Klassen, Arten von Instanzen und Operatorüberladung.

Person.h

```
class Person
{
 private:
     char name[30];
     int alter;
 public:
     void setName(char * name);
     char * getName();
     void setAlter(int alter);
     int getAlter();
     Person();
     Person(char * name, int alter);

};
```

Person.cpp

```
#include "Person.h"
#include <string.h>

void Person::setName(char * name)
{
   strcpy(this->name, name);
}

char * getName()
{
   return this->name;
}

void Person::setAlter(int alter)
{
   this->alter = alter;
}

int Person::getAlter()
{
    return this->alter;
}

Person::Person()
{
   // tut nichts außer Speicher
  //  zuweisen.
}

Person::Person(char * name, int alter)
{
  // hier werden nach dem Instantieren die
 //  Variablen initialisiert.
  setName(name);
  setAlter(alter);
}
```

Main.cpp

```
#include <stdio.h>
#include <iostream.h>
#include "Person.h"

ostream& operator<<(ostream&, Person * p);
ostream& operator<<(ostream&, Person  p);

void main(void)
{
  char name[30];
  int alter;
  Person p1; // Statische Variable
  Person * pp1; // Dynamische Variable
  Person * pp2; // 2. Dynamische Variable

  cout << "Geben Sie einen Namen ein !";
  cin >> name;
  cout << "Geben Sie ein Alter ein !";
  cin >> alter;

  p1.setName(name);
  p1.setAlter(alter);

  // Leerer Konstruktor wird aufgerufen !
  pp1 = new Person();
  pp1->setName(name);
  pp1->setAlter(alter);

  // 2. Konstruktor wird aufgerufen.
  pp2 = new Person(name, alter);

   cout << p1;
   cout << pp1;
   cout << pp2;

}// main()-end


// "<<" – Operator wird überladen.
ostream& operator<<(ostream&, Person * p)
{
   cout << "\nName: " << p->name << "\n Alter: "
        << p->alter;
}

ostream& operator<<(ostream&, Person  p)
{
   cout << "\nName: " << p.name << "\n Alter: "
        << p.alter;
}
```

Person1.h

```
struct Person
{
 private:
     char name[30];
     int alter;
};
```

Virtuelle Funktionen, abstrakte Klassen !

```
class C1
{
 public:
  virtual void func1();

};

class C2 : public C1
{
 public:
  void func1(); // wird überlagert;

};

C1 c1;
C2 c2;
c1.func1();
c2.func1();
```

```
class C1
{
 public:
  virtual void func1()=0; // Klasse wird
                          // abstrakt !

};

class C2 : public C1
{
 public:
  void func1(); // überlagert C2::func1()

};

C1 c1; // NICHT ERLAUBT !!!
C2 c2;
c1.func1(); // NICHT ERLAUBT !!!
c2.func1();
```

Main.cpp

```
#include <stdio.h>
#include <string.h>
#include <malloc.h>
#include <iostream.h>
#include "Person1.h"
ostream& operator<<(ostream&, Person * p);
ostream& operator<<(ostream&, Person  p);

void main(void)
{
  char name[30];
  int alter;
  Person p1; // Statische Variable
  Person * pp1; // Dynamische Variable

  cout << "Geben Sie einen Namen ein !";
  cin >> name;
  cout << "Geben Sie ein Alter ein !";
  cin >> alter;

  strcpy(p1.name, name);
  p1.alter = alter;

  // Mittels malloc sapeicher reservieren !
  pp1 = (Person)malloc(sizeOf(Person));
  strcpy(pp1->name, name);
  pp1->alter = alter;;


   cout << p1;
   cout << pp1;


}// main()-end


// "<<" – Operator wird überladen.
ostream& operator<<(ostream&, Person * p)
{
   cout << "\nName: " << p->name << "\n Alter: "
        <<  p->alter;
}

ostream& operator<<(ostream&, Person  p)
{
   cout << "\nName: " << p.name << "\n Alter: "
        <<  p.alter;
}
```