _notenbuch
2  {
3   double Noten[10];
4   int position;
5  };
6
7  _notenbuch buch;
8  buch.position=0;
9  buch.Noten[2]=1.5;
```

9.2. Dynamische Speicherverwaltung: new / delete

Im folgenden werden wir häufinger dynamisch Speicher anfordern und abgeben müßen. Dynamisch steht in diesem Fall für, während das Programm läuft. Im Gegensatz dazu waren alle Variablen die wir bisher verwendet haben dem Computer schon beim compileren bekannt und der konnte schon vor dem Start des Programms genau festlegen was wo im Speicher liegt.

Variablen die wir über dynamische Speicherwaltung erhalten, haben keine Namen. Wir gekommen nur die Zeiger auf ihre Speicherblocke.

Um also dynamischen Speicher anzufordern verwendet man den Operator new und um ihn freizugeben delete;

Beispiele für new und delete

```
1  int main()
2  {
3  int *a = new int; // a zeigt auf speicherplatz für einen int
4  int *ar = new int[20]; // ar zeigt auf specherplatz fuer 20 int;
5
6  *a=12;
```

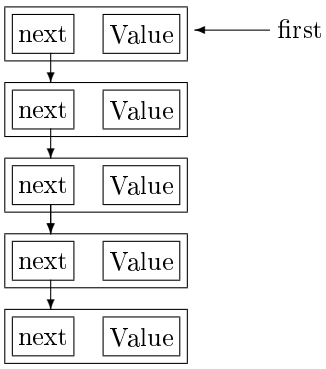
```
7  ar[2] = 12;
8  ar[0] = 2;
9  cout << ''ar[0]=`` << ar[0] << endl;
10 cout << ''*array=`` << *ar << endl;
11 cout << ''ar[2]=`` << ar[2] << endl;
12 cout << ''*(ar+2)=`` << *(ar+2) << endl;
13
14 cout << a << '' zeigt auf `` << *a << endl;
15 delete a;
16 a = new int;
17 cout << a << '' zeigt auf `` << *a << endl;
18
19 /* Wenn wir den Speicher ,d.h. die Variablen, nicht mehr brauchen
20 sollten wir ihn freigeben */
21 delete a;
22 delete[] ar;
23 /* Wenn man einen Dynamischen Wert mit delete freigegeben hat,
24 sollte man immer alle Pointer die darauf zeigen auf 0 setzten */
25 a=0;
26 ar=0;
27 }
```

9.3. Verkettete Liste

Bisher haben wir zum Speichern der Informationen Arrays verwendet. Bei einem Array muß man aber vorher wissen, wieviele Elemente es Maximal werden sollen. Oft kann man das aber beim Schreiben des Programms nicht feststellen. Eine der gängigen Lösungen für dieses Problem, sind Verkette Listen.

9.3.1. Die einfach verkettete Liste

Bei einem Array fordern wir einen großen zusammenhängenden Speicherblock an, in den alle Werte hineinpassen. Bei einer einfach verketteten Liste fordern wir für jeden Wert eine kleinen Speicherblock an. Dafür speichern wir für jeden Wert , wo der nächste Wert zu finden ist, oder 0 wenn es der letzte Wert ist.



struct Entry { Entry *next; int Value; };

Entry *first;

Beispiel einer einfach verketteten Liste

```
/*************************************************************
** Beispiel fuer die Umsetzung einer einfach verketteten Liste *
*************************************************************/

/* Unsere Datenstruktur */
struct Entry
{
  Entry *next; // Zeiger auf den naechsten Eintrag;
  int Value; // Beispielwert
};

Entry *first=0; // Zeiger auf den ersten Eintrag;

/* Neuen Wet am Anfang einfuegen */
Entry *insertValueFirst(int v)
{
  Entry *neu = new Entry();
  neu->Value=v;
  neu->next=first;
  first = neu;
}

/* Neuen Wert am Ende anhaengen */
Entry *appendValue(int v)
{
  Entry *neu = new Entry();
  neu->Value = v;
  neu->next = 0;
  if(first)
  {
    Entry *i=0;
```

