Themen

- Sichtbarkeit von Variablen
- einfache Typumwandlung (Typecasting)
- logische Operatoren
- **enum** Datentyp
- **switch** Anweisung

Globale Variablen

- **globale** Variablen werden ausserhalb von Funktionen oder Methoden deklariert

```
const double PI=3.14159;
```

```
void test1() {  
    std::cout << PI;  
}
```

```
void test2() {  
    std::cout << PI;  
}
```

Lokale Variablen

- **lokale** Variablen sind innerhalb von Funktionen oder Methoden deklariert

```
void test1() {  
    const double PI=3.14159;  
    std::cout << PI;  
}
```

```
void test2() {  
    std::cout << PI;    // ERROR  
}
```

Sichtbarkeit von Variablen

- globale Variablen sind überall sichtbar
- lokale Variablen sind lediglich innerhalb des Codeblocks in dem sie deklariert wurden sichtbar

Lokale Variablen - Beispiel 1

```
• void test1()  
{  
    const double PI=3.14159;  
    std::cout << PI;  
    for(int i=0;i<10;++i)  
    {  
        std::cout << PI;  
    }  
    std::cout << PI;  
}
```

Lokale Variablen - Beispiel 2

```
• void test1()  
{  
    for(int i=0;i<10;++i)  
    {  
        const double PI=3.14159;  
        std::cout << PI;  
    }  
    std::cout << PI; // ERROR  
}
```

Variablen Beispiel

```
unsigned int a=4;
```

```
int main()
```

```
{
```

```
    float a=3.3f;
```

```
    for(int a=0 ; a<3 ; ++a )
```

```
    {
```

```
        double a=34.5;
```

```
    }
```

```
}
```

Variablen Beispiel

```
unsigned int a=4;
```

```
int main()
```

```
{
```

```
    float a=3.3f;
```

```
    for(int a=0 ; a<3 ; ++a )
```

```
    {
```

```
        double a=34.5;
```

```
    }
```

```
    char a='a';
```

```
}
```

Implizite Typumwandlung 1/2

```
int c=true; // c = 1
```

```
char a=3.5; // a = 51
```

Implizite Typumwandlung 2/2

Benutzt man einen Wert oder eine Variable vom Typ X an einer Stelle, an der ein Wert vom Typ Y erwartet wird, versucht der Compiler, den Wert in einen Wert vom Typ X umzuwandeln.

Beispiele für die implizite Typumwandlung nach int

```
int a = 23.5;           // a=23
int b = 3.12e24;       //b=2147483647
int c = true;          // c=1
int d = false;         // c=0
unsigned int e = -5;   //e=4294967291
unsigned int f = true; // f=1
unsigned int g = false; // g=0
```

Beispiele für die implizite Typumwandlung nach float

```
float a = 4;           // a=4.0f
```

```
float b = 2.0e300;    // b=inf
```

```
float c = true;       // c=1.0f
```

```
float d = false;      // d=0.0f
```

Beispiele für die implizite Typumwandlung nach bool

```
bool a = 23;           // a=true
bool b = 1.23;        // b=true
bool c = 0;           // c=false
bool d = 0.0;        // d=false
```

Regeln für die implizite Typumwandlung 1/3

- Kein Problem, wenn man Daten eines Typs mit kleinem Umfang, einem entsprechenden Typ mit grossem Umfang zuweist

```
int a=5;  
long b=a;
```

```
int a=-5;  
unsigned long b=a; // VORSICHT!!!
```

Regeln für die implizite Typumwandlung 2/3

- Möglicher Verlust von Genauigkeit bei Zuweisung von Gleitkommazahlen an einen Integer Typ

```
float a=3.14159;  
int b=a;           // b=3
```

Regeln für die implizite Typumwandlung 3/3

- Möglicherweise falsches Ergebnis bei Zuweisung von einem Typ mit grossem Umfang auf einen Typ mit kleinerem Umfang

```
int a=513;  
char b=a;    // b=1
```

```
float a=3e6;  
short b=a;   // b=-14656
```

Überladen von Funktionen

In einem Programm dürfen keine zwei Funktionen mit derselben Signatur vorkommen, es dürfen aber Funktionen vorkommen, die den selben Namen haben, sich jedoch in ihrer Argumentenliste unterscheiden.

Dies bezeichnet man als Überladen der Funktion.

Beispiel für überladene Funktionen

```
int add(int x, int y);
```

```
double add(double x, double y);
```

```
double add(int x, int y); // verboten
```

Überladene Funktionen und implizite Typumwandlung 1/2

```
int add(int x,int y) ;
```

```
int a=5;
```

```
double b=2.5;
```

```
add(a,a) ;
```

```
add(b,b) ;
```

```
add(b,a) ;
```

Überladene Funktionen und implizite Typumwandlung 2/2

```
int add(int x,int y);  
double add(double x,double y);
```

```
int a=5;  
double b=2.5;
```

```
add(a,a);  
add(b,b);  
add(b,a);
```

Explizite Umwandlung einfacher Datentypen (C-cast)

- um dem Compiler explizit zu sagen, dass er einen Datentyp in einen bestimmten anderen Datentyp umwandeln soll, benutzt man die explizite Typumwandlung (Casting)
- Dem umzuwandelnden Wert oder der umzuwandelnden Variablen wird der Datentyp, in den umgewandelt werden soll, in Runden Klammern vorangestellt.

