C++
Programmbeispiele

Programmieren II
Übungsaufgaben

Implementierung einer linearen Liste

```
#include <iostream>
#include <string>

struct ListItem
{
    char     Name[30],
             PhoneNr[30];
    ListItem *Next;
};

ListItem     *ListRoot = (ListItem *) NULL;
```

```
// add_item fügt ein neues Listenelement am Anfang der Liste ein
// und verändert die globale Listenwurzel entsprechend
ListItem *add_item(char *Name, char *PhoneNr)
{
    ListItem  *LItem;

    LItem = new ListItem;
    if (LItem == (ListItem *) NULL)
       return (ListItem *) NULL;

    strcpy(LItem->Name, Name);
    strcpy(LItem->PhoneNr, PhoneNr);

    LItem->Next = ListRoot;
    ListRoot = LItem;

    return LItem;
}
```

```
// print_list gibt alle in der Liste gespeicherten Daten
// entsprechend ihrer Position in der Liste aus
void print_list()
{
    ListItem *LItem = ListRoot;

    while (LItem)
    {
       cout << ">> " << LItem->Name << ": " << LItem->PhoneNr <<
    endl;
       LItem = LItem->Next;
    }
}
```


Erweiterung: sortierte Datenhaltung

```
ListItem *add_item(char *Name, char *PhoneNr)
{
    ListItem  *LItem = new ListItem;
    if (LItem == (ListItem *) NULL)
       return (ListItem *) NULL;

    strcpy(LItem->Name, Name);
    strcpy(LItem->PhoneNr, PhoneNr);

    LItem->Next = ListRoot;
    ListRoot    = LItem;
    return LItem;
}
```

```
// add_item2 - sortiertes Einfügen eines neuen Elements
ListItem *add_item2(char *Name, char *PhoneNr)
{
    ListItem  *LItem, *LIp;

    LItem = new ListItem;
    if (LItem == (ListItem *) NULL)
       return (ListItem *) NULL;

    strcpy(LItem->Name, Name); strcpy(LItem->PhoneNr, PhoneNr);

    if (ListRoot == (ListItem *) NULL ||
        strcmp(Name, ListRoot->Name) < 0)
    {
       LItem->Next = ListRoot;
       ListRoot    = LItem;
       return LItem;
    }
    else
    {
       LIp = ListRoot;
       while (LIp->Next &&
              strcmp(LIp->Next->Name, Name) < 0)
          LIp = LIp->Next;
       LItem->Next = LIp->Next;
       LIp->Next   = LItem;
       return LItem;
    }
}
```

Wegesuche in einem Labyrinth (1)

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


char   Labyrinth[80][80];
int    Lx, Ly;


// liest das Labyrinth aus der angegebenen Datei
void LeseLabyrinth(char *LName)
{
    FILE     *fp;
    int      y;

    if (!(fp = fopen(LName, "r")))
    {
       fprintf(stderr, "Labyrinth-Datei existiert nicht\n");
       getchar();
       exit(1);
    }
    fscanf(fp, "%d %d\n", &Lx, &Ly);

    for (y = 0; y< Ly; y++)
       fgets(Labyrinth[y], 80, fp);

    fclose(fp);
}



// gibt das Labyrinth auf dem Bildschirm aus
void DruckeLabyrinth()
{
    int      y;

    printf("\f\n\n");
    for (y = 0; y < Ly; y++)
       printf(Labyrinth[y]);
}
```


Wegesuche in einem Labyrinth (2)

```
// Reinitialisiert das Labyrinth, d.h. alle Eintraege mit
// Ausnahme der Hindernisse werden gelöscht

void ReinitLabyrinth()
{
    int      x, y;

    for (y = 0; y < Ly; y++)
       for (x = 0; x < Lx; x++)
              if (Labyrinth[y][x] != 'X')
                     Labyrinth[y][x] = ' ';
}



// Sucht einen Weg von sx,sy nach zx,zy durch Rekursion
// Liefert 1, falls ein Weg gefunden wurde, sonst 0

int SucheWeg(int sx, int sy, int zx, int zy)
{
    if (sx == zx && sy == zy)
       return 1;
    if (Labyrinth[sy][sx] != ' ')
       return 0;

    Labyrinth[sy][sx] = '>';
       if (SucheWeg(sx+1, sy  ,  zx, zy))     return 1;
    Labyrinth[sy][sx] = '<';
       if (SucheWeg(sx-1, sy  ,  zx, zy))     return 1;
    Labyrinth[sy][sx] = 'v';
       if (SucheWeg(sx  , sy+1,  zx, zy))     return 1;
    Labyrinth[sy][sx] = '^';
       if (SucheWeg(sx  , sy-1,  zx, zy))     return 1;

    Labyrinth[sy][sx] = '.';     // WICHTIG!!!
    return 0;
}
```

Wegesuche in einem Labyrinth (3)

```
main()
{
    int      Sx, Sy, Zx, Zy;
    char     test;

    LeseLabyrinth("D:\\Eigene
    Dateien\\Fachhochschule\\Vorlesungen\\C++\\src\\labyrinth\\D
    ebug\\test.lab");

    do {
       ReinitLabyrinth();
       DruckeLabyrinth();
       do {
             printf("\nStart, Ziel: ");
             scanf("%d %d %d %d", &Sx, &Sy, &Zx, &Zy);
             if (Labyrinth[Sy][Sx] == 'X')
                   printf("Startpunkt liegt auf Hindernis!\n");
             if (Labyrinth[Zy][Zx] == 'X')
                   printf("Zielpunkt liegt auf Hindernis!\n");
       } while (Labyrinth[Sy][Sx] == 'X' ||
                Labyrinth[Zy][Zx] == 'X');

       SucheWeg(Sx,Sy,  Zx,Zy);
       Labyrinth[Sy][Sx] = 'S';
       Labyrinth[Zy][Zx] = 'Z';

       DruckeLabyrinth();
       printf("\nWeitere Suche? ");
       fflush(stdin);
       scanf("%c", &test);
    } while (toupper(test) != 'N');

    return 0;
}
```


Einführung in benutzerdefinierte Datentypen: Implementierung eines Stacks (1)

```
#include <iostream>


class Stack
{
   int         *buf;          // Membervariablen
   int         top;
   int         max_size;

public:                       // Zugriffskontrolle
   Stack(int s = 10);         // Konstruktor
   ~Stack();                  // Destruktor

   void push(int data);       // Elementfunktionen
   int  pop();
};

Stack::Stack(int s)           // Konstruktor:
{                             // Auto. Init
   top = 0;
   max_size = s;
   buf = new int[s];          // Freispeicher
   anfordern
}

Stack::~Stack()               // Destruktor
{
   delete [] buf;
}
```

Definition eines Stacks (2)

```
void Stack::push(int data) // Elementfunktion
{
   if (top < max_size)
     buf[top++] = data;
   else
     cout << "Stack full\n";
}

int Stack::pop()
{
   if (top > 0)
     return buf[--top];
   else
   {
     cout << "Stack empty ";
     return -1;
   }
}
```

```
main()              // Verwendungsbeispiele
{
   Stack  var1;
   Stack  *var2 = new Stack(20);
   Stack  var3[100];

   var1.push(100);       // Zugriff auf neue DT
   var2->push(200);      // Aufruf der Elementfkt.
   var1.push(101);
   var2->push(201);
   var3[10].push(1000);
}
```


Klasse Datum