```
32      for(i=first;i->next; i=i->next) ;
33      i->next = neu;
34    }
35    else
36     first = neu;
37    return neu;
38  }
39
40  /* Eintrag mit bestimmten Wert finden */
41  Entry *findValue(int v)
42  {
43    for(Entry *i=first;i;i=i->next)
44    {
45      if(i->value==v) return i;
46    }
47  }
48
49  /* Eintrag entfernen */
50  void removeEntry(Entry *e)
51  {
52    if(e==first) first = e->next;
53    else
54    {
55      Entry *i=0;
56      for(i=first;i && i->next!=e;i=i->next) ;
57      if(!i) return; else i->next = e->next;
58    }
59    delete e;
60  }
```

Vorteile der einfach verketteten Liste:

- einfach zu verstehen
- sehr geringer Overhead, wir brauchen pro Wert nur einen Zeiger zusätzlich. (In unserem Beispiel verdoppelt sich dadurch allerdings der Speicherverbrauch!)
- einfach der Reihe nach auszulesen oder zu durchsuchen
- man kann sehr schnell den ersten Wert auslesen
- man kann sehr schnell den nächsten Wert auslesen
- man kann sehr schnell einen Wert am Anfang einfügen.
- man kann sehr schnell den ersten Wert entfernen.

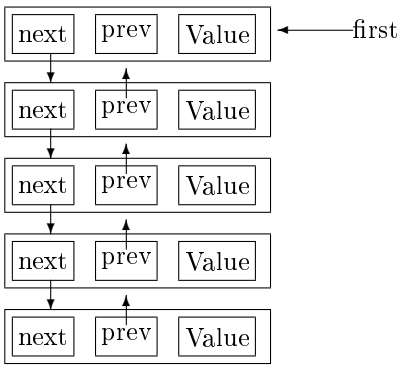
Nachteile der einfach verketteten Liste:

- es ist teuer (an rechenzeit) den n-ten Wert auszulesen
- es ist teuer die Werte in irgend einer anderen Reihenfolge als vom Anfang zu Ende anzuarbeiten.
- es ist im verhältniss besonder aufwänding den Vorgägner eines Eintrags zu finden.
- es ist teuer einen Wert am Ende anzuhängen.
- es ist teuer und aufwänding einen Wert an beliebier Stelle einzufügen.
- es ist teuer und aufwänding einen Wert an beliebier Stelle zu entfernen.
- es ist teuer und aufwänding einen Wert an beliebie Stelle zu verschieben.

Man kann das Einfügen und Auslesen am Ende stark vereinfachen und beschleunigen in dem man zusetztlich einen Zeiger auf den letzten Eintrag verwaltet.

9.3.2. Die doppelt verkettete Liste

Um einige der Nachteile der einfachverketteten Liste zu beheben, kann man zusätzlich zum nächsten auch den vorherigen Eintrag speichern. Damit verbraucht man zwar mehr Speicher. Aber fast alle Operationen innerhalb der Liste werden einfacher und schneller.



struct Entry { Entry *next,*prev; int Value; };

Entry *first;

Beispiel einer dopppelt verketteten Liste

```
1 /*************************************************************
2 ** Beispiel fuer die Umsetzung einer doppelt verketteten Liste *
3 *************************************************************/
4
5 /* Unsere Datenstruktur */
6 struct Entry
7 {
8   Entry *next; // Zeiger auf den naechsten Eintrag;
```

```
9   Entry *prev; // Zeiger auf den vorherigen Eintrag;
10  int Value; // Beispielwert
11 };
12
13 Entry *first=0; // Zeiger auf den ersten Eintrag;
14
15 /* Neuen Wet am Anfang einfuegen */
16 Entry *insertValueFirst(int v)
17 {
18   Entry *neu = new Entry();
19   neu->Value=v;
20   neu->next=first;
21   neu->prev=0;
22   first = neu;
23 }
24
25 /* Neuen Wert am Ende anhaengen */
26 Entry *appendValue(int v)
27 {
28   Entry *neu = new Entry();
29   neu->Value = v;
30   neu->next = 0;
31   neu->prev = 0;
32   if(first)
33   {
34     Entry *i=0;
35     for(i=first;i->next; i=i->next) ;
36     i->next = neu;
37     neu->prev = i;
38   }
39   else
40    first = neu;
41   return neu;
42 }
43
44 /* Eintrag mit bestimmten Wert finden */
45 Entry *findValue(int v)
46 {
47   for(Entry *i=first;i;i=i->next)
48   {
49     if(i->value==v) return i;
50   }
51 }
52
53 /* Eintrag entfernen */
```

