MPS – Praktikum Termin 1 – Sebastian Zill(769544)

1. Installieren Sie einen GNU-C-Compiler auf Ihrem PC, beispielsweise *CodeBlocks*.

1.1 Testen Sie die Entwicklungsumgebung durch Erstellen eines Konsolenprojekts (*Console application*).

1.2 *Achtung: bei CodeBlocks muss die Konsole explizit von Ihnen geschlossen werden, bevor neu kompiliert und ausgeführt werden kann!!*

1.3 Nehmen Sie einige einfache Änderungen und Erweiterungen an Ihrem Programm vor und testen Sie.

Notiz: Probleme mit Standard-Codeblocks Installation gehabt:
– Mingw zusätzlich laden & installieren (in C:/Mingw packen - autodetect)

1. Betrachten Sie das beigefügte Projekt und erweitern Sie

2.1 Was machen die Precompiler-Statements __DATE__, __TIME__, __LINE__?

__DATE__: Gibt das aktuelle Kompilierdatum aus Systemzeit im Format MM DD YYYY anstatt des Platzhalters %s aus
__TIME__: Gibt die aktuelle Kompilierungsuhrzeit aus Systemuhrzeit im Format HH:MM:SS anstatt des Platzhalters %s aus
__LINE__: Gibt die Codezeile anstatt des Platzhalters %d an der Stelle aus

```
printf( "Line %d: This is %s (%s) compiled on %s %s!\n",
        __LINE__, "Test1[main]", "Samuel Sample",
         __DATE__, __TIME__);
```

```
Line 77: This is Praktikum 1[main] (Sebastian Zill 769544) compiled on Nov  1 2021 10:00:20!
```

2.2 Geben Sie eine 8bit-Integer Zahl (uint8_t) als Dezimal- und Hexadezimal-Wert aus.

```
// include <inttypes.h>
uint8_t x8bit = 8;
uint8_t y8bit = 15;
printf("Dezimal: %d \n", x8bit);
// 0: Left-padding number with 0
// 4: Minimum number of characters
printf("Hexadezimal: %04X \n", y8bit);
```

```
Dezimal: 8
Hexadezimal: 000F
```

2.3 Wie kann der Wert binär ausgegeben werden? (eigener Code)

2.4 Ist Ihre Binärdarstellung little- oder big-endian?

Hier: Mit Funktion 127 aufgerufen

```
void printBinary_short(int x) {
    for(int i=31;0<=i;i--) {
        printf("%u", (x>>i)&1);
    }
    printf("\n");
}
```

```
127 Binaer (big endian - most significant bit an hoechster Speicheradresse):
00000000000000000000000001111111
```

3 Characters lesen vom Terminal, d.h. auf dem PC von der Tastatur

3.1 Um das Verhalten der Funktion getc auf einem ARM-Prozessor zu simulieren, ist die Funktion getch_nb()*(nicht blockierend)* vorgeschlagen. (Ohne Eingabe wird der Wert ‚0' zurückgegeben oder bei Tastendruck der ASCII-Wert)

3.2 Testen Sie das Verhalten der Funktion.

3.3 Führen Sie bestimme Aktionen aus bei einer Reihe von Eingaben, z.B. 0, 1, 2, 3.

```
char c;
int asciinull = 48;              // ASCII 0
while (1) {
    c = getch_nb();              // nonblocking read aus util_Term_Keyb
    if (c != 0) {
        int x = c;               // ASCII Wert
        printf( ">%d \n", x );
        if (x == 13) {           // Exit loop on enter (ASCII 13)
            return;
        }
        if (x == (asciinull += 1)) { // ASCII 0 += 1->2->3->4
            if (asciinull == 52) {
                printf( "1-2-3-4 gezaehlt pog!");
                return;
            }
        } else {
            int asciinull = 48;
        }
    }
}
```

4 Setzen und Löschen einzelner Bits.

4.1 Wie können mit C-Code einzeln Bits gesetzt werden?

```
void setBit() {
    int a = 53;                  // Number to manipulate
    printf("Welches Bit von 53 setzen (big endian):\n");
    printBinary_short(53);

    int b;
    scanf("%d", &b);

    printBinary_short(53);
    printBinary_short(1 << b);// Maske mit bit an der zu manipulierenden Stelle erstellen
    printf("==========\n");

    a |= (1 << b);               // A OR Maske (also 1 an der stelle behalten oder schreiben)
    printBinary_short(a);
}
```

4.2 Wie können mit C-Code einzeln Bits gelöscht werden?

```
void deleteBit() {
    int a = 53;                  // Number to manipulate
    printf("Welches Bit von 53 loeschen (big endian):\n");
    printBinary_short(53);

    int b;
    scanf("%d", &b);
    b = ~(1 << b);

    printBinary_short(53);
    printBinary_short(b);// A AND Komplement von Maske (00100 -> 11011)
    printf("==========\n");

    a &= b;
    printBinary_short(a);
}
```

4.3 Schreiben Sie zwei wiederverwendbare Funktionen uint8_t sbit(bytevalue, pos) und uint8_t cbit(bytevalue, pos).

4.4 Testen Sie mit 2.3.

util_Term_Keyb.c:

```
int sbit(int bytevalue, int pos)
{
    return bytevalue | (1 << pos);
}
int cbit(int bytevalue, int pos)
{
    return bytevalue &= ~(1 << pos);
}
```

main.c:

```
int a = 127;                // Number to manipulate

printf("Welches Bit von 127 setzen (big endian):\n");
printBinary_short(127);
int b;
scanf("%d", &b);
printBinary_short(sbit(a, b));

printf("\n");

printf("Welches Bit von 127 loeschen (big endian):\n");
printBinary_short(127);
int c;
scanf("%d", &c);
printBinary_short(cbit(a, c));
```

output:

```
Welches Bit von 127 setzen (big endian):
00000000000000000000000001111111
7
00000000000000000000000011111111

Welches Bit von 127 loeschen (big endian):
00000000000000000000000001111111
5
00000000000000000000000001011111
```

4.5 Was ist ein Macro?

Ein Makro ist eine Quelltextersetzung, die vor dem Kompiliervorgang durchgeführt wird. Sie wird häufig zur Definierung von Konstanten verwendet.

5 Kommentieren Sie – gegebenenfalls nach dem Praktikum zu Hause – Ihren Code. Archivieren Sie Ihr Projekt zu Ihrem späteren Gebrauch.

6 Schreiben Sie ein kurzes Protokoll und fassen Sie Ihre Erkenntnisse zusammen und fügen Sie die jeweiligen Codeabschnitte hinzu. Laden Sie Ihren Code+) als *.zip und Ihr Protokoll als *.pdf in Moodle hoch bis maximal 1 Woche nach dem Termin.