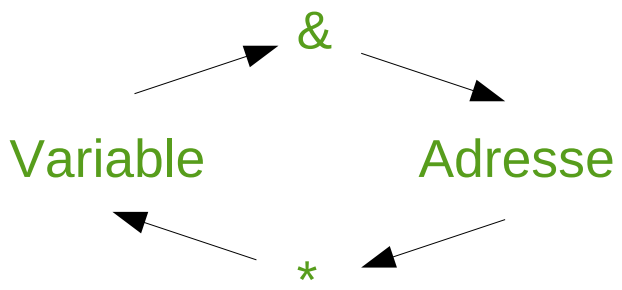
# Zeiger

Variable ist ein „Speicherort“ im C++ Programm und hat eine Adresse.
- Zugriff auf Variable und/oder Adresse




- & Operator liefert die Adresse einer Variablen im Speicher

```
int myVariable = 10 ;
&myVariable ;
```
Speicheradresse von myVariable

- Zeiger ist eine Variable deren Wert eine Adresse enthält. Zeiger werden im Programm definiert

```
Type *PointerName ;
```

```
int    *pMyInteger ;
double *pMyDouble ;
char   *pMyCharacter ;
```

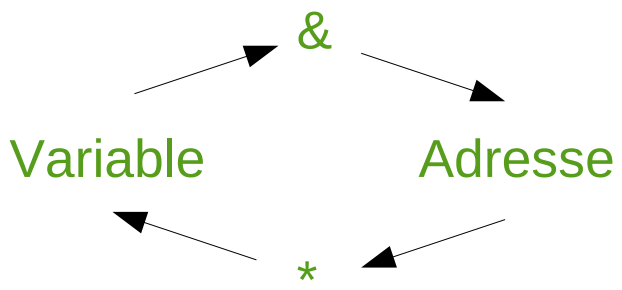
Datentyp eines Zeiger ist für alle Variablentypen eine hex Zahl

- Vorsicht beim Deklarieren von pointern

```
int * p1, p2
```
deklariert eine integer pointer variable und eine integer variable !

# Zeiger

Variable ist ein „Speicherort“ im C++ Programm und hat eine Adresse.
- Zugriff auf Adresse und Variable




- * Operator ist eine häufige Anwendung in C++ Programmen und hat zwei unterschiedliche Bedeutungen
  - Pointer Variablen Definition mit Typ Angabe
  - Dereferenz Operator im Quellcode in Form von
    value pointed to by (Wert auf den gezeigt wird)

- * Operator als Dereferenz im code

```
int MyVariableA = 10 ;
int *pMyVariableA ;
int MyVariableB ;

pMyVariableA = &MyVariableA;
MyVariableB = *pMyVariableA;
```

Adresse der Variablen `MyVariableA`

Wert auf den gezeigt wird

`MyVariableB` ist jetzt 10

# Zeiger

- Zuweisungen und Rechnen mit Zeigern

```
int MyVariableA = 10 ;
int MyVariableB = 20 ;
int *pMyVariableA , *pMyVariableB ;

pMyVariableA = &MyVariableA ;
pMyVariableB = &MyVariableB ;

*pMyVariableA = 100;
```

Value pointed to by pMyVariableA wird auf 100 gesetzt

```
*pMyVariableA = *pMyVariableB;
```

Value pointed to by pMyVariableB = value pointed to by pMyVariableA

```
pMyVariableA = pMyVariableB;
```

A und B haben die gleichen pointer

```
*pMyVariableA = 200 ;
```

Value pointed to by pMyVariableA = 200

# Zeiger und Arrays

- In der Deklaration einer array Variablen ist der array Name der Zeiger auf das erste array Element

```
const int nLength = 50 ;
double MyArray [nLength] ;
MyArray   is equivalent to   &MyArray[0]
```

```
double *pMyArray ;
pMyArray = MyArray ;
*pMyArray = 500;
```

wird häufig so verwendet

setzt das erste array Element auf 500

```
pMyArray++ ;
*pMyArray = 1000;
```

läuft durch die array Elemente

setzt das zweite array Element auf 1000

```
pMyArray = MyArray + 2;
*pMyArray = 3000;
```

setzt den pointer auf Element 2

setzt das dritte array Element auf 3000

```
pMyArray = MyArray;
*(pMyArray + 3) = 2;
```

setzt den pointer auf Element 0

setzt das vierte array Element auf 2

```
MyArray[4] = 10000;
```

setzt das fünfte array Element auf 10000

# Zeiger und Arrays

- Komplexe Zeiger Konstruktionen
  Das Zeigerkonstrukt wird von innen nach aussen aufgelöst.

```
int n = 50 , m = 100 ;
int *nptr;                          int Zeiger
int **npptr;                        Zeiger auf einen int Zeiger
int *ptrArray[2];                   Array von int Zeigern
nptr = &n ;                         Adresse von n
npptr = &nptr ;                     Adresse vom Zeiger auf  n
ptrArray[0] = &n;
ptrArray[1] = &m;                   Adressen von m und n
…...
**(&ptrArray[0])                    = 50
**npptr                             = 50
…...
```

