Referenzen

- **Ersatzbezeichner** (**Aliasname**, symbolischer Name) für **bereits existierenden** Speicherplatz einer Variablen bzw. eines Objektes

```
/* C  */                          // C++
int i=0;                          int i = 0;
#define  alias   i                int  &alias = i;    // alias ist Referenz, mit i initialisiert
alias = 2;  /* i = 2;  */         alias = 2;          // i = 2;  //im Ausdruck ohne &
```

- Referenzen müssen bereits bei der Deklaration mit einer Variablen oder einem Objekt des **gleichen Typs initialisiert** werden, **int &alias;** oder **double d; int &refd = d;** sind **falsch** !

- Alle Operationen mit der **Referenz** werden auch mit der **Variablen** ausgeführt, die die Referenz initialisiert hat (s.o.)

- Eine **Referenz** ist über die sie initialisierende Variable während der **gesamten Lebensdauer** an den Speicherplatz der Variablen gebunden. Eine **Freigabe** oder **Neuzuordnung** seitens einer anderen Variablen ist **nicht** möglich.

- **int k = 4; int &alias = k; alias++;** cout<<"k = "<< **k** <<endl; // liefert **k = 5**

Referenzen

- Eine Variable kann **mehrere Referenzen** besitzen, **Referenzen** können seitens **bestehender Referenzen** initialisiert werden (vgl. refdem0.cpp)

- **Referenzen** können für **Felder** und **Feldelemente** verwendet werden (vgl. refdem10)

- Der **Kontext** definiert die Bedeutung von ***** und **&** (vgl. refdem12)

```
                          //Referenz    //Zeiger       //Adressoperator
void main(){int ivar = 2, &iref = ivar, * const iptr = &iref;
            cout<<"ivar = "<< (*iptr)++ <<" iref = "<< ++iref <<endl;
}                            //Dereferenzierung
```

- Wenn eine Variable bzw. ein Objekt über einen **Referenzparameter** an eine Funktion übergeben wird, dann wird nur die **Adresse** dieser Variablen bzw. dieses Objektes **übergeben**, **ohne** daß innerhalb der Funktion die Parametervariable als **Zeigervariable** behandelt wird. D.h. im Gegensatz zu **Zeigerparametern** muß der Parameter **nicht** mittels ***** bzw. **--> dereferenziert** werden.

- Im Gegensatz zu **Zeiger**- oder **Wertparametern entfällt** bei **Referenzparametern** die Erzeugung einer lokalen **Parameterkopie**, bei Objekten wird der **Kopierkonstruktor nicht** gerufen.

Referenzen

- Der **Referenzparameter** greift **direkt** auf die **Variable** der Aufrufliste zu (vgl. refdem5)

- **Referenzparameter** mit **const** können seitens der Funktion **nicht** verändert werden (vgl. refdem6, refdem4, refdem2)

- **Referenzen als Funktionstypen** müssen Variablen oder Objekte sein, die auch **außerhalb der Funktion** gültig sind (vgl. refdem7, higher, refdem8).

- Das können **globale Variablen**, lokale **static-Variablen** oder **Variablen** sein, die als **Referenz-parameter** an die Methode übergeben wurden. Auch mit **new** angelegte Objekte können zur **Initialisierung** einer **Referenz** verwendet werden (vgl. ohmref)
  Beispiele: int *& **func**(int *&s){ *s += 5; return s; } //s ist Referenz auf Zeiger auf int
  int &**func**(){ int *s = new int(5); return *s; } //*s ist der dynamische Speicherplatz

- **Lokal** in einer Funktion **vereinbarte Variablen** dürfen **nicht** mit **return** zurückgegeben werden, wenn der Funktionstyp eine **Referenz** ist.

- **Referenzen** von **dynamisch** instantiierten Objekten sind **äquivalent** zu **Zeigern** auf **dynamisch** instant. Objekte für die **späte Bindung** von **virtual-Methoden** die wesentliche Voraussetzung