Die Bytemühle - einfacher Zugriff auf einzelne Bytes einer Mehrbyte-Variablen

Manchmal besteht die Notwendigkeit auf einzelne Bytes einer Mehrbyte-Variablen (wie int, float, long, ect) zuzugreifen. Meist erledigt man das mittels Zeigern/Pointer. Dies ist aber nicht jedermanns Sache, Einsteigern fällt es oft schwer Zeiger zu verwenden und viele mögen die Operatoren * und & nicht. Deshalb möchte ich hier einen Weg über eine Hilfsvariable zeigen der vielleicht leichter zu verstehen ist. Funktioniert hat es bei mir mit der Arduino IDE 1.6.5 und einem ProMini. Deshalb sollte es auch auf einem UNO (und all seinen Brüdern mit dem 328er Controller) und dem MEGA2560 funktionieren.

Um auf die einzelnen Bytes einer Mehrbyte-Variablen leichter zugreifen zu können "basteln" wir uns eine Hilfsvariable aus einem selbst definierten Datentyp. Bei diesem Datentyp kann man dann leicht die 4-Byte Variable und die einzelnen Bytes ansprechen. Die Definition des neuen Datentyps, eine Union mit dem Namen "bytemuehle32", sieht so aus:

Code:

```
union bytemuehle32
{
   char           c[4];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
   unsigned char uc[4];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
   byte           b[4];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
   int8_t        i8[4];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
   uint8_t      ui8[4];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
   int            i[2];   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
   unsigned int  ui[2];   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
   word           w[2];   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
   short          s[2];   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
   int16_t      i16[2];   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
   uint16_t    ui16[2];   // 16 bit (2 byte) 0 bis 65,535 / 0 bis (2^16)-1)
   float             f;   // 32 bit (4 byte) -3.4028235E^-38 bis 3.4028235^+38
   long              l;   // 32 bit (4 byte) -2,147,483,648 bis 2,147,483,647 / -2^31 bis (2^31)-1)
   unsigned long    ul;   // 32 bit (4 byte) 0 to 4,294,967,295 / 0 bis (2^32)-1)
   double            d;   // 32 bit (4 byte) -3.4028235E^-38 bis 3.4028235^+38
   int32_t         i32;   // 32 bit (4 byte) -2,147,483,648 bis 2,147,483,647 / -2^31 bis (2^31)-1)
   uint32_t       ui32;   // 32 bit (4 byte) 0 to 4,294,967,295 / 0 bis (2^32) - 1)
};
```

Nachdem der neue Datentyp angelegt ist kann man ihn genau so verwenden wie alle bisher bekannten Datentypen. Jetzt können wir eine Hilfsvariable mit diesem neuen Datentyp anlegen.

Code:

```
bytemuehle32   universalvariable32 ;
```

Das Besondere an dieser Variablen (universalvariable32) ist, dass sie nur 4 Byte im Speicher belegt. Der Speicherbedarf der Hilfsvariablen richtet sich nach der größten Einzelvariablen welche in der Union definiert wurde. In 'universalvariable32' liegen jetzt alle in der Union definierten Variablen "übereinander", was bedeutet dass sie alle gleichzeitig vorhanden sind. Dadurch kann man jetzt auf alle in der Union definierten einzelnen Elemente direkt zugreifen. Die einzelnen Elemente werden dabei über variablenname.elementname angesprochen.
Nehmen wir an wir hätten in unserem Programm eine Variable vom Typ unsigned long, dem Namen 'startzeit' und Inhalt '0x89ABCDEF' und wollen mit Teilbytes dieser Variablen in unserem Programm weiter arbeiten. Dazu muss man den Inhalt von 'startzeit' passend in 'universalvariable32' kopieren. Sinnvollerweise kopiert man 'startzeit' also in die unsigned long Variable der Union, also nach 'ul'.