```
#include <iostream>
using namespace std;

class Datum
{
   int    Tag, Monat, Jahr;

public:
   Datum(int t, int m, int j)
              { Tag=t; Monat=m; Jahr=j; }
   void SetzeTag(int t)   { Tag = t; }
   void SetzeMonat(int m) { Monat = m; }
   void SetzeJahr(int j)  { Jahr = j; }
   int  LeseTag()   { return Tag; }
   int  LeseMonat() { return Monat; }
   int  LeseJahr()  { return Jahr; }
   // int Differenz(Datum d1, Datum d2);
   // int Wochentag(Datum d);
   // Datum Ostern(int Jahr);
   // Datum Fasching(int Jahr);
};


void main()
{
   Datum d1(24,12,1999);
   Datum d2(1,1,1999);
   cout << "Tag:  " << d1.LeseTag() << endl;
   cout << "Monat:" << d1.LeseMonat() << endl;
   cout << "Jahr: " << d1.LeseJahr() << endl;
   cout << "d2:   " << d2.LeseTag() << "." <<
   d2.LeseMonat() << "." << d2.LeseJahr() <<
   endl;
}
```

Beispielausgabe der Klasse 'Datum'

```
void main()
{
   Datum d1(24,12,1999);
   Datum d2(1,1,1999);

   cout << "Tag:   " << d1.LeseTag() << endl;
   cout << "Monat: " << d1.LeseMonat() << endl;
   cout << "Jahr:  " << d1.LeseJahr() << endl;

   cout << "d2:    " << d2.LeseTag() << "." <<
   d2.LeseMonat() << "." << d2.LeseJahr() <<
   endl;
}
```

```
Tag:   24
Monat: 12
Jahr:  1999
d2:    1.1.1999
```


Um Standarddatum erweiterte Klasse Datum

```
#include <iostream>
using namespace std;

class Datum
{
   int    Tag, Monat, Jahr;
   static Datum StdDatum;

public:
   Datum(int t=0, int m=0, int j=0);
   static void SetzeStdDatum(int t, int m, int j);
   void DruckeDatum() { cout << Tag << "." <<
                        Monat << "." << Jahr; }
   // ...
};


Datum::Datum(int t, int m, int j)
{
   Tag   = (t != 0) ? t : StdDatum.Tag;
   Monat = (m != 0) ? m : StdDatum.Monat;
   Jahr  = (j != 0) ? j : StdDatum.Jahr;
}


void Datum::SetzeStdDatum(int t, int m, int j)
{
   StdDatum.Tag   = t;
   StdDatum.Monat = m;
   StdDatum.Jahr  = j;
}


Datum Datum::StdDatum(1,1,1999);
          // tatsächliches Definieren des StdDatums
          // globale Variable!!!
```

Beispielausgabe der erweiterten Klasse 'Datum'

```
void main()
{
   Datum d1(24,12,1999);
   Datum d2;

   cout << endl << "d1: "; d1.DruckeDatum();
   cout << endl << "d2: "; d2.DruckeDatum();

   Datum::SetzeStdDatum(24,12,1999);
   Datum d3;
   Datum d4(11);

   cout << endl << "d3: "; d3.DruckeDatum();
   cout << endl << "d4: "; d4.DruckeDatum();
}
```

```
d1: 24.12.1999
d2: 1.1.1999
d3: 24.12.1999
d4: 11.12.1999
```


Lineare Liste als Klasse (1)

```
#include <iostream>
#include <string>

using namespace std;


class CLinListe
{
   struct ListItem
   {
     char    Name[30];
     ListItem   *next;

     ListItem(char nName[])
          { strcpy(Name, nName); next = NULL; }
   };
   ListItem *first, *aktuell;

public:
   CLinListe() { first = NULL; aktuell = NULL;}
   void   Append(char nName[]);
   void   Insert(char nName[]);
   void   Print();
   void   Begin() { aktuell = first; }
   void   GotoNext()
       { if (aktuell) aktuell = aktuell->next; }
   char   *GetCurrentName()
       { return (aktuell != NULL) ?
                          aktuell->Name : NULL; }
};
```

Lineare Liste als Klasse (2)

```
// Ausgeben aller Listenelemente

void CLinListe::Print()
{
   ListItem    *lp = first;

   while (lp != NULL)
   {
     cout << lp->Name << endl;
     lp = lp->next;
   }
}


// Append fügt ein neues Element am Ende an

void CLinListe::Append(char nName[])
{
   ListItem    *newItem = new ListItem(nName);

   if (first == NULL)
     first = newItem;          // Erstes Element?
   else
   {
     ListItem *lp = first;

     while (lp->next)          // Ende suchen
          lp = lp->next;

     lp->next = newItem;       // Element anfügen
   }
}
```


Lineare Liste als Klasse (3)

```
// Insert fügt ein neues Element alphabetisch sortiert ein

void CLinListe::Insert(char nName[])
{
   ListItem    *newItem = new ListItem(nName);

   if (first == NULL)    // Ex. eine Liste???
     first = newItem;
   else
   {
     if (strcmp(nName, first->Name) < 0)
     {  // Ist das Element als erstes einzusortieren???
          newItem->next = first;
          first         = newItem;
     }
     else       // Irgendwo in der Mitte einsortieren
     {
          ListItem *lp = first;

                        // Einfügestelle suchen und ...
          while (lp->next &&
              strcmp(lp->next->Name, nName) < 0)
               lp = lp->next;

          newItem->next = lp->next; // ... einfügen
          lp->next   = newItem;
     }
   }
}
```

Lineare Liste als Klasse (4)

```
void main()
{
   char        Name[30];
   char        *currName;
   CLinListe   liste1;

   do
   {
      cout << "Bitte Name eingeben [stop = Ende] ";
      cin >> Name;
      liste1.Insert(Name);     // Einfügen
      // liste1.Append(Name);  // Anfügen

      liste1.Print();
      cout << "--------------------" << endl;

      liste1.Begin();
      while (currName = liste1.GetCurrentName())
      {
           cout << currName << endl;
           liste1.GotoNext();
      }
   } while (strcmp(Name, "stop") != 0);
}
```


Überladen von Operatoren
Beispiel: Rationale Zahlen (1)

```
#include <iostream>

class rational
{
   int         zaehler;
   int         nenner;
   void        ggt();
public:
   rational(int z=0, int n=1) :
                    zaehler(z), nenner(n) {}
   int      z() { return zaehler; }
   int      n() { return nenner;  }
   rational operator+(rational);
   rational operator*(rational);
};


void rational::ggt(void)
{
   int   t;
   int   u = zaehler;
   int   v = nenner;

   while (v != 0)
   {
     t = u % v;
     u = v;
     v = t;
   }
   zaehler /= u;
   nenner  /= u;
}
```

Überladen von Operatoren
Beispiel: Rationale Zahlen (2)

```
rational rational::operator+(rational b)
{
   rational    r;

   r.zaehler =   zaehler * b.nenner  +
               b.zaehler * nenner;
   r.nenner  = nenner * b.nenner;

   r.ggt();
   return r;
}


rational rational::operator*(rational b)
{
   rational    r;

   r.zaehler = zaehler * b.zaehler;
   r.nenner  = nenner  * b.nenner;

   r.ggt();
   return r;
}



ostream  &operator<<(ostream &s, rational r)
{
   return s << r.z() << '/' << r.n();
}
```


Überladen von Operatoren
Beispiel: Rationale Zahlen (3)

```
main()
{
   rational    a(2,3);
   rational    b(3,5);

   rational    c=a+b;
   rational    d=a*b;

   cout << "a: " << a << endl;
   cout << "b: " << b << endl;

   cout << "c: " << c << endl;
   cout << "d: " << d << endl;

   cout << "a+b*c+d: " << a+b*c+d<< endl;
}
```

Programmausgaben:

```
a: 2/3
b: 3/5
c: 19/15
d: 2/5
a+b*c+d: 137/75
```

AUFZUG.H

```
#ifndef _AUFZUG
#define _AUFZUG

#include "simconsts.h"

enum FRichtung {
   NichtGedrueckt = 0, NachOben=1, NachUnten=2
};

class CAufzug
{
   unsigned m_SystemZeit;
   int         m_AktGewicht;
   FRichtung   m_Anforderungssensor[AnzEtagen];
   bool        m_TuerOffen;
   int         m_Etage;
   bool        m_Fahrziele[AnzEtagen];
   unsigned    m_letzteAktion;
   int         m_Fahrtrichtung;    // -1, 0, +1
   bool        m_DEBUG;

   void        FahrStrategie();
   bool        FahrtrichtungNichtMehrGueltig();
   int         SucheNaechsteAnforderung();
public:
               CAufzug(bool debug = false);
   void        DoOneLiftCycle();
   bool        AufzugBetreten(int Gew, int Ziel);
   void        AufzugVerlassen(int Gewicht);
   int         AktEtage() const { return m_Etage;}
   void        AufzugAnforderung(int Et, int R);
};
#endif
```


AUFZUG.CPP

