10. Abend

Ziel :

- Nach diesem Abend werden wir alias-namen, sogenannte Referenzen auf Variablen erzeugen und anwenden können. Wir sind auch in der Lage die Speicheradresse von Variablen herauszufinden und uns diese in einer Zeigervariable zu speichern.

Referenzen

Beim erzeugen einer Variablen belegen wir einen gewissen Teil des Speichers und schreiben einen Wert an diese Speicherstelle indem wir der Variable einen Wert geben. Beim erzeugen einer neuen Variable belegen wir neuen Speicherplatz, auch wenn wir der neuen Variablen den Wert der alten Variablen zuweisen !

```
using namespace std;

int main()
{
    // Variable a erzeugen und einen Wert geben
    long a = 0;

    // neue Variable erzeugen und ihr den Wert
    // von a geben
    long b = a;

    // neuer Wert für a
    a = 10;

    cout << "a hat den Wert : " << a << endl;

    cout << "b hat den Wert : " << b << endl;

    return 0;
}
```

Mit einer Referenz können wir auf eine bereits existierende Variable zugreifen, dieser aber einen anderen Namen geben !
Eine Referenz definiert man indem man einen Datentypen wählt und ein **&** hinzufügt: Um also in unserem Beispiel von oben b als Referenz von a zu definieren schreiben wir folgendes :

```
int main()
{
    // Variable a erzeugen und einen Wert geben
    long a = 0;

    // b ist jetzt nur ein anderer Name für
    // a !
    long& b = a;

    // neuer Wert für a
    a = 10;

    cout << "a hat den Wert : " << a << endl;

    cout << "b hat den Wert : " << b << endl;

    return 0;
}
```

Mit b "refererenzieren" wir also a. Beachte, dass der Datentyp der Referenz der gleiche Datentyp sein muss wie der Datentyp der referenzierten Variable. Und beachte auch : **Referenzen müssen initialisiert werden** ! Das heisst folgender Code compiliert nicht und ist darum rot (nur WEB).

```
    long a = 0;
    long& c;
    // falsch, Referenzen müssen initialisiert werden
    c = a;
```

Über eine Referenz kann also eine vorher definierte Variable mit einem anderen Namen angesprochen und auch verändert werden. Soll eine Referenz nur verwendet werden um eine bereits bestehende Variable auszulesen, diese aber nicht zu ändern verwenden wir eine konstante Referenz.

```
int main()
{
    // Variable a erzeugen und einen Wert geben
    double z = 4.5;

    // b ist jetzt nur ein anderer Name für
    // a ! b kann aber nur gelesen werden !
    const double& b = a;
```

```
    // neuer Wert für a
    a = 14.567;

    cout << "a hat den Wert : " << a << endl;

    cout << "b hat den Wert : " << b << endl;

    return 0;
}
```

Funktionsaufrufe mit und ohne Referenzen

Die Bedeutung von Referenzen und nicht-Referenzen zeigt sich bei Funktionsaufrufen.
Betrachten wir eine ganz einfache Funktion “test”, die keinen Rückgaberwert hat (also void) und einen Parameter vom Datentyp long hat.

```
void test(long z)
{
    z++;
    cout << z << endl;
}
```

Beim Aufruf einer solchen Funktion geschieht folgendes : Der Compiler erzeugt für die Funktion eine neue Variable mit dem Namen z und gibt dieser den Wert der beim Funktionsaufruf als Parameter übergeben wird.

```
void test(long z)
{
    z++;
    cout << z << endl;
}

int main()
{
    // Variable a erzeugen und einen Wert geben
    long a = 0;
    // Funktion test aufrufen mit a als Parameter
    test(a);

    cout << "a hat den Wert : " << a << endl;

    return 0;
}
```

Versuchen wir ungefähr niederzuschreiben was der Compiler für Code erzeugt :

long a = 0;

// **test:**
long z = a;
z++;
cout << z << endl;

cout << "a hat den Wert : " << a << endl;

Die ursprüngliche Variable a bleibt also unverändert, denn beim Funktionsaufruf wird eine Kopie der Variable a mit dem Namen z erzeugt. Wollen wir eine Funktion, die unsere Variable a wirklich verändert brauchen wir so etwas :

long a = 0;