Code:

```
unsigned long  startzeit  =  0x89ABCDEF;
universalvariable32.ul    =  startzeit;
```

Jetzt ist `universalvariable32.ul = 0x89ABCDEF`
Wie die einzelnen Elemente der Union in den 4 Byte von 'universalvariable32' liegen, kann man folgender Tabelle entnehmen.

Lage der Elemente im RAM für union "bytemuehle32"

Little Endian

LSB ← → MSB			
Adr.+0	Adr.+1	Adr.+2	Adr.+3
c[0]	c[1]	c[2]	c[3]
uc[0]	uc[1]	uc[2]	uc[3]
b[0]	b[1]	b[2]	b[3]
i8[0]	i8[1]	i8[2]	i8[3]
ui8[0]	ui8[1]	ui8[2]	ui8[3]
i[0]		i[1]	
ui[0]		ui[1]	
w[0]		w[1]	
s[0]		s[1]	
i16[0]		i16[1]	
ui16[0]		ui16[1]	
f			
l			
ul			
d			
i32			
ui32			

Alle Elemente von 'universalvariable32' (Zeilen der Tabelle) haben den gleichen Inhalt, nämlich 0x89ABCDEF aus der obigen Zuweisung
universalvariable32.ul = startzeit;

Jetzt noch ein paar Beispiele als Programmcode die zeigen sollen wie man einzelne Elemente von 'universalvariable32' mit den Informationen aus der Tabelle lesen/beschreiben kann.

Code:

```
// Ausgabe der höherwertigen Hälfte von universalvariable32
Serial.println(universalvariable32.ui[1],HEX);
// Ausgabeergebnis --> 89AB

// Ausgabe der niederwertigen Hälfte von universalvariable32
Serial.println(universalvariable32.ui[0],HEX);
// Ausgabeergebnis --> CDEF

// Ausgabe der einzelnen Bytes von universalvariable32 aufsteigend
Serial.print(universalvariable32.b[0],HEX);
Serial.print(universalvariable32.b[1],HEX);
Serial.print(universalvariable32.b[2],HEX);
Serial.println(universalvariable32.b[3],HEX);
// Ausgabeergebnis --> EFCDAB89

// Ausgabe der einzelnen Bytes von universalvariable32 absteigend
Serial.print(universalvariable32.b[3],HEX);
Serial.print(universalvariable32.b[2],HEX);
Serial.print(universalvariable32.b[1],HEX);
Serial.println(universalvariable32.b[0],HEX);
// Ausgabeergebnis --> 89ABCDEF

// niederwertigste Stelle um 1 erhöhen und gesamt ausgeben
universalvariable32.b[0]++;
Serial.println(universalvariable32.ul,HEX);
// Ausgabeergebnis --> 89ABCDF0

// höchstwertige Stelle um 2 verringern und gesamt ausgeben
universalvariable32.b[3]-=2;
Serial.println(universalvariable32.ul,HEX);
// Ausgabeergebnis --> 87ABCDF0

// Länge der einzelnen Elemente von universalvariable32 anzeigen
Serial.println(sizeof (universalvariable32) );
// Ausgabeergebnis --> 4
Serial.println(sizeof (universalvariable32.f) );
// Ausgabeergebnis --> 4
Serial.println(sizeof (universalvariable32.ui32) );
// Ausgabeergebnis --> 4
```

```
Serial.println(sizeof (universalvariable32.i16[0]) );
// Ausgabeergebnis --> 2
Serial.println(sizeof (universalvariable32.w[1]) );
// Ausgabeergebnis --> 2
Serial.println(sizeof (universalvariable32.i8[3]) );
// Ausgabeergebnis --> 1
Serial.println(sizeof (universalvariable32.uc[2]) );
// Ausgabeergebnis --> 1

// eine float Variable im EEPROM ab Adr. 10 speichern
float testwert_w = 1234.56;
int   adrw       = 10;
universalvariable32.f = testwert_w;
EEPROM.write(adrw+0, universalvariable32.b[0]);
EEPROM.write(adrw+1, universalvariable32.b[1]);
EEPROM.write(adrw+2, universalvariable32.b[2]);
EEPROM.write(adrw+3, universalvariable32.b[3]);
// eine float Variable aus dem EEPROM ab Adr. 10 auslesen
float testwert_r;
int   adrr     = 10;
universalvariable32.b[0] = EEPROM.read(adrr+0);
universalvariable32.b[1] = EEPROM.read(adrr+1);
universalvariable32.b[2] = EEPROM.read(adrr+2);
universalvariable32.b[3] = EEPROM.read(adrr+3);
testwert_r = universalvariable32.f;

// Manche seriellen Übertragungsmethoden lassen nur die Übertragung einzelner Bytes zu.
// Will man damit z.B. eine float Variable übertragen, so muss man diese in einzelne Bytes zerlegen
// Beispiel I2C
//
// float testwert = 1234.56;
// universalvariable32.f = testwert;
// Wire.write(universalvariable32.b[0]);
// Wire.write(universalvariable32.b[1]);
// Wire.write(universalvariable32.b[2]);
// Wire.write(universalvariable32.b[3]);
```