```
#include <iostream>
#include "aufzug.h"

CAufzug::CAufzug(bool debug)
{
   for (int i=0; i<AnzEtagen; i++) {
     m_Anforderungssensor[i] = NichtGedrueckt;
     m_Fahrziele[i] = false;
   }
   m_TuerOffen      = false;
   m_Etage          = 0;
   // ...
}

void CAufzug::DoOneLiftCycle()
{
   if (m_TuerOffen) {
     if (m_SystemZeit > m_letzteAktion+6) {
               // Automatisches Tuerschliessen
          m_TuerOffen = false;
          if (m_DEBUG) cout << "\t[" << m_Etage
                            << "] Tuer Schliessen\n";
          m_SystemZeit += ZeitTuerSchliessen;
          if (m_AktGewicht == 0 &&
              FahrtrichtungNichtMehrGueltig())
               m_Fahrtrichtung = 0;
     }
     else
          m_SystemZeit += ZeitWarten;
   }
   else
     FahrStrategie();    // ohne Objekt ???
}
```

MAIN.CPP

```
#include <iostream>
#include "sim.h"
#include "aufzug.h"

main()
{
   int              AnzPersonen;
   int              Auslastung = 10;
   CAufzug          Aufzug;

   cout << "Zu simulierende Anzahl Personen: ";
   cin >> AnzPersonen;

   cout << "Angenommene Auslastung [%]: ";
   cin >> Auslastung;

   CSimulator  Sim(Auslastung);

   do {
     Sim.DoOneSimCycle(Aufzug);
   } while (Sim.TotalPersAnzahl() <
            AnzPersonen);

   cout << "Anz. Personen:      "
        << Sim.TotalPersAnzahl() << endl;
   cout << "Abbruchquote:       "
        << Sim.AbbrQuote() << endl;
   // ...
}
```

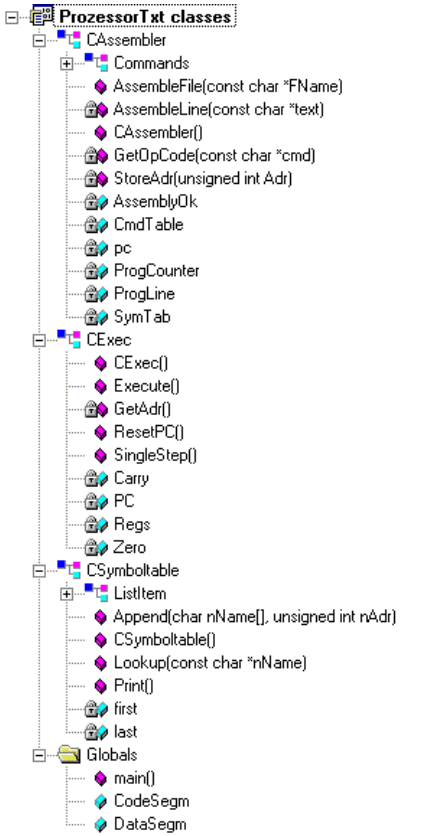

SIM.H

```
#ifndef _SIMULATOR
#define _SIMULATOR

#include "fifo.h"
#include "person.h"
#include "aufzug.h"

const     int  MaxPersonen = 30;
const     int  Geduld = 60;

class CSimulator {
   CFifo<CPerson>   m_PersVorAufzug;
   CFifo<CPerson>   m_PersImAufzug;
   unsigned         m_maxPersonenImAufzug;
   unsigned         m_maxPersonenWartend;
   int              m_PersonenAktuell;
   int              m_PersonenTotal;
   int              m_AbbruchTotal;
   double           m_SumTransZeit;
   int              m_Auslastung;
   bool             m_DEBUG;
public:
                    CSimulator(int Ausl = 10,
                          bool debug = false);
   void             DoOneSimCycle(CAufzug &A);
   double           AbbrQuote() const
     { return (double) m_AbbruchTotal /
                    (double) m_PersonenTotal; }
   int              TotalPersAnzahl() const
                    { return m_PersonenTotal; }
   // ...
};
#endif
```

SIM.CPP

```
#include <iostream>
#include "sim.h"

CSimulator::CSimulator(int Auslastg, bool debug)
{
   m_PersonenTotal       = 0;
   m_AbbruchTotal        = 0;;
   m_maxPersonenImAufzug = 0;
   m_maxPersonenWartend  = 0;
   m_DEBUG               = debug;
   m_Auslastung          = Auslastg;
   // ...
}

void CSimulator::DoOneSimCycle(CAufzug &Aufzug)
{
   int    i, size;

   if (m_PersonenAktuell < MaxPersonen &&
         rand() % 100 < m_Auslastung)
   {     // Neue Person erzeugen
     CPerson   NPerson(Aufzug.AktZeit());

     m_PersVorAufzug.enqueue(NPerson);
     Aufzug.AufzugAnforderung(NPerson.Start(),
                       NPerson.Fahrtrichtung());
     m_PersonenAktuell ++;
     m_PersonenTotal   ++;
   }
   Aufzug.DoOneLiftCycle();
   // ...
}
```


Projekt Prozessorsimulation Klassenübersicht



Projekt Prozessorsimulation
Assembler.h

```
#ifndef __Assembler      // Include-Wächter
#define __Assembler

#include "common.h"

class CSymboltable {
   struct ListItem {
       char       Name[20];
       unsigned   Adr;
       ListItem   *next;

       ListItem(char nName[], unsigned nAdr);
   } *first, *last;
public:
           CSymboltable() { first = NULL; }
   void    Append(char nName[], unsigned nAdr);
   int     Lookup(const char *nName);
};

class CAssembler {
   struct Commands {
     char      *Cmd;
     OpCode    Code;
   };
   static Commands CmdTable[];

   CSymboltable     SymTab;   // Initialisierung???
   int              ProgCounter, ProgLine;
   unsigned char    *pc;
   bool             AssemblyOk;

   OpCode GetOpCode(const char *cmd);
   int    AssembleLine(const char *text);
   void   StoreAdr(unsigned Adr);

public:
   CAssembler() {};
   bool   AssembleFile(const char *FName);
};

#endif
```


Projekt Prozessorsimulation
Assembler.cpp (1)

```
#include <iostream>
#include <iomanip>
#include <string.h>
#include <stdlib.h>
#include "Assembler.h"

using namespace std;

// Initialisierung eines Listenelements
inline CSymboltable::ListItem::ListItem(
                    char nName[], unsigned nAdr)
{
   strcpy(Name, nName);
   Adr  = nAdr;
   next = NULL;
}



// Append fügt ein neues Element am Ende der Liste an
void CSymboltable::Append(
                    char nName[], unsigned nAdr)
{
   ListItem *newItem = new ListItem(nName,nAdr);

   if (first == NULL)
     first = last = newItem;
   else
   {
     last->next = newItem;
     last = newItem;
   }
}
```

Projekt Prozessorsimulation
Assembler.cpp (2)

```
// Lookup sucht nach dem übergebenen String und
// liefert die abgespeicherte Adresse

int CSymboltable::Lookup(const char *nName)
{
   ListItem    *lp = first;

   while (lp)
   {
     if (strcmp(lp->Name, nName) == 0)
          return lp->Adr;
     else
          lp = lp->next;
   }
   cout << "Unknown Label: " << nName;
   return -1;  // Fehler: Name existiert nicht
}
```


Projekt Prozessorsimulation
Assembler.cpp (3)