// **test:**
long& z = a; // hier Referenz !
z++;
cout << z << endl;

cout << "a hat den Wert : " << a << endl;

Das können wir jetzt einfach wieder als sauberes C++ so niederschreiben :

```
// Funktion test jetzt mit Referenzvoid test(long& z)
{
    z++;
    cout << z << endl;
}

int main()
{
    // Variable a erzeugen und einen Wert geben
    long a = 0;
    // Funktion test aufrufen mit a als Parameter
    test(a);

    cout << "a hat den Wert : " << a << endl;

    return 0;
}
```

Im Debugger und auch im Konsolenfenster solltest du den Unterschied jetzt feststellen. Der Compiler erzeugt jetzt nicht eine neue Variable sondern greift auf die originale Variable zu, auch wenn mit einem anderen Namen.

Referenzen sind nicht nur nützlich um übergeben Werte zu ändern, sondern können helfen Code effizienter zu machen.
Betrachte folgenden Code mit einer zugegebenermassen absurden NamenTest-Funktion :

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

using namespace std;

bool NamenTest(string name)
{
    string testName = "Markus";

    if(testName == name)
    {
        return true;
    }
    else
    {
        return false;
    }
}

int main()
{
    string Benutzer;

    cout << "Gib deinen Namen ein : ";
    cin >> Benutzer;

    bool testOk = NamenTest(Benutzer);
    if(testOk)
    {
        cout << endl << "Guter Junge " << Benutzer << endl;
    }
    else
    {
       cout << endl << "Schlechter Junge " << Benutzer << endl;
    }

    return 0;
}
```

Wie wir wissen muss der string name in der Funktion neu erzeugt werden und mit dem gleichen Inhalt gefüllt werden wie der string, der der Funktion als Parameter übergeben wird. Je nach Länge des strings kann das bei häufigen Aufrufen ein Weile

dauern ! Durch einfaches ändern des Parameters auf eine string-Referenz können wir unser Programm effizienter gestalten.

```
bool NamenTest(string& name)
{
    string testName = "Markus";

    if(testName == name)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Da wir in unserem Beispiel den Namen in der Funktion nicht ändern wollen sondern nur effizienter übergeben wollen, verwenden wir in solchen Fällen eine konstante Referenz.

```
bool NamenTest(const string& name)
{
    string testName = "Markus";

    if(testName == name)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

Ihr werdet ab heute von mir häufig solchen Code zu sehen bekommen ! Wenn Euch nicht klar ist wieso ich gewisse Parameter als Referenzen, als konstante Referenzen oder "By Value" definiere fragt sofort danach ! Diesen Code hier oben werden wir in der Lektion noch gemeinsam analysieren und optimieren !

Zeiger

Gegner von C und von C++ führen häufig als Argument gegen diese wunderbare Programmiersprache ins Feld, dass man in C mit Zeigern herumhantieren kann und dadurch unsichere Programme schreiben kann. Sie haben recht. Ich habe selber schon kriminellen Code gesehen, der besonders durch unachtsame Anwendung von Zeigern zu einer Zeitbombe geworden ist.

Trotzdem, man kann Zeiger sorgfältig anwenden und hat mit ihnen ein weiteres Mittel gute Designs in C++ zu verwirklichen (Amen).

Erstes Beispiel für einen Zeiger:

```
int main()
{
    int test = 2;
    // Speicher anlegen für eine int - Variable
    // und mit 2 initialisieren

    int* zeigerAufTest = &test;
    // mit dem Adress-of operator & finden
    // wir die Adresse und weisen sie unserer
    // Variable zeigerAufTest zu.

    return 0;
}
```

Das Anlegen einer Variable - hier der Variable test - führt dazu, dass zur Laufzeit Speicher für diese Variable angelegt wird. In unserer Umgebung werden für eine int - Variable 4 Bytes Speicher reserviert. Die Variable zeigerAufTest ist ein Zeiger auf so eine Speicherstelle.

Zur Laufzeit können wir den Inhalt der Variablen mit Hilfe des Debuggers genau ansehen.

siehe http://www.devmentor.ch (etc.) für Bilder !

Ergänztes Beispiel

Beim nächsten Beispiel, das sich kaum vom ersten unterscheidet weisen wir über einen Zeiger der Variablen einen neuen Wert zu.