Nicht immer hat man mit 4-Byte Variablen zu tun, deshalb jetzt noch eine Union für 2-Byte Variablen.

Code:

```
union bytemuehle16
{
char           c[2];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
unsigned char uc[2];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
byte           b[2];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
int8_t        i8[2];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
uint8_t      ui8[2];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
int               i;   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
unsigned int     ui;   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
word              w;   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
short             s;   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
int16_t         i16;   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
uint16_t       ui16;   // 16 bit (2 byte) 0 bis 65,535 / 0 bis (2^16)-1)
};
```

und für 8-Byte Variablen
Code:

```
union bytemuehle64
{
char                   c[8];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
unsigned char         uc[8];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
byte                   b[8];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
int8_t                i8[8];   //  8 bit (1 byte) -128 bis 127 / -2^7 bis (2^7)-1)
uint8_t              ui8[8];   //  8 bit (1 byte) 0 bis 255 / 0 bis (2^8)-1)
int                    i[4];   // 16 bit (2 byte) -32,768 to 32,767 / -2^15 bis (2^15)-1)
unsigned int          ui[4];   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
word                   w[4];   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
int16_t               i16[4];   // 16 bit (2 byte) -32,768 bis 32,767 / -2^15 bis (2^15)-1)
uint16_t             ui16[4];   // 16 bit (2 byte) 0 to 65,535 / 0 bis (2^16)-1)
float                  f[2];   // 32 bit (4 byte) -3.4028235E^-38 bis 3.4028235^+38
long                   l[2];   // 32 bit (4 byte) -2,147,483,648 bis 2,147,483,647 / -2^31 bis (2^31)-1)
unsigned long         ul[2];   // 32 bit (4 byte) 0 to 4,294,967,295 / 0 bis (2^32)-1)
double                 d[2];   // 32 bit (4 byte) -3.4028235E^-38 bis 3.4028235^+38
int32_t               i32[2];   // 32 bit (4 byte) -2,147,483,648 bis 2,147,483,647 / -2^31 bis (2^31)-1)
uint32_t             ui32[2];   // 32 bit (4 byte) 0 to 4,294,967,295 / 0 bis (2^32) - 1)
int64_t                 i64;   // 64 bit (8 byte) -9223372036854775808 bis 9223372036854775807/ -2^63 bis (2^63)-1)
```

```
uint64_t                 ui64;   // 64 bit (8 byte) 0 bis 18446744073709551615 / 0 bis (2^64)-1)
long long int           lli64;   // 64 bit (8 byte) -9223372036854775808 bis 9223372036854775807/ -2^63 bis (2^63)-1)
unsigned long long int ulli64;   // 64 bit (8 byte) 0 bis 18446744073709551615 / 0 bis (2^64)-1)
};
```