```
// Initialisierung der Text-OpCode Konvertierungstabelle
CAssembler::Commands CAssembler::CmdTable[] =
{
   {"load",    Load},
   {"store",   Store},
   {"mov",     Mov},
   {"mvi",     Mvi},
   {"add",     Add},
   {"addc",    Addc},
   {"inc",     Inc},
   {"dec",     Dec},
   {"jump",    Jump},
   {"jnz",     Jnz},
   {"jnc",     Jnc},
   {"in",      In},
   {"out",     Out},
   {"stop",    Stop},
   {"",        Invalid}
};


// Suche den OpCode zu einem gegebenen Text
OpCode CAssembler::GetOpCode(const char *cmd)
{
   for (int i=0; CmdTable[i].Cmd[0] != '\0'; i++)
   {
      if (strcmp(CmdTable[i].Cmd, cmd) == 0)
          return CmdTable[i].Code;
   }

   cout << "Unknown Command: " << cmd << endl;
   return Invalid;
}
```

Projekt Prozessorsimulation
Assembler.cpp (4)

```
// Private Hilfsfunktion: Abspeichern einer Adresse im
// Codesegment
inline void CAssembler::StoreAdr(unsigned Adr)
{
   *pc++ = Adr >> 8;
   *pc++ = Adr & 255;
}

// Übersetzen einer Zeile
int CAssembler::AssembleLine(const char *line)
{
   char        label[20], cmd[20], val1[8];
   char        val2[8], val3[8];
   const char  *lp = line;
   int         nRead;
   OpCode      Code;
   unsigned    Adr;

   if (line[0] == '\0')           // Leere Zeile ???
     return 0;

   if (line[0] != ' ' && line[0] != '\t')
   {           // Beginnt die Zeile mit einem Label???
     if (sscanf(line, "%s", label) != 1)
          return -1;
     SymTab.Append(label, ProgCounter);
     lp = strchr(line, ' ');
   }           // Suche erstes Leerzeichen in der Zeile
   nRead = sscanf(lp, "%s %[^ \t\n,], %[^ \t\n,],\
             %[^ \t\n,]", cmd, val1, val2, val3);
   Code = GetOpCode(cmd);    // OpCode bestimmen
   switch (Code)

                          // Fortsetzung folgt ...
```


Projekt Prozessorsimulation
Assembler.cpp (5)

```
   switch (Code)
   {
   case Invalid:
     return -1;
     break;
   case Load:
   case Store:
     if (nRead != 3)
          return -2;
     *pc++ = Code;
     *pc++ = val1[0]-'a';
     StoreAdr(atoi(val2));
     ProgCounter += 4;
     break;

     // ...

   case Add:
   case Addc:
     if (nRead != 4)
          return -2;
     *pc++ = Code;
     *pc++ = val1[0]-'a';
     *pc++ = val2[0]-'a';
     *pc++ = val3[0]-'a';
     ProgCounter += 4;
     break;

     // ...
```

// Fortsetzung folgt ...

Projekt Prozessorsimulation
Assembler.cpp (6)

```
   case Jump:
   case Jnz:
   case Jnc:
     if (nRead != 2)
          return -2;
     *pc++ = Code;
     Adr = SymTab.Lookup(val1);
     if (Adr < 0)
          return -3;
     StoreAdr(Adr);
     ProgCounter += 3;
     break;

   case In:
   case Out:
     if (nRead != 2)
          return -2;
     *pc++ = Code;
     StoreAdr(atoi(val1));
     ProgCounter += 3;
     break;
   case Stop:
     *pc++ = Code;
     break;

   default:
     cout <<endl<< "Hmmm...Ausfahrt verpaßt???\n";
     break;
   }
   return 0;
}
```


Projekt Prozessorsimulation
Assembler.cpp (6.b1)

```
CAssembler::Commands CAssembler::CmdTable[] =
{
   {"load",    Load,     "ra"},
   {"store",   Store,    "ra"},
   {"mov",     Mov,      "rr"},
   {"mvi",     Mvi,      "rv"},
   {"add",     Add,      "rrr"},
   {"addc",    Addc,     "rrr"},
   {"inc",     Inc,      "r"},
   {"dec",     Dec,      "r"},
   {"jump",    Jump,     "a"},
   {"jnz",     Jnz,      "a"},
   {"jnc",     Jnc,      "a"},
   {"in",      In,       "a"},
   {"out",     Out,      "a"},
   {"stop",    Stop,     ""},
   {"",        Invalid,  ""}
};


OpCode CAssembler::GetOpCode(const char *cmd,
                             char* &Format)
{
   int         i;

   for (i=0; CmdTable[i].Cmd[0] != '\0'; i++) {
     if (strcmp(CmdTable[i].Cmd, cmd) == 0) {
          Format = CmdTable[i].Format;
          return CmdTable[i].Code;
     }
   }
   cout << "Unknown Command: " << cmd << endl;
   Format="";
   return Invalid;
}
```

Projekt Prozessorsimulation
Assembler.cpp (6.b2)

```
int CAssembler::AssembleLine(const char *line)
{
   char        label[20], cmd[20], val[3][8];
   const char  *lp = line;
   char        *format;
   int         nRead, i;
   OpCode      Code;
   // Leerzeilentest & Eintragen von Labels wie oben
   nRead = sscanf(lp, "%s %[^ \t\n,], %[^ \t\n,],
   %[^ \t\n,]", cmd, val[0], val[1], val[2]);

   Code = GetOpCode(cmd, format);
   if (strlen(format) != nRead-1)
     return -2;
   *pc++ = Code;
   ProgCounter += 1;

   for (i=0; format[i] != '\0'; i++) {
     switch (format[i]) {
     case 'r': // Typ ist ein Register
          *pc++ = val[i][0] - 'a';
          ProgCounter += 1;
          break;
     case 'a': // Typ ist eine Adresse
          StoreAdr(atoi(val[i]));
          ProgCounter += 2;
          break;
     case 'v': // Typ ist ein 8-Bit Wert
          *pc++ = atoi(val[i]);
          ProgCounter += 1;
          break;
     default:
          cout << "Fehler, unbekanntes Format!\n";
          break;
     }
   }
   return 1;
}
```


Projekt Prozessorsimulation
Assembler.cpp (7)

```
bool CAssembler::AssembleFile(const char *FName)
{
   FILE   *fp;
   char   line[200];

   ProgLine    = 0;      // Initialisierung der Assembler-
   ProgCounter = 0;      // Objektvariablen
   pc          = CodeSegm;
   AssemblyOk  = true;

   fp = fopen(FName, "r");
   if (fp == NULL)
     return false;

   while (fgets(line, 200, fp))
   {
     if (AssembleLine(line) < 0)
     {
          cout << setw(4) << ProgLine <<
             ": Error in line " << line << endl;
          AssemblyOk = false;
     }
     ProgLine ++;
   }

   fclose(fp);
   if (AssemblyOk)
     cout << "Assembly successfull: " <<
            ProgCounter << " Bytes generated\n";

   return AssemblyOk;
}
```

Projekt Prozessorsimulation
common.h

```
#ifndef __common
#define __common


enum OpCode {
   Invalid, Load, Store, Mov, Mvi, Add, Addc,
   Inc, Dec, Jump, Jnz, Jnc, In, Out, Stop };


extern unsigned char CodeSegm[];
extern unsigned char DataSegm[];


#endif
```


Projekt Prozessorsimulation
exec.h

```
#ifndef __Exec
#define __Exec

#include "common.h"



class CExec
{
   unsigned char Regs[8];
   bool          Carry, Zero;
   unsigned char *PC;

   unsigned      GetAdr();

public:
        CExec() { Carry = false; Zero = false;
                  PC = CodeSegm; }
   void ResetPC() { PC = CodeSegm; }
   bool SingleStep();
   void Execute();
};



#endif
```

Projekt Prozessorsimulation
Exec.cpp (1)