Explizite Typumwandlung

Beispiel 1

```
int add(int x,int y);  
double add(double x,double y);
```

```
int a=5;  
double b=2.5;
```

```
add( (int)b , a );  
add( b , (double)a );
```

Explizite Typumwandlung

Beispiel 2

```
for (unsigned char i=0;i<10;++i)
{
    std::cout << i << "\n";
}
```

Explizite Typumwandlung

Beispiel 2

```
for (unsigned char i=0;i<10;++i)
{
    std::cout << (int)i << "\n";
}
```

Logische Operatoren

- zählen zu den binären Operatoren
- verknüpfen zwei `bool` Werte miteinander
- liefern einen `bool` Wert zurück
- logisches Und `&&`
- logisches Oder `||`

Beispiele zu logischen Operatoren

```
int a=34;  
int b=2;
```

```
if( a > 4 && b < 5 ) ...
```

```
if( a > 4 || b < 5 ) ...
```

Beispiele zu logischen Operatoren

```
int a=34;  
int b=2;
```

```
if( (a > 4) && (b < 5) ) ...
```

```
if( (a > 4) || (b < 5) ) ...
```

logische Operatoren Beispiel 2

```
bool istEineZahl(char z1, char z2)
{
    if (z1>47 && z1<58 && z2>47 && z2<58)
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

logische Operatoren Beispiel 2

```
bool istEineZahl(char z1, char z2)
{
    if(
        (z1>47 && z1<58) &&
        ( z2==0 || (z2>47 && z2<58) )
    )
        ...
}
```

logische Operatoren Beispiel 2

```
bool istEineZahl(char z1, char z2)
{
    return (
        (z1>47 && z1<58) &&
        (z2==0 || (z2>47 && z2<58))
    );
}
```

enum Deklaration 1/6

- benutzerdefinierter Ganzzahldatentyp (Aufzählungstyp)
- z.B.

```
enum WeekendDay =  
{  
    Saturday,  
    Sunday  
};
```

```
WeekendDay day = Sunday;
```

enum Deklaration 2/6

```
• enum Wochentag =  
{  
    Sonntag,           // 0  
    Montag,            // 1  
    Dienstag,         // 2  
    Mittwoch,          // 3  
    Donnerstag,        // 4  
    Freitag,           // 5  
    Samstag            // 6  
};
```

enum Deklaration 3/6

```
• enum Wochentag =  
{  
    Sonntag=1,           // 1  
    • Montag,           // 2  
    • Dienstag,        // 3  
    • Mittwoch,         // 4  
    • Donnerstag,       // 5  
    • Freitag,          // 6  
    • Samstag           // 7  
};
```

enum Deklaration 4/6

```
• enum Wochentag =  
{  
    Sonntag=1,           // 1  
    Montag=3,           // 3  
    Dienstag=5,        // 5  
    Mittwoch=7,         // 7  
    Donnerstag=9,       // 9  
    Freitag=11,         // 11  
    Samstag=13          // 13  
};
```

enum Deklaration 5/6

```
• enum Wochentag =  
{  
    Sonntag=1,           // 1  
    Montag,             // 2  
    Dienstag=23,       // 23  
    Mittwoch,          // 24  
    Donnerstag=111,     // 111  
    Freitag,           // 112  
    Samstag            // 113  
};
```

enum Deklaration 6/6

```
• enum Wochentag =  
{  
    Sonntag=1,           // 1  
    Montag,             // 2  
    Dienstag=23,       // 23  
    Mittwoch,          // 24  
    Donnerstag=4,       // 4  
    Freitag,           // 5  
    Samstag             // 6  
};
```

enum – Beispiel 1

```
Wochentag tag=Donnerstag;
```

```
Wochentag tag=1; // Fehler
```

```
Wochentag tag=(Wochentag) 1;
```

```
unsigned int zahl=Donnerstag;
```

```
Wochentag tag=(Wochentag) 45; // !!!
```

enum – Beispiel 2

Verschiedene Aufzählungstypen sind typfremd, auch wenn sich ihre Wertebereiche überschneiden:

```
enum Farbe{Rot, Gruen, Blau};    // Werte 0,1,2  
enum Obst{Apfel, Banane, Birne}; // Werte 0,1,2
```

```
Farbe farbe;  
Obst frucht = Banane;
```

```
farbe = frucht; // ERROR
```

farbe und frucht haben verschiedene Datentypen, deren Werte einander **nicht** zugewiesen werden können!!!

switch Statement

- verzweigt das Programm aufgrund des Wertes eines Integer Datentyps (meist **enum**)
- kann durch verschachtelte **if then else** Anweisungen ersetzt werden

switch – Beispiel 1

```
enum Color {red,green,blue};
Color color=red;
switch(color)
{
    case red:
        std::cout << "red\n";
        break;
    case blue:
        std::cout << "blue\n";
        break;
    case green:
        std::cout << "green\n";
        break;
}
```

if instead of switch

```
if (color==red)
{
}
else if (color==blue)
{
}
else if (color==green)
{
}
```

switch - Beispiel 2

```
enum Color {red,green,blue};
Color color=red;
switch(color)
{
    case red:
        std::cout << "red\n";
        break;
    default:
        std::cout << "not red\n";
        break;
}
```

Schlechtes switch Beispiel

```
enum Color {red,green,blue};  
Color color=red;  
switch(color)  
{  
    case red:  
        std::cout << "red\n";  
        break;  
}
```

Gutes switch Beispiel

```
enum Color {red,green,blue};
Color color=red;
switch(color)
{
    case red:
        std::cout << "red\n";
        break;
    default:
        break;
}
```

switch - Beispiel 3

```
enum Color {red,green,blue};
Color color=red;
switch(color)
{
    case red:
    case green:
        std::cout << "nice color\n";
        break;
    default:
        std::cout << "ugly!!!\n";
        break;
}
```

switch - Beispiel 4

```
enum Color {red,green,blue};
Color color=red;
switch(color)
{
    case red:
        break;
    case green:
        std::cout << "nice color\n";
        break;
    default:
        std::cout << "ugly!!!\n";
        break;
}
```