```
54 void removeEntry(Entry *e)
55 {
56   if(e==first)
57   {
58         first = e->next;
59         first->prev = 0;
60   }
61   else
62   {
63         Entry *n = e->next;
64         Entry *p = e->prev;
65         p->next = n;
66         n->prev = p;
67   }
68   delete e;
69 }
```

Vorteile der doppelt verketteten Liste:

- einfach zu verstehen
- einfach Vorwärts oder Rückwärts auszulesen oder zu durchsuchen
- man kann sehr schnell den ersten Wert auslesen
- man kann sehr schnell den nächsten Wert auslesen
- man kann sehr schnell den vorherigen Wert auslesen
- man kann sehr schnell einen Wert am Anfang einfügen
- man kann schnell einen Wert an beliebiger Stelle einfügen
- man kann sehr schnell den ersten Wert entfernen.
- man kann schnell einen Wert an beliebiger Stelle einen Wert an beliebiger Stelle entfernen.

Nachteile der doppelt verketteten Liste:

- es ist teuer (an rechenzeit) den n-ten Wert auszulesen
- der Overhead ist doppelt so groß wie bei der einfach verketteten Liste.
- es ist teuer einen Wert am Ende anzuhängen.

Und auch hier kann man das Einfügen und Auslesen am Ende stark vereinfachen und beschleunigen in dem man zusetztlich einen Zeiger auf den letzten Eintrag verwaltet.

9.4. Binäre Bäume

Bisher waren unsere Datenstrukturen eindimensional. Bäume sind ein Beispiel für 2 dimensionale Stukturen. Sie werden verwendet um Hierarchien oder sturkturierte Informationen zu speichern. Der einfachste Fall sind binäre Bäume. Jeder Knoten/Blatt hat zwei Pointer zusätzlich zur eigentlichen Information, einer nach links und einer nach rechts. Für den gesammten Baum gilt eine Vorschrift, wann ein neuer Knoten rechts und wann er links angehängt wird. Zeigt der entsprechende Pointer bereits auf einen anderen Knoten, so wir versucht den neuen Knoten dort anzuhängen, bis man am Ende bei einem freien Pointer (0) angekommen ist.

Einer der Vorteile eines binären Baumes ist, das der Zugriff auf einen bestimmten Wert oder Knoten im mittel deutlich schneller ist als bei einer verketteten Liste.

Da wir im Moment noch nicht so weit sind den Computer koplizierter Grafiken anzeigen zu lassen, behandeln wir binäre Bäume zunächst auf den Papier, da man sonst die Struktur des Baumes nicht erkennen kann.

9.4.1. Zeichenketten sortieren 1

Unser erster Baum soll eine Zeichenkette speichern. Die Vorschrift lautet:

ist der neue Wert größer als der Wert des aktuellen Knotens, gehört der neue Wert nach rechts, ansonsten nach links.

Als machen wir mal den Versuch und tragen "Hallo“ in einen leeren Baum ein.

```
1  struct Knoten {
2 char Wert;
3 Knoten *links;
4 Knoten *rechts;
5 };
6
7 Knoten wurzel=0;
8
9 /* füge einen neuen Wert n in den Baum ein */
10 void insert(char n)
11 {
12   Knoten *neu = new Knoten;
13   neu->Wert = n;
14   neu->rechts = 0;
15   neu->links = 0;
16   if(wurzel==0)
17   {
18     wurzel = neu;
19   }
20   else
21   {
22    Knoten *i = wurzel;
23    while(i)
```