```
#include <iostream>
#include <iomanip>
#include "exec.h"

using namespace std;

// Lesen einer Adreßinfo aus dem Codesegment
inline unsigned CExec::GetAdr()
{
   unsigned    high = *PC++;
   unsigned    low  = *PC++;

   return high*256 + low;
}



// Ausführen des gesamten Programms im Codesegment
void CExec::Execute()
{
   while (SingleStep())
      ;
}
```


Projekt Prozessorsimulation
Exec.cpp (2)

```
// Ausführen des nächsten Befehls aus dem Codesegm.
bool CExec::SingleStep()
{
   OpCode   Cmd = static_cast<OpCode> (*PC++);
   char     Reg, Reg2;
   unsigned tmp;

   switch (Cmd)
   {
   case Load:
     Reg = *PC++;
     Regs[Reg] = DataSegm[GetAdr()];
     break;
   case Store:
     Reg = *PC++;
     DataSegm[GetAdr()] = Regs[Reg];
     break;
   case Mov:
     Reg  = *PC++;
     Reg2 = *PC++;
     Regs[Reg] = Regs[Reg2];
     break;
   case Mvi:
     Reg = *PC++;
     Regs[Reg] = CodeSegm[*PC++];
     break;
```

Fortsetzung folgt ...

Projekt Prozessorsimulation
Exec.cpp (3)

```
    case Add:
      Carry = false;
    case Addc:
      Reg  = *PC++;
      Reg2 = *PC++;
      tmp = (unsigned) Regs[Reg2]  +
            (unsigned) Regs[*PC++] + Carry;
      Regs[Reg] = tmp & 255;
      Carry = tmp > 255;
      Zero  = Regs[Reg] == 0;
      break;
    case Inc:
      Reg = *PC++;
      Regs[Reg]++;
      Zero = Regs[Reg] == 0;
      break;
    case Dec:
      Reg = *PC++;
      Regs[Reg]--;
      Zero = Regs[Reg] == 0;
      break;
    case Jump:
      PC = CodeSegm+GetAdr();
      break;
    case Jnz:
      if (Zero == false)
           PC = CodeSegm+GetAdr();
      else
           PC += 2;
      break;
```

Fortsetzung folgt ...

Projekt Prozessorsimulation
Exec.cpp (4)

```
    case Jnc:
      if (Carry == false)
           PC = CodeSegm+GetAdr();
      else
           PC += 2;
      break;
    case In:
      {
           unsigned val;
           tmp = GetAdr();
           cout << setw(3) << tmp << " ? ";
           cin >> val;
           DataSegm[tmp] = (unsigned char) val;
      }
      break;
    case Out:
      tmp = GetAdr();
      cout << setw(3) << tmp << ": " <<
                         (unsigned) DataSegm[tmp];
      break;
    case Stop:
      {
           char yn;
           cout << endl << "Continue (y/n) ?";
           cin >> yn;
           if (yn == 'n')
                return false;
      }
      break;
    default:
      cout << "Was machen wir denn hier ...???\n";
      break;
    }
    return true;
}
```

Projekt Prozessorsimulation
main.cpp

```
#include <iostream>
#include <iomanip>
#include "Assembler.h"
#include "Exec.h"

using namespace std;

unsigned char CodeSegm[1000];
unsigned char DataSegm[1000];



void main()
{
   CAssembler       Assembler;
   CExec            Exec;

   Assembler.AssembleFile("test.asm");
   Exec.Execute();
}
```


Übungsaufgabe Operatorüberladen (1)
Eigener Datentyp 'Point'

(Klausuraufgabe WS98/99, Zeitvorstellung: ca. 35 Min.)

- Bitte implementieren Sie (ohne die Verwendung der Klasse Vector der Standardbibliothek) eine neue Klasse CPoint, welche die Handhabung von Koordinaten vereinfachen soll. Beim Erzeugen eines Objekts dieser Klasse soll angegeben werden können, welche Dimension dieser Punkt hat; Beispiel:
  CPoint x(1); // 1D, ~ x hat nur eine x-Komponente
  CPoint p(2); // 2D ~ p hat x und y-Komponenten
  CPoint q(3); // 3D ~ q hat x,y und z-Komp. usw.
- Um die Dimension beliebig halten zu können, sollen die Komponenten in einem Feld abgelegt werden, welches zum Zeitpunkt der Objekterzeugung angefordert wird. Der Datentyp der Komponenten sei double.
- Implementieren Sie eine Klasse CPoint, die dieses leistet und folgende weitere Eigenschaften besitzt:
  - Der Zugriff auf eine Komponente soll wie ein Zugriff auf ein Feld geschehen; Beispiele:
    p[0] ~ x-Komponente des Punkts p
    q[1] ~ y-Komponente des Punkts q
  - Falls versucht wird, auf eine Komponente außerhalb der angegebenen Dimension zuzugreifen, so ist die Ausnahme "OutOfDimension" zu generieren.
  - Über obigen Komponentenzugriffsmechanismus sollen auch die Komponentenwerte gesetzt oder gelesen werden können, so daß zum Beispiel folgende Zugriffe möglich sind:
    p[0] = 100.0; // x-Komp. von p auf 100.0 setzen
    p[1] = 0.0; // y-Komp. von p auf 0.0 setzen
    x[0] += p[0]; // x-Komp. von p zur x-Komp. von x hinzuaddieren

Hinweis: Bitte überlegen Sie, welche eine Zugriffsart auf Elemente alle diese Zugriffe erlaubt! Geben Sie NICHT für jede Möglichkeit eine eigene Überladung an!!!

Übungsaufgabe Operatorüberladen (2)
Eigener Datentyp 'Point'

- Die Konstruktion eines 'Punkt-Objekts' aus einem anderen 'Punkt-Objekt' soll möglich sein
  - Beispiel: CPoint q(3);
  - CPoint p = q;
- Die Zuweisung eines Punkt-Objekts an ein (evtl. anderes) Punkt-Objekt soll nur dann gelingen, wenn die Dimension des Ausdrucks auf der rechten Seite der Zuweisung kleiner oder gleich der Dimension des Objekts auf der linken Seite ist. Falls die Dimension der rechten Seite echt kleiner ist, so werden nur die vorhandenen Komponenten kopiert; Beispiel:
  - CPoint p1(2), p2(2), q(3);
  - // ...
  - p1 = p2; // o.k.
  - q = p1; // o.k. es werden nur die in auch in p1 enthaltenen Komponenten von q überschrieben.
  - p2 = q; // Fehler, da die Dimension von q drei, von p2 jedoch nur zwei beträgt!
  - Im Fehlerfall ist die Ausnahme "DimensionMismatch" zu generieren.
- Die Ausnahmen sollen aus Komfortgründen sowohl einzeln als auch gesammelt als "CPointErrors" gefangen werden können.

Main – Programm

- Schreiben Sie ein einfaches main-Programm, welches nur die Aufgabe hat, die beiden Ausnahmen DimensionMismatch und OutOfDimension der Klasse CPoint hervorzurufen und abzufangen. Es ist ausreichend, wenn das main-Programm als Fehlerbehandlung lediglich eine Fehlermeldung auf dem Bildschirm ausgibt.

Übungsaufgabe Operatorüberladen
Eigener Datentyp 'Point' - Lösung (1)

```
class CPoint
{
   double *data;
   int    dim;
public:
   class  CPointErrors {};
   class  DimensionMismatch : public CPointErrors
      {};
   class  OutOfDimension : public CPointErrors {};

