Inf II

Pointer

```
datentyp * Zeigervariable;        Zeigervariable = & Variable;
                                  Zeigervariable = & Feldvariable [feldindex];

Zeigervariable [index];
Zeigervariable + index;           Zeigervariable - index;

Zeigervariable -> Strukturvariable;
```

Speicherplatz dynamisch allokieren

C#: `Zeigervariable = new datentyp;` `Zeigervariable = new datentyp [feldgröße];`

```
delete (Zeigervariable);
```

C: `Zeigervariable = (datentyp *) malloc (Anzahl * sizeof (datentyp));`

```
free (Zeigervariable);
```

Cast-Anweisung

```
datentyp1 Variable1;  datentyp2 Variable2;   Variable2 = (datentyp2) Variable1;
```

Strukturen / Objekte

```
typedef struct { Variablenliste } Typname;
```

Instanz: `Instanz1 Variable1;` `Variable1.a = Zahl;` `Variable1.b = Zahl;`

C-Präprozessor

```
#include "Deklarationsfile"
#define TEXTERSATZ          #undef TEXTERSATZ
```

Verhinderung des Doppelaufrufs von Headerfiles

```
#if !defined (Headerfilename_h)
#define Headerfilename_h
...
#endif
```

Headerfile "name.h"

```
class Name {
private: datentyp1 Name1 (...);
public: datentyp2 Name2(...);  Name (...);  ~Name();
};
```

Sourcefile "name.cpp"

```
datentyp Name :: Name2 { return (...) }
```

Sourcefile "main.cpp"

```
Instanz1. funktion;   Instanz2. variable;
```

Mehrfachvererbung: `Instanz :: Klasse :: variable;`

Konstruktor ^ Destruktor

```
Name :: Name (...) { ... }
Name :: ~Name () { ... }
```

Vererbung

```
class Basis { ... };
class Abgeleitet : public Basis { ... };
```

Mehrfachvererbung

```
class Basis1 {...};  class Basis2 {...};
class Abgeleitet : public Basis1, public Basis2 {...};
```

Virtuelle Funktion

Basisklasse: `virtual datentyp Funktion();` abgel.: `datentyp Funktion();`

Abstrakte Funktion

```
virtual datentyp Funktion() = 0;
```

friend-Funktion

```
public: friend Rückgabetyp Funktionsname ( Übergabeargumente );
```

this-Pointer

```
cout << this;
```

Inf 11

Überladen von Operatoren

```
return_typ Klassen_name:: operator #( typ2 Operand2) {...}
```

oder

```
return_typ operator #( typ1 Operand1, typ2 Operand2) {...}
```

Cklasse : `CKlasse CKlasse :: operator * ( CKlasse &var);`
main() : `Variable = Instanz1 * Instanz2;` ^ `Variable = Instanz1. operator *(Instanz2);`

Cklasse : `CKlasse Funktionsname ( CKlasse instanz1, CKlasse instanz2);`
main() : `instanz3. Funktionsname ( instanz1, instanz2);`

„fstream.h"

Anlegen einer Fileinstanz + Verbinden des Streams mit der Datei auf der Festplatte

```
fstream Dateiverweis;
Dateiverweis.open("test.txt", ios::out);
```

```
fstream Dateiverweis("test.txt", ios::out);
```

Schließen der Datei

```
Dateiverweis.close();
```

Lesen von Dateien

```
ifstream Dateiverweis;
Dateiverweis.open("test.dat", ios::in | ios::nocreate, 0);
```

Fehlerabfrage

```
if(!Dateiverweis) { Fehlermeldung }
```

Lesen + Schreiben von binären Dateien

```
ifstream Lesen ("Dateiname", ios::in | ios::binary);
ofstream Schreiben("Dateiname", ios::out | ios::binary);
Lesen.read( char &ch, anzahl );
Schreiben.write( char ch, anzahl );
```