Aufgabe 1: Bitmanipulationen (C)

Implementierungsaufgabe.

Vertauschen von Bytes

Schreiben Sie eine Funktion die das Low Byte (Bits 0-7) und das High Byte (Bits 8-15) vertauscht. Z.B. aus der Zahl 0x4020 wird die Zahl 0x2040.

Verwenden Sie folgenden Funktionsprototypen.

```
short int switchLowHighByte(short int i);
```

Serialisierung/Deserialisierung von Datenstrukturen

Gegenben sind zwei enum Datentypen.

```
typedef enum {
  Stop = 0,
  Start = 1,
  Finish = 5,
  Fail = 255
} Status;
```

```
typedef enum {
  One = 1,
  Fifteen = 15,
  Last = 255
} Numbers;
```

Ihre Aufgabe ist es jeweils Werte der beiden enums in ein Datenpacket der Groesse 16Bit zu packen (serialisieren). Werte des enums Status sollen dabei in das Low Byte und Werte des enums Numbers sollen in das High Byte gepackt werden.

Schreiben Sie eine weitere Funktion, die Werte der enums Status und Number aus einem 16Bit Wert entpackt (deserialisiert). Wir nehmen an, dass die enum Werte mittels der serialize Funktion verpackt wurden.

Verwenden Sie folgende Funktionsprototypen.

```
void serialize(Status s, Numbers n, short int* data);
```

```
void deserialize(short int data, Status* s, Numbers* n);
```

Hinweise

Die Schreibweise

```
typedef enum {
  Stop = 0,
  Start = 1,
  Finish = 5,
  Fail = 255
} Status;
```

vereint die Deklaration eines enum und die Einführung einer Abkürzung. Ausführlicher

```
enum StatusEnum {
  Stop = 0,
  Start = 1,
  Finish = 5,
  Fail = 255
};
typedef enum StatusEnum Status;
```

Tips und Tricks

Bitoperationen

Sollten immer auf `unsigned` ausgeführt werden. Lokaler Typcast!

Hexadezimal Darstellung

C unterstützt die Darstellung und Ausgabe von Zahlen in Hexadezimal Darstellung.

```
  short int zahl = 0x2040;
  printf("%x \n", zahl);
```

Dies ist eine Hilfe beim Testen obiger Funktionen.

Rückgabe als Referenz

`serialize` und `deserialize` verwenden die Methode *Rückgabe als Referenz*.

Per Default, Parameterübergabe in C/C++ ist immer "call-by value". Sprich (Argument)werte werden in die formalen Parameter der Funktion kopiert.

Einfaches Beispiel.

```
int inc(int x) {
     x++;
     return x;
}

int main() {
 int y,z;
 y = 1;
 z = inc(y);
 // z == 2 und y == 1
 return 1; // Falls weggelassen wird irgendein Wert, wohl 0, zurückgelieft
}
```

Referenzübergaben wird in C durch Zeiger simuliert. Angewandt auf unser Beispiel.

```
void incByRef(int x1, int* x2) {
   x1++;
   *x2 = x1;
}

int main() {
  int y,z;
  y = 1;
  incByRef(y, &z);
}
```

- y ist Zeiger auf int
- Zugriff auf int Wert via `*y`.
    - Korrekt. C hat eine kryptische Syntax aber man gewöhnt sich daran.
- Auf der "call site" übergeben wir die Adresse von y via `&y`.

Auch geht folgende Variante

```
void incByRef2(int* x) {
   (*x) ++;
}

int main() {
  int y;
  y = 1;
  incByRef2(&y);
}
```

Testen auf erwartetes Ergebnis

Ein *Testfall* besteht im allgemeinen aus

1. Setzen der Eingabe
2. Ausführung
3. Protokollierung der Ausgabe

Der Testfall ist erfolgreich, falls die Ausgabe mit dem erwarteten Ergebnis übereinstimmt.

Im unseren Fall, ist es recht einfach, das erwartete Ergebnis zu Beschreibung, falls z.B. `switchLowHighByte` zweimal ausgeführt wird.

```
enum TestEnum {
  OK,
  FAIL
};
typedef enum TestEnum Test;

Test testLowHigh(short int i) {
  Test t;
  if(i == switchLowHighByte(switchLowHighByte(i)))
    t = OK;
  else
    t = FAIL;

  return t;
}
```

Als Aufgabe für Sie bleibt die Auswahl einer Reihen von signifikanten Testeingaben.

Beachte: Obige Testeigenschaft `testLowHigh` ist notwendig aber nicht hinreichend für die Korrektheit von `switchLowHighByte`.

- Notwendig weil falls `switchLowHighByte` korrekt ist die Testeigenschaft sicherlich git.
- Nicht hinreichend weil z.B. eine fehlerhafte Implementierung in der immer die Eingabe unverändert zurückgeliefert wird, die Testeigenschaft erfüllt, nicht aber die Anforderungen an `switchLowHighByte`.

Test Set-up serialize/deserialize

```
Test testSD(Status s, Numbers n) {
 Test t;
 short int data;
 Status s2;
 Numbers n2;

 // Test execution
 serialize(s, n, &data);
 deserialize(data, &s2, &n2);

 if(s2 == s && n2 == n) {
```

```
    t = OK;
 }
 else {
    t = FAIL;
 }
 return t;
}
```

Source file

Verwenden Sie folgendes Source file.

```
/**
 * @file   aufgabe1-bit.c
 * @brief  Aufgabe1 - Bit Manipulationen
 */

#include <stdio.h>
// Falls notwendig erweitern Sie die Liste der includes


/**
   @brief Aufgabe1a: Vertauschen von Bytes
   @param [in] short int i
   @return short int

Schreiben Sie eine Funktion die das Low Byte (Bits 0-7) und
das High Byte (Bits 8-15) vertauscht.
Z.B. aus der Zahl 0x4020 wird die Zahl 0x2040.
*/

short int switchLowHighByte(short int i) {

  // Ihre Loesung

  return 1;
}


typedef enum {
  Stop = 0,
  Start = 1,
  Finish = 5,
  Fail = 255
} Status;


typedef enum {
  One = 1,
  Fifteen = 15,
  Last = 255
} Numbers;

/**
   @brief Aufgabe1b: Serialisierung von Datenstrukturen
   @param [in] Status s
   @param [in] Numbers n
   @param [out] short int* data

Gegeben sind zwei enums. Ihre Aufgabe ist es jeweils Werte der
```

```
beiden enums in ein Datenpacket der Groesse 16Bit zu packen (serialisieren).
Werte des enums Status sollen dabei in das Low Byte und Werte
des enums Numbers sollen in das High Byte gepackt werden.
*/


void serialize(Status s, Numbers n, short int* data) {
  // Ihre Loesung
}

/**
   @brief Aufgabe1c: Deserialisierung von Datenstrukturen
   @param [in] short int data
   @param [out] Status* s
   @param [out] Numbers* n

Schreiben Sie eine Funktion die Werte der enums Status und Number
aus einem 16Bit Wert entpackt (deserialisiert).
Wir nehmen an, dass die enum Werte mittels der serialize Funktion
verpackt wurden.
*/

void deserialize(short int data, Status* s, Numbers* n) {
  // Ihre Loesung
}


enum TestEnum {
  OK,
  FAIL
};
typedef enum TestEnum Test;

Test testLowHigh(short int i) {
  Test t;
  if(i == switchLowHighByte(switchLowHighByte(i)))
    t = OK;
  else
    t = FAIL;

  return t;
}

int main() {
  // Ihre Testroutinen

  short int zahl = 0x2040;
  printf("%x \n", zahl);

}
```