   CPoint (int l) { data = new double [dim = l];};
   CPoint (const CPoint &);   // Copy-Konstruktor
   CPoint & operator=(const CPoint &);
   ~ CPoint () { delete [] data; }

   double &operator[](int ind) {
     if (ind >= 0 && ind < dim)
       return data[ind];
     else {
       throw OutOfDimension();
       return data[0];
     } };
};
```

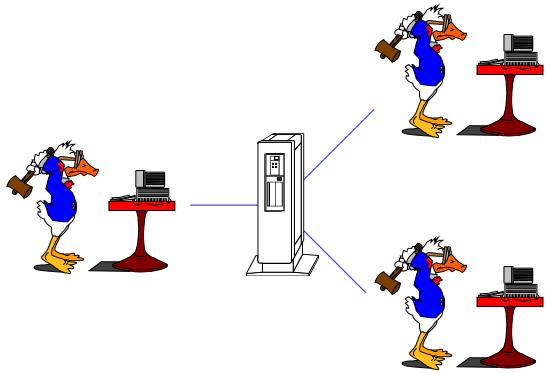
Übungsaufgabe Operatorüberladen
Eigener Datentyp 'Point' - Lösung (2)

```
CPoint:: CPoint (const CPoint &a)
     // Copy-Konstruktor
{
   data = new double[dim = a.dim];
   for (int i=0; i < dim; i++)
     data[i] = a.data[i];
}


CPoint & CPoint::operator=(const CPoint &a)
     // Zuweisungsoperator
{
   if (this != &a) { // Vorsicht bei Selbstzuweisung a = a
     if (dim < a.dim)
          throw DimensionMismatch();
     else
          for (int i=0; i < a.dim; i++)
               data[i] = a.data[i];
   }
   return *this;
}
```


Abhängigkeiten im Programm
Verwaltung durch "MAKE"

- aufzug.exe -- alle *.obj zusammenbinden
  - aufzug.obj
    - aufzug.cpp -- aufzug.cpp compilieren
      - aufzug.h
      - iostream
  - sim.obj
    - sim.cpp -- sim.cpp compilieren
      - iostream
      - sim.h
        - fifo.h
        - person.h
        - aufzug.h
  - person.obj
    - person.cpp -- person.cpp compilieren
      - iostream
      - stdlib.h
      - person.h
      - sim.h
        - fifo.h
        - person.h
        - aufzug.h
  - main.obj
    - main.cpp -- main.cpp compilieren
      - iostream
      - sim.h
        - fifo.h
        - person.h
        - aufzug.h !!!
      - aufzug.h

Einschub: Versionskontrollsysteme Motivation



- **Aspekte bei der Arbeit von mehreren Programmierern an einem Projekt**
  - **Möglichst unabhängige Arbeit (1), jedoch auf dem gleichen Datenbestand (2)**
    - **1 - lokale Projektkopie für jeden**
    - **2 - zentrale Verwaltung der Daten**
  - **Änderungen an einer Datei sollen erst dann an alle weitergehen, wenn die Routinen "laufen"**
    - **Schwierig bei zentraler Verwaltung**
  - **Es dürfen nicht zwei Personen gleichzeitig eine Datei (schreibend) editieren**
    - **Zugriffsregelung**
  - **Änderungen sollen (personell) erkennbar und auch (beliebig mehrstufig) wieder rückgängig gemacht werden können**

--> Versionsverwaltungssysteme

Versionsverwaltungssysteme

- **Idee:**
  - Gemeinsame Datenbasis (zentral)
  - Jeder Entwickler kann aus der Datenbasis eine Kopie (des gesamten Codebestands) erhalten
  - Jeder arbeitet auf seiner Kopie (nur lesbar!)
  - Sind Änderungen an einer Datei erforderlich, so wird diese "ausgecheckt" -- diese Datei ist dann beschreibbar
  - Hat die Datei nach den Programmtests einen befriedigenden Stand erreicht, so kann sie wieder "eingecheckt" werden
  - Konsequenz: ALLE Entwickler erhalten beim nächsten Compilieren automatisch eine aktuelle (NUR-LESE-) Kopie der geänderten Datei (Realisierung über MAKE)
  - Somit sind jetzt auch wieder bei allen Entwicklern die Dateien aktuell
- **Realisierung**
  - Änderungen an Dateien werden in der zentralen Datenbasis als Deltas zur vorangegangenen Version vermerkt
  - --> jede beliebige alte Version der Datei kann wiederhergestellt werden
  - --> jede einzelne Änderung kann nachvollzogen werden

Einschub: Versionsverwaltungssysteme (Beispiel SCCS)

man sccs(1)

NAME

sccs - Administration program for Source Code Control System (SCCS) commands

SYNOPSIS

sccs [**-r**] [**-d** pathname] [**-p** pathname] command [flags] [file ...]

The **sccs** command is an administration program that incorporates the set of SCCS commands into the Operating System.

clean

Removes from the current directory or the named directory all files that can be recreated from SCCS files. Does not remove files that are in the process of being edited.

create

Creates an SCCS file, copying the initial contents from a file of the same name. If the file creation is successful, the original file is renamed with a comma on the front. It is not necessary for you to move or remove the original file (as is the case with the **admin** command).

delget

Performs a **delta** command on the named files and then gets a new version. The new version of the files has expanded identification keywords, and cannot be edited.

SCCS (cont'd)

deledit

Equivalent to the **delget** pseudocommand, except that the **get** portion of the sentence includes the **-e** flag. The **deledit** option is useful for creating a checkpoint in your current editing session.

diffs

Shows the difference between the current version of the files you are editing and the versions in SCCS format.

edit

Equivalent to the **get -e** command.

info

Lists all the files being edited.

print

Prints information about named files. This SCCS pseudocommand is equivalent to the SCCS command **prs**.

tell

Lists all the files being edited on standard output, with a newline after each entry.

unedit

Equivalent to the **unget** command. Note that any changes made since the **get** command was used are lost.

SCCS Beispiele

1. To get a file for editing, edit it, and produce a new delta, enter:

```
$ sccs edit file.c
$ vi file.c
$ sccs delta file.c
```

3. To make a delta of a large number of files in the current directory, enter:

```
sccs delta *.c
```

4. To get a list of files being edited, enter:

```
sccs info
```

5. To make a delta of everything being edited by you, enter:

```
sccs delta `sccs tell -u`
```


sccs print <dateiname>

~wienkop/src/roamer/SCCS/s.c_lplan.c:

D 1.109 96/04/19 14:13:02 bauer 109 108 00021/00010/00817
COMMENTS: RDoorWay, Gyro reaktiviert, Rudi
:
D 1.107 96/04/10 17:08:52 reitz 107 106 00001/00000/00823
COMMENTS:

D 1.106 96/03/29 12:23:41 bauer 106 105 00000/00002/00823
COMMENTS: test_loc_influence raus

D 1.105 96/03/28 08:38:15 bauer 105 104 00003/00003/00822
COMMENTS:

D 1.104 96/03/25 12:50:59 lawitzky 104 103 00005/00000/00820
COMMENTS: new command MoveHallway introduced, Gisbert
:
D 1.102 96/03/25 10:39:52 soika 102 101 00002/00003/00817
COMMENTS:

D 1.101 96/03/22 16:38:16 bauer 101 100 00001/00001/00819
COMMENTS: Moveto function umgeschrieben, Rudi

D1.85 96/02/09 15:57:05 bauer 85 84 01022/01500/00000
COMMENTS:Umstellung auf Pilot-Cmds, remove inav, csi, Rudi

D 1.84 96/01/22 15:36:26 bauer 84 83 00001/00001/01499
COMMENTS: Einbau der Sprachausgabe, TalkString(), Rudi
:
D 1.27 94/09/26 12:51:35 bauer 27 26 00050/00049/01521
COMMENTS: Umstellung auf rmos, rudi

Ableiten von Klassen

"Auftrag" vom Kraftfahrtbundesamt, o.ä.
- Neue Organisation der Fahrzeugdaten -

- Gesucht ist eine neue, übersichtliche und leicht erweiterbare Organisation der verschiedenen Fahrzeugdaten
- zu erfassende Fahrzeugdaten von PKWs, Motorrädern und LKWs sind unterschiedlich und sollen später für die Weiterverarbeitung (z.B. Steuerberechnung) auch unterschiedlich behandelt werden können.
  - Für alle Fahrzeugtypen
    Kennzeichen, Jahr der Erstzulassung
  - PKWs, zusätzlich:
    Hubraum, Leistung, Schadstoffklasse
  - Motorräder, zusätzlich:
    Hubraum
  - LKWs, zusätzlich:
    Anzahl der Achsen, Zuladung
- Entgegen dem alten System möchte man sich bei der reinen Verwaltung der Daten (z.B. Datenablage, Suche nach Fahrzeugen gemäß Kennzeichen, etc.) nicht mehr mit unterschiedlichen Fahrzeugtypen "herumschlagen"

Fahrzeugliste-Header
FZliste.h (1)