```
24      {
25        if(i->Wert<n && i->rechts!=0) i=rechts;
26        if(i->Wert>=n && i->links!=0) i=links;
27        if(i->Wert<n && i->rechts==0) { rechts = neu;i=0; }
28        if(i->Wert>=n && i->links=0)  { links  = neu;i=0; }
29      }
30    }
31 }
32
33 /* Finde einen Knoten mit Wert==w unterhalb von Knoten r */
34 Knoten *findeKnoten(Knoten *r,char w)
35 {
36         if(r==0) return 0;
37         if(r->Wert == w ) return r;
38         if(r->Wert < w ) return findeKnoten(r->rechts,w);
39         else return findeKnoten(r->links,w);
40 }
41
42 /* alle Knoten von links nach rechts ausgeben */
43 void ListKnoten(Knoten *r)
44 {
45         if(r==0) return;
46         cout << r->Wert << endl;
47         if(r->links) ListKnoten(r->links);
48         if(r->rechts) ListKnoten(r->rechts);
49 }
```

Was man an diesem Beispiel gut sehen kann, ist eine Technik namens Rekursion, bei der ein Funktion sich selbst wieder mit neuen Parametern aufruft. Diese Technik ist beim Umgang mit komplexeren Strukturen oft sehr elegant und schnell zu schreiben. Sie hat aber auch einige Nachteile:

- Sie verbraucht leicht viel Speicherplatz auf dem Stack, was man oft nicht bemerkt bis das Programm plötzlich abbricht.
- Sie ist nicht besonders schnell in der Ausführung.
- Wenn man nicht aufpasst, kann die Rekursion endlos weitergehen bis der Speicher auf dem Stack verbraucht ist.

9.5. Objekte

9.5.1. Klassen

Die Klasse(**class**) geht noch einen Schritt weiter und bringt auch noch die zu dem Daten gehörenden Funktionen (**methoden**) mit dazu. Außerdem gibt es Zugriffsstuffen, Vererbung und Konstruktoren/Destruktoren.

```
1  class Notenbuch
2  {
3   /* Private Variablen */
4   double Noten[10];
5   int position;
6  public:
7   Notenbuch() {position=0;} // constructor
8   /* Öffentliche Methoden */
9   void neueNote(double n);
10  void zeigeNoten();
11  double leseNote(int p);
12  double durchschnitt();
13  int anzahl();
14  void Laden(char* filename);
15  void Speichern(char *filename);
16 };
17
18 int main()
19 {
20 Notenbuch Buch;
21 Buch.neueNote(1);
22 Buch.neuenNote(2);
23 Buch.zeigeNoten();
24 Buch.durchschnitt();
25 cout << ''Anzahl:`` << Buch.anzahl() << endl;
26 }
```

9.5.2. Kapselung und Zugriffsstuffen

9.5.3. Methoden

9.5.4. Konstruktoren und Destruktoren

9.5.5. Klasse vs. Instanz

9.5.6. Vererbung

Teil V.
Die fünfte Stunde

10. STL Container

11. STL Algorithmen

Teil VI.
Die sechste Stunde

Bisher haben wir den C++ Compiler nur durch das Menu der Entwicklungsumbgebung aufgerufen, ohne uns Gedanken zu machen was hinter den kulissen passiert. Außerdem bestanden unsere Programme bisher immer nur aus einer einzelnen Datei. In diesem Abschnitt geht es darum was hinter den Kulissen passiert und wie man damit umgeht wenn die Programme immer länger und komplexer werden.

12. Compiler

13. Bibliotheken

14. Projekte aus mehr als einer Datei

15. Makefiles und Co

16. Projektemanagement

Teil VII.
Die siebte Stunde

Nach dem wir bisher immer nur die Tastatur verwenden konnten, bereiten wir uns jetzt langsam darauf vor die Maus ins Spiel zu bringen.

17. Events und Mainloop

18. GUI

19. Qt/KDE - Einstieg

19.1. Graphisches Hello World

```
1  #include <qapplication.h>
2  #include <qlabel.h>
3
4
5  int main(int argc, char *argv[])
6  {
7          QApplication app(argc,argv);
8          QLabel hello(QString("Hallo Welt"),0,"test");
9          hello.resize(150,50);
10         hello.show();
11         return app.exec();
12 }
```