pointer.cc

```cpp
// Demonstrate usage of pointer

#include <iostream>
using namespace std;

int main()

{

  int MyInteger = 20 ;
  const int nDim = 3 ;
  double MyArray[nDim] = {1.5,2.4,3.9};

  int *pMyInteger;
  double *pMyArray;

  cout << "Adresse der Zahl " <<  MyInteger << ": " << &MyInteger << endl;
  cout << "Adresse der Zahl " <<  MyArray[0] << ": " << &MyArray << endl;

  pMyInteger =  &MyInteger ;

  cout << "Value of MyInteger variable: " << MyInteger << endl;

  cout << "Pointer to MyInteger variable: " << pMyInteger << endl;

  cout << "Value of  *pMyInteger: " << *pMyInteger << endl;

  return 0 ;

}
```

```
// Demonstrate usage of pointer

#include <iostream>
using namespace std;

int main()

{

  int MyInteger = 20
  const int nDim = 3
  double MyArray[nDim] = {1.5,2.4,3.9};

  int *pMyInteger;
  double *pMyArray;

  cout << "Adresse der Zahl " <<  MyInteger << ": " << &MyInteger << endl;
  cout << "Adresse der Zahl " <<  MyArray[0] << ": " << &MyArray << endl;

  pMyInteger =  &MyInteger ;

  cout << "Value of MyInteger variable: " << MyInteger << endl;

  cout << "Pointer to MyInteger variable: " << pMyInteger << endl;

  cout << "Value of  *pMyInteger: " << *pMyInteger << endl;

  return 0 ;

}
```

Output:

```
Adresse der Zahl 20: 0x7fffaeed5efc
Adresse der Zahl 1.5: 0x7fffaeed5ee0
Value of MyInteger variable: 20
Pointer to MyInteger variable: 0x7fffaeed5efc
Value of  *pMyInteger: 20
```

```
// Demonstrate usage of pointer

#include <iostream>
using namespace std;

int main()

{

  int MyInteger = 20
  const int nDim = 3
  double MyArray[nDim] = {1.5,2.4,3.9};

  int *pMyInteger;
  double *pMyArray;

  cout << "Adresse der Zahl " <<  MyInteger << ": " << &MyInteger << endl;
  cout << "Adresse der Zahl " <<  MyArray[0] << ": " << &MyArray << endl;

  pMyIn

  cout

  cout

  cout

  retur

}
```

Output:

```
Adresse der Zahl 20: 0x7fffaeed5efc
Adresse der Zahl 1.5: 0x7fffaeed5ee0
Value of MyInteger variable: 20
Pointer to MyInteger variable: 0x7fffaeed5efc
Value of  *pMyInteger: 20
```

Arbeitsvorschlag:

- modifizieren Sie das Programm so, das im Code der * Operator als Dereferenzierung verwendet wird. Siehe letztes cout statement.
- schreiben Sie ein Programm in dem die verschiedenen Methoden zum Setzen von array Elementen verwendet werden.

- schreiben Sie ein Programm, das aus dem character array die Positionen der Buchstaben o ausgibt.

```
const char * s = “Hello kids, you are too lo
```

charPointer.cc

# Dynamischer Speicher

```
const int maxTage = 24 * n ;
double Temperatur[maxTage]={0.};
```

Fehler: Arraygrenze dynamisch

Mit den C++ Operatoren `new` und `delete` lässt sich dynamisch (während der Laufzeit des Programmes) Speicher allokieren.
Die Operatoren können auf beliebige Objekte angewandt werden.

- Syntax

```
Datentyp * Name =  new Datentyp [ Anzahl ];
delete [] Name ;
```

Bei jeder Verwendung von `new` sollte man überlegen wie der Speicher wieder freigegeben werden soll.

```
…….
int N , K , J;
…….
N = K*J ;
double* DynamicArray = new double [N];
DynamicArray[K+1] = (double) ((K+1)*(K+1));
……..
delete [] DynamicArray;
```

Erzeugung eines Arrays mit einer Länge, die während der Laufzeit im Programm festgelegt wird.

Erzeugung des Arrays mit N Elementen

Verwendung wie jedes Array

Speicherfreigabe

# Dynamischer Speicher

```
const int maxTage = 24 * n ;
double Temperatur[maxTage]={0.};
```

Fehler: Arraygrenze dynamisch

Mit den C++ Operatoren `new` und `delete` lässt sich dynamisch (während der Laufzeit des Programmes) Speicher allokieren.
Die Operatoren können auf beliebige Objekte angewandt werden.

- Syntax

```
Datentyp * Name =  new Datentyp [ Anzahl ];
delete [] Name ;
```

Bei jeder Verwendung von `new` sollte man überlegen wie der Speicher wieder freigegeben werden soll.

Arbeitsvorschlag:
- Schreiben Sie ein Programm, ausgehend von dynamicArray_0.cc , das die Elementnummer in eine Tabelle mit 4 Spalten und 10 Zeilen schreibt. Die Tabelle soll als ein eindimensionales `double` array implementiert werden, das dynamisch allokiert wird. Es soll zeilenweise durch die Tabelle gegangen werden. Drucken Sie Elementnummer und Inhalt. Verwenden Sie den Programmnamen dynamicArray.cc

dynamicArray.cc