```
#ifndef _FZLISTE
#define _FZLISTE

class CFZbase {
protected:  // Bei Ableitung wie public, sonst private
   char m_Kennzeichen[10];
   int  m_Erstzulassung;
public:
            CFZbase(char *k, int j);
            CFZbase();
   virtual  void    Print();     // darf überdef. werden
   virtual  CFZbase *Copy()=0;   // Abstrakte Klasse
   char     *Kennzeichen() {return m_Kennzeichen;}
   //virtual double Steuer()=0;
   //virtual int    TUEV()=0;
};
```

```
class CPKW : public CFZbase { // Ableitg. von Basisklasse
   int      m_hubraum;          // hier private
   int      m_leistung;
   short    m_schadstofftyp;
public:
   CPKW(char *Knz, int j, int h, int l, short s=0):
     CFZbase(Knz, j), m_hubraum(h),
     m_leistung(l), m_schadstofftyp(s) {};
        // Konstruktor ruft "Basiskonstruktor" mit Parametern auf
        // anschließend werden eigene Variablen initialisiert
   CPKW(); // Alternativer Defaultkonstruktor ohne Parameter
   virtual void Print();    // Überdefinition von Print
   virtual CFZbase *Copy(); // dto. für Copy
        // liefert identische Kopie des aktuellen Objekts
        // gilt für alle von CFZbase abgel. Klassen -> CFZbase *
};
```

Fahrzeugliste-Header
FZliste.h (2)

```
class CMotorrad : public CFZbase {
   int         m_hubraum;
public:
   CMotorrad(char *Kennz, int jahr, int h) :
     CFZbase(Kennz, jahr) , m_hubraum(h) {};
   CMotorrad();
   virtual void Print();
   virtual CFZbase *Copy();
};
```

```
class CLKW : public CFZbase {
   int         m_AnzAchsen, m_Zuladung;
public:
   CLKW();
   CLKW(char *Kennz, int jahr, int A, int Z) :
     CFZbase(Kennz, jahr),
     m_AnzAchsen(A), m_Zuladung(Z) {};
   virtual void Print();
   virtual CFZbase *Copy();
};
```

```
class CNode {  // Verwaltung aller von CFZbase abgel. Objekte
   CFZbase     *m_object;
   CNode       *m_left, *m_right; // binärer Baum
   int         CmpNode(CNode *);
public:
   CNode(CFZbase *obj) { m_object = obj->Copy();
   m_left = m_right = NULL; }
   void   Print();
   CFZbase *Suchen(char *Kennzeichen);
   CNode   *AddObject(CFZbase *obj);
};
#endif
```


Verwaltung der Fahrzeugdaten
FZliste.cpp (1) -- abstrakte Basisklasse

```
#include <iostream>
#include "FZliste.h"


inline CFZbase::CFZbase (char *k, int j)
{  // Konstruktor mit Parametern
   strcpy(m_Kennzeichen, k);
   m_Erstzulassung = j;
}



CFZbase::CFZbase()
{   // Konstruktor ohne Parameter
   cout << "Kennzeichen: ";
   cin  >> m_Kennzeichen;
   cout << "Jahr der Erstzulassung: ";
   cin  >> m_Erstzulassung;
}



void CFZbase::Print()
{
   cout << endl << m_Kennzeichen << endl;
   cout << "Erstzulassung:\t\t" <<
           m_Erstzulassung << endl;
}
```

Verwaltung der Fahrzeugdaten
FZliste.cpp (2) -- Verwaltung der PKWs

```
CPKW::CPKW()
{  // Konstruktor ohne Parameter: Abfragen von Tastatur
   // !Achtung! Der Default-Konstruktor der Basisklasse wird
   // automatisch aufgerufen! Siehe .h-Datei!
   cout << "Hubraum:\t\t" ;
   cin  >> m_hubraum;
   cout << "Leistung:\t\t";
   cin  >> m_leistung;
   cout << "Schadstoffklasse:[0..2]\t";
   cin  >> m_schadstofftyp;
}


void CPKW::Print()
{
   static char *Klassen[3] = {"Sauber",
   "Dreckschleuder", "Diesel"};

   CFZbase::Print(); // Ausgeben der Infos der Basisklasse
                     // CFZbase:: ist unbedingt notwendig!
   cout << "Typ:\t\t\tPKW\n";
   cout << "Hubraum:\t\t" << m_hubraum << endl;
   cout << "Leistung:\t\t" << m_leistung << endl;
   cout << "Schadstoffklasse:\t" <<
           Klassen[m_schadstofftyp] << endl;
}


CFZbase* CPKW::Copy()
{  // liefert identische Kopie des aktuellen Objekts
   return new CPKW(*this);
}
```


Verwaltung der Fahrzeugdaten
FZliste.cpp (3) -- Verwaltung der Motorräder

```
CMotorrad::CMotorrad()
{   // Konstruktor ohne Parameter: Abfragen von Tastatur
   cout << "Hubraum:\t\t" ;
   cin  >> m_hubraum ;
}


void CMotorrad::Print()
{
   CFZbase::Print();
   cout << "Typ:\t\t\tMotorrad\n";
   cout << "Hubraum:\t\t" << m_hubraum << endl;
}


CFZbase* CMotorrad::Copy()
{
   return new CMotorrad(*this);
}
```

Verwaltung der Fahrzeugdaten
FZliste.cpp (4) -- Verwaltung der LKWs

```
CLKW::CLKW()
{   // Konstruktor ohne Parameter: Abfragen von Tastatur
   cout << "Anzahl Achsen:\t\t" ;
   cin  >> m_AnzAchsen;
   cout << "max. Zuladung:\t\t" ;
   cin  >> m_Zuladung;
}


void CLKW::Print()
{
   CFZbase::Print();
   cout << "Typ:\t\t\tLKW\n";
   cout << "Anzahl Achsen:\t\t" << m_AnzAchsen
        << endl;
   cout << "max. Zuladung:\t\t" << m_Zuladung
        << endl;
}


CFZbase* CLKW::Copy()
{
   return new CLKW(*this);
}
```


Verwaltung der Fahrzeugdaten
FZliste.cpp (5) -- binärer Baum (1)

```
inline int CNode::CmpNode(CNode *node2)
{
   return strcmp(m_object->Kennzeichen(),
                 node2->m_object->Kennzeichen());
}


CNode *CNode::AddObject(CFZbase *obj)
{
   CNode  *root    = this;
   int    inserted = 0;
   CNode  *newnode = new CNode(obj);
     // Erzeugen eines neuen Knotens mit einer identischen Kopie
     // des angegebenen Objekts; ruft automatisch die jeweilige
     // Copy-Funktion auf, siehe Konstruktor von CNode!

   if (root == NULL) // Es ex. noch keine Wurzel d. Baums!
     return newnode;
   do {
     if (root->CmpNode(newnode) > 0) {
          // Gehört der neue Kn. in den li. oder re. Teilbaum?
          if (root->m_left == NULL) {
               // Blattknoten gefunden?
               root->m_left = newnode;
               inserted = 1;
          }
          else
               root = root->m_left;
     }
     else {
          /* ... */ // ... dto. für rechten Teilbaum
     }
   } while (inserted == 0);
   return this;
}
```

Verwaltung der Fahrzeugdaten
FZliste.cpp (6) -- binärer Baum (2)

