```
//Prozessoranweisung
#include <datei.h>
#include <math.h>

#define NAME
#define NAME text
#define ADD(x,y)
#define x + y
#undef TEST

#ifdef TEST
#error "Test schon definiert"
#elif (TEST == 1)
#define TEST = 2
#else
#define TEST = 3
#endif

//Datentypen
uint8_t x;    //8 Bit
uint16_t x;   //16 Bit
uint32_t x;   //32 Bit
uint64_t x;   //64 Bit
int8_t x;   //8 Bit
int16_t x;    //16 Bit
int32_t x;    //32 Bit
int64_t x;    //64 Bit

bool x;     //1 Bit   0/1
unsigned char x;//8 Bit   0-255
signed char x;  //8 Bit   -128-127
unsigned int x; //16 Bit  0-65535
signed int x; //16 Bit  -32768-32767
unsigned long x;//32 Bit  0-4294967295
signed long x;  //32 Bit  -2147483648-2147483647
float x;    //32 Bit  +-10^-37/+-10^+38   reelle Zahlen mit ca.  7 Dezimalstellen
double x;   //64 Bit  +-10^-307/+-10^308    reelle Zahlen mit ca. 15 Dezimalstellen

//Typen
auto unsigned char c,d;   // automatisch lokal und dynamisch anlegen  **selten
verwendet
extern unsigned char e,f; // bereits außerhalb des blocks vereinbart  **selten
verwendet
register unsigned char e,f; // möglichst in einem Register anlegen    **selten
verwendet

static unsigned char a,b; // auf fester Adresse (statisch) anlegen, ohne Zuweisung
Anfangswert 0
const unsigned char x = 7;  // Daten sind konstant und dürfen nicht geändert werden
volatile unsigned char y; // Daten können von außen geändert werden (flüchtig)

//Logische Operationen
x = ~a;     //logisches NICHT
x = a & 0x00FF; //logisches UND
x = a ^ b;    //logisches EODER
```

```
x = a | b;    //logisches ODER

x &= y;      //x = x & y
x ^= y;      //x = x ^ y
x |= y;      //s = x | y

// schieben
unsigned char x ,a = 0x04;  // Bitmuster 0000 0100 = 4
x = a << 1; //gibt x = 0x08 = 0000 1000 = 8 (4*2)
x = a >> 1; //gibt x = 0x02 = 0000 0010 = 2 (4/2)

x = (x << 1) | (x >> 7);  //rotiere links 23456781 <- 12345678
x = (x >> 1) | (x << 7);  //rotiere rechts 12345678 -> 81234567

unsigned char x, y;
x = (1 << 7) | (1 << 0);  // lade Konstante 1000 0001
y |= (1 << 7) | (1 << 0); // ODER-Operation y = y ODER Konstante

unsigned char x = 0xff;

x &= ~((1 << 7) | (1 << 0));  // UND-Operation x = x UND Konstante

//PORT Anweisung
PORTB = 0x81;           // jeweils
PORTB = (1 << PB7) | (1 << PB0);  // 1000 0001


//Schleifen
for(unsigned char i = 1; i < 255; i++)  {}  // 8 Bit Schleife aufwärts zählend
for(unsigned int i = 1; i < 65535; i++) {}  // 16 Bit Schleife aufwärts zählend
for(unsigned char i = 255; i > 0; i--)  {}  // 8 Bit Schleife abwärts zählend
for(unsigned int i = 65525; i > 0; i--) {}  // 16 Bit Schleife abwärts zählend
for(unsigned char i = 1; i <= 100; i++) for(unsigned char j = 1; j <= 100; j++);
// 10000 Durchlääufe für Verzögerung! kein Schleifeninhalt, Vorischt COMPILER
for(;;) // unendliche schleife, wie while(1)

while(1)  {}
while(  PORTA & (1 << PA0));  //  warte solange PA0 High
while(!(PORTA & (1 << PA0))); //  warte solange PA0 Low
while(1)
{
  if(!(PINA & (1 << PA0))) continue;  // unterbrechen für PD0 Low (zurück zum
  Anfang)
  if(!(PINA & (1 << PA1))) break;   // abbrechen für PD1 Low (Schleife
  unterbrechen)
}

do {} while (!(PORTA & (1 << PA0)));

loop: PORTA++;
  if(PINA & (1 << PA0))
    goto loop;

//Case
switch(x)
```

```
{
  case 0  : y = 0x30; break;  //Fall x == 0
  case 1  : y = 0x30; break;  //Fall x == 1
  case 2  : y = 0x30; break;  //Fall x == 2
  default : y = 0x30; break;  //Fehlerfall x > 2
}

//Funktion
void warte_1ms(void)
{
  unsigned int i;