```
int main()
{
    int test = 1;
    // Speicher anlegen für eine int - Variable
    // und mit 1 initialisieren

    int* zeigerAufTest = &test;
    // mit dem Adress-of operator & finden
    // wir die Adresse und weisen sie unserer
    // Variable zeigerAufTest zu.

    int test2 = *zeigerAufTest;
    // Die Variable test2 soll den Wert erhalten
    // der an der Speicherstelle steht, auf die
    // zeigerAufTest zeigt.

    *zeigerAufTest = 3;
    // Mit diesem Aufruf ändern wir die
    // Speicherstelle auf die zeigerAufTest zeigt.
    // das heisst die Variable test, ABER nicht test2.

    return 0;
}
```

Wichtig ist, dass wir durch die Zuweisung auf der zweitletzten Zeile direkt die Variable test ändern, nicht aber die Variable test2. Die Variable test2 wird neu erzeugt Und erhält auch einen eigenen Speicherplatz. Sie wird jedoch mit dem Wert initialisiert der im Speicher steht, wo zeigerAufTest hin zeigt.

Funktionsaufrufe mit Zeigern

Auch mit dem Zeiger haben wir die Möglichkeit die Funktion Test von oben so umzuschreiben, dass sie sich ähnlich wie mit der Referenz verhält. Wir können nämlich anstatt einen long als Parameter einen Zeiger auf long (long*) als Parameter verwenden.

```
// Funktion test jetzt mit Zeiger
void test(long* z)
{
    // Wir müssen z dereferenzieren
    (*z)++;
    cout << *z << endl;
}
```

```
int main()
{
    // Variable a erzeugen und einen Wert geben
    long a = 0;
    // Funktion test aufrufen mit Adresse von
    // a als Parameter
    test(&a);

    cout << "a hat den Wert : " << a << endl;

    return 0;
}
```

Im Gegensatz zum Aufruf mit Referenzen muss aber diesmal der aufrufende Code so umgeschrieben werden, dass der Funktion wirklich eine Adresse übergeben wird ! Zeiger müssen initialisiert werden. Sie zeigen meistens auf Speicher, der uns nicht gehört !
Im folgenden Beispiel sehen wir einen Fehler, den "Anfänger" (und auch andere) häufig machen.

Das Problem ist, dass ein Zeiger irgendwohin in den Speicher zeigen kann und häufig nicht auf Speicher, der unserem Programm gehört. Im Programm unten legen wir eine Variable ZeigerAufEinZeichen als Zeiger auf eine char - Variable an, ohne aber eine char - Variable wirklich anzulegen ! Das Schreiben an die Speicherstelle, auf die ZeigerAufEinZeichen zeigt ist eine "Speicherschutzverletzung".

```
int main()
{
    char* ZeigerAufEinZeichen;
    // Zeiger - Variable anlegen, der auf ein
    // einzelnes char zeigt

    *ZeigerAufEinZeichen = 'A';     // Schreibe dorthin wo der
                                   //Zeiger im Speicher
    // hinzeigt das Zeichen A (ASCII 65)

    // Der Compiler sollte eine Warnung anzeigen,
    // da zeigerAufEinZeichen verwendet
    // wird ohne dass die Variable vernünftig initialisiert
    // wird.
    // Wenn wir das Programm laufen lassen, also zur Laufzeit,
    // stürzt das Programm sehr wahrscheinlich ab,
    // da wir an eine Speicherstelle schreiben wollen,
    // die uns nicht gehört.
    return 0;
}
```

Korrigiert sieht das Beispiel so aus :

```
int main()
 {
    // Korrekt

    char* zeigerAufEinZeichen = 0;
    // Zeiger - Variable anlegen, der auf ein
    // einzelnes char zeigen kann und auf 0 initialisieren

    char einZeichen = 'A';      // Variable anlegen, die ein
Zeichen enthält     // und mit 'A' initialisiert wird.

    zeigerAufEinZeichen = &einZeichen;
    // Der Zeiger zeigt jetzt an die Adresse
    // von einZeichen (Adress- operator &)

    *zeigerAufEinZeichen = 'B';
    // Schreibe dorthin wo der Zeiger im Speicher
    // hinzeigt das Zeichen B (ASCII 66);

    return 0;
}
```