```
CFZbase *CNode::Suchen(char *Kennz)
{
   int    suchstatus = 1;
   CNode  *Start = this;

   while (Start != NULL && suchstatus != 0) {
     suchstatus = strcmp(Kennz,
              Start->m_object->Kennzeichen());
     if (suchstatus != 0) { // Suche fortsetzen?
         if (suchstatus < 0)  // Kennzeichen kleiner
              Start = Start->m_left;
         else                 // Kennzeichen groesser
              Start = Start->m_right;
     }
   }

   if (suchstatus != 0)
     return NULL;   // Kennzeichen nicht gefunden!
   else
     return Start->m_object;
              // Gesuchtes Fahrzeugobjekt zurückliefern
}



void CNode::Print()
{
   if (m_left != NULL)
     m_left->Print();   // Daten des li. Teilbaums ausgeben
   m_object->Print();  // Ausgeben der Objektdaten
   if (m_right != NULL)
     m_right->Print(); // Daten des re. Teilbaums ausgeben
}
```


Hauptprogramm
Aufruf der einzelnen Komponenten

```
#include <iostream>
#include "FZliste.h"
main()
{
   char   Cmd;
   CNode   *root = NULL;

   do {
     cout << "Cmd: ";    cin  >> Cmd;
     switch (Cmd) {
     case 'p': // Neuen PKW anlegen
          root = root->AddObject(new CPKW);
          break;
     case 'm': // Neues Motorrad anlegen
          root = root->AddObject(new CMotorrad);
          break;
     case 'l': // Neuen LKW anlegen
          root = root->AddObject(new CLKW);
          break;
     case 's':  // Suche nach Kennzeichen
          char kennz[10];
          CFZbase *FZ;
          cin >> kennz;
          FZ = root->Suchen(kennz); // Suchen
          if (FZ != NULL)    // Ausgeben der
               FZ->Print(); // betreffenden FZdaten
          break;
     case 'd': // Ausgeben aller Daten aller Fahrzeuge
          root->Print();
          break;
     }
   } while (Cmd != 'x');
}
```

Graphikobjektverwaltung

- Entwerfen Sie die Datenhaltungskomponente für ein Graphikprogramm. Das Graphikprogramm soll in der ersten Ausbaustufe lediglich die Figuren Linie und Kreis kennen. Für die eigentliche Darstellung auf dem Bildschirm stehen aus einer Graphikbibliothek folgende Funktionen zur Verfügung:

```
DrawLine(int x1, int y1, int x2, int y2, int LineColor);
    // Linie von x1,y1 nach x2,y2 mit Farbe LineColor zeichnen

DrawCircle(int x, int y, int radius, int LineColor, int
    FillColor);
    // Kreis um x,y mit Radius in der Farbe LineColor zeichnen und
    // diesen mit der Farbe FillColor ausfüllen
```

- Entwerfen Sie entsprechende Klassen für die Verwaltung der genannten Figurtypen Linie und Kreis unter Hinzunahme einer gemeinsamen Basisklasse derart, daß alle Figuren z.B. in einem gemeinsamen Feld abgelegt werden können.
- Berücksichtigen Sie beim Entwurf, daß alle Variablen in den Klassen so lokal und geschützt (private oder protected) wie möglich gehalten werden sollen.
- Als Zugriffsfunktionen auf die Klassen werden jeweils folgende Funktionen benötigt:
  - Konstruktion des jeweiligen Objekts unter Angabe aller Parameter
  - Destruktion des Objekts
  - Funktion 'GetParams': Auslesen aller Parameter eines Objekts über Call-by-reference-Mechanismus
  - Funktion 'Plot': Aufruf der entsprechenden Graphikbibliotheksfunktion zur Ausgabe der Daten auf dem Bildschirm

Aufgaben (Graphikobjektverwaltung)

- Geben Sie von der Basisklasse und von der Klasse Kreis die vollständige Klassendefinition an
- Erstellen Sie ein Mini-Hauptprogramm, in dem Sie von jeder der zwei Klassen ein Objekt erzeugen und dieses so in ein gemeinsames Feld eintragen, daß später ein bequemer Durchlauf durch das Feld z.B. mittels einer for-Schleife möglich wird, etwa in der Art:

  "feld[0] = Kreis"

  "feld[1] = Rechteck"

  "feld[2] = Kreis"

- Programmieren Sie die for-Schleife, mit der Sie alle Objekte auf dem Bildschirm anzeigen lassen (Aufruf der Plot-Funktionen)

Graphikobjektverwaltung (1)

```
#include <iostream.h>

class CFigur
{
protected:
   int m_LineColor;
public:
   CFigur(int L) { m_LineColor = L; }
   ~CFigur() {}
   virtual void Plot() = 0;
};

class CKreis : public CFigur
{
   int m_x,m_y, m_r, m_FColor;

public:
   CKreis(int x, int y, int r, int LC, int FC) :
     CFigur(LC) {
     m_x = x; m_y = y; m_r = r; m_FColor = FC;
   }
   ~CKreis() {}
   void GetParams(int &x, int &y, int &r,
          int &LC, int &FC) {
     x = m_x; y = m_y; r = m_r;
     LC = m_LineColor; FC = m_FColor;
   }
   void Plot() {
     cout << "DrawCircle(" << m_x  << ", " <<
       m_y << ", " << m_r << ", " << m_LineColor
      << ", " << m_FColor << ")\n";
   }
};
```


Graphikobjektverwaltung (2)

```
void main()
{
   CFigur *feld[10];

   feld[0] = new CKreis(10,20, 1, 1, 0);
   feld[1] = new CLinie(10,10, 100,10, 5);
   feld[2] = new CKreis(10,20, 1, 1, 0);


   for (int i=0; i<3; i++)
     feld[i]->Plot();
}
```

Templates
Beispiel: Generisches FIFO (1)

```
template <class T> class CFifo
{
   T      *Data;
   int    fifosize;
   int    ElemCount;
   int    NxtIn, NxtOut;
   int    Next(int ind) {return (ind+1==fifosize)?
                                0 : ind+1;};
public:
   class  Overflow {};
   class  Underflow {};
          CFifo(int s = 15);
          ~CFifo();
   void   enqueue(const T &NewItem);
   T      dequeue();
   T      front();
   int    size() { return ElemCount; };
};


template <class T>  CFifo<T>::CFifo(int s)
{
   Data = new T[s];
   fifosize = s;
   NxtIn = NxtOut = ElemCount = 0;
}


template <class T>  CFifo<T>::~CFifo()
{
   delete [] Data;
}
```


FIFO (2)

```
template <class T>
     void CFifo<T>::enqueue(const T &NewItem)
{
   if (ElemCount == fifosize)
     throw Overflow();      // Fehler: Fifo voll
   Data[NxtIn] = NewItem;
   NxtIn       = Next(NxtIn);
   ElemCount ++;
}


template <class T>  T CFifo<T>::dequeue()
{
   if (ElemCount == 0)
     throw Underflow();

   int    Out = NxtOut;
   NxtOut = Next(NxtOut);
   ElemCount--;
   return Data[Out];
}


template <class T>  T CFifo<T>::front()
{
   if (ElemCount == 0)
     throw Underflow();

   return Data[NxtOut];
}
```

FIFO / Main

```
#include <iostream>
#include <string>
#include "fifo.h"

main()
{
   string           Name;
   char             Cmd;
   CFifo<string>    fifo(10);

   do {
     cout << endl << "Befehl: "; cin >> Cmd;
     try {
          switch (Cmd) {
          case 'i':
            cin >> Name;
            fifo.enqueue(Name);
            cout << fifo.size() << " Eintraege\n";
            break;
          case 'o':
            cout << fifo.dequeue() << endl
            break;
          }
     }
     catch (CFifo<string>::Overflow)
     {
          cerr << "Hey, kein Platz mehr!\n";
     }
     catch (CFifo<string>::Underflow)
     {
          cerr << "Null Daten 4U!!!\n";
     }
   } while (Cmd != 'x');  return 1; }
```