    for (i = TAKT/4000ul; i > 0; i--);
} // keine Rückgabe mit return

void warte_x10ms(unsigned char faktor)
{
  unsigned char j;
  unsigned char i;
    for(j = 0; j < faktor; j++)
    {
      for(i = TAKT/400ul; i > 0; i--);
    }
} // keine Rückgabe mit return

unsigned char bin2ascii(unsigned char bin)
{
  if(bin <= 9) return bin + 0x30; else return bin + 0x37;
} // rückgabe mit return

void dual2dezi(unsigned char x, unsigned char *h, unsigned char *z, unsigned char
*e)
{
  *h = x / 100;
  *z = (x % 100) / 10;
  *e = x / 10;
} // keine Rückgabe mit return

unsigned int dual2bcd(unsigned char z)
{
  return ((z/100) << 8) | (((z % 100) / 10) << 4) | ((z %100) %10);
}

void dual3bcd(unsigned char wert, unsigned char *hund, unsigned char *zehn,
unsigned char *ein)
{
  *hund = wert / 100;     //ganzzahlige Division
  *zehn = (wert % 100) / 10;  //ganzzahlige Division des Restes
  *ein = wert %10;      //Divisionsrest
} // kein return da Type void

unsigned char bcd2dual(unsigned char x)
{
  return((x >> 4) * 10 + (x & 0x0f)); //Zener und Einer addieren
}
```

```
unsigned char ascii2bin(unsigned char as)
{
  if (as >= '0' && as <= '9') return as - '0';
    else if (as >= 'A' && as <= 'F') return as - 'A' + 10;
      else if (as >= 'a' && as <= 'f') return as - 'a' + 10;
        else return 0xff; //Fehlermarke
}

warte_1ms();            // Aufruf ohne Argumente
warte_x10ms(10);          // Aufruf mit Wert 10
PORTA = bin2ascii(0x0f);      // Aufruf mit Wert 0x0f
PORTA = ascii2bin(PINB);      // Eingabe umcodiert ausgeben (while Schleife
notwendig)
dual2dezi(123, &hund, &zehn, &ein); // Aufruf mit Wert und Adressen


//Speicher
malloc("Anzahl der Bytes")              //Speicher nicht vorbesetzt zuweisen
Ergebnis: Zeiger
calloc("Anzahl der Werte", "Länge des Datentyps") //Speicher mit NULL vorbesetzt
zuweisen    Ergebnis: Zeiger
free("Zeiger")                    //Speicher freigeben
sizeof("Datentyp")                  //Liefert die Länge des Datentyps
NULL                          //NULLmarke mit Wert 0

  //++Array
      unsigned char eins[1000], x;                            //1-Dimensionales
      Array mit 1000 Bytes
      const unsigned char zwei[2][16] = {
      {0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}
                     {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
                      'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',  } }
                      //2-Dimensionales Array mit

      #define N 1000
    for(unsigned int i=0; i < N; i++) { eins[i] = PORTA;  }         //Port N mal
    speichern
    for(unsigned int i=0; i < N-1; i++) { if(eins[i] != eins[i+1])  x++;  }
    //Flanken zählen

      unsigned char sinus[360];                          //Sinustabelle von 0 bis
      359 Grad
    for (unsigned int i=0; i < 360; i++) sinus[i] = sin(i*M_PI/180)*127 + 127;
  //++Malloc
      #define N 10
    char *p=NULL,*z=NULL;                       //Zeiger mit Null-Marke vorbesetzt

    while(1)
    { p = malloc(N * sizeof(char) );                    //Speicher zuweisen
      if (p != NULL)  PORTA=0x55; else PORTA = 0xAA;          // gelungen??
        for (unsigned char i=0; i < N; i++) *(p+i) = i;       //besetzen
          free(p);
      z = calloc(N, sizeof(char));                  //Speicher mit 0 besetzt
      if (z != NULL)  PORTA=0x55; else PORTA = 0xAA;          // gelungen??
        for (unsigned char i=0; i < N; i++) *(z+i) = *(z+i) + i;  //besetzen
```

```
        free(z);
    }
```