3. Aufgabe: Funktionen/Bit-Operatoren

1 Einleitung

In dieser Aufgabe sollen Sie sich mit Funktionen und bitweisen Operatoren vertraut machen. Dabei kommen als Instrument zur Visualisierung sog. *Aktivitätsdiagramme* zum Einsatz.

2 Aufgaben

2.1 Anlegen eines Projekts

Erstellen Sie ein Projekt mit Namen *Bitoperatoren*. Implementieren und testen Sie folgende Funktion zur sicheren Abfrage einer ganzen Zahl vom Typ *short*:

```
short getShort(          // [out] user input
        char text[])     // [in] question text for user input
{
    short value;         // user input
    int finished=0;      // flag for correct user input
    char ch;             // character behind number
    int retVal;          // return value of scanf

    do {
        // get user input
        printf("%s: ", text);
        ch = '\0';
        retVal = scanf("%hd%c", &value, &ch);

        // check for valid user input
        if(retVal!=2) printf("Das war keine korrekte Zahl!\n");
        else if(ch!='\n') printf("Unerwartete Zeichen hinter der Zahl!\n");
        else finished = 1;

        // clear input stream
        while(ch!='\n') scanf("%c", &ch);

        // repeat if not finished
    } while(!finished);

    // return user input
    return value;
}
```

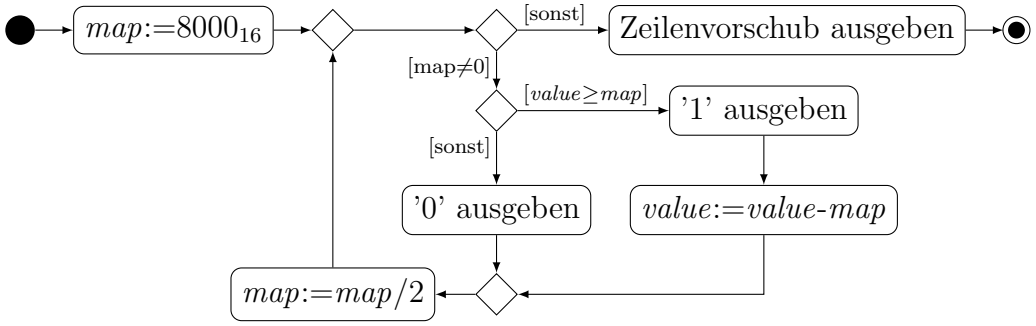
2.2 Erstellung eines Aktivitätsdiagramms

Erstellen Sie zu der Funktion aus Aufgabe 2.1 ein Aktivitätsdiagramm und bringen Sie es vorbereitet zum Praktikum mit.

2.3 Funktion zur binären Ausgabe

Erstellen Sie eine Funktion mit Namen *printBinary*, die als Parameter eine Zahl vom Typ *unsigned short* mit Namen *value* erwartet und nichts zurück gibt. Die Funktion soll ganze Zahlen binär auf

dem Bildschirm ausgeben. Realisieren Sie die Funktion wie folgt:



Testen Sie anschießend die Funktion. Entspricht die Ausgabe Ihren Erwartungen?

2.4 Operatoren zur Bitmanipulation

Fragen Sie im Hauptprogramm *main()* vom Benutzer eine Zahl vom Typ *short* ab und speichern Sie die z.B. in die Variable *val*. Danach geben Sie eine Tabelle mit folgender Kopfzeile aus:

```
Inhalt | dez. mit Vorz. | dez. ohne Vorz |  oktal  | hexadezimal | dual
```

In dieser Tabelle sollen die Operatoren zur Bitmaipulation getestet werden. Geben Sie in jeweils einer Zeile folgende Werte aus: Die Benutzereingabe *val*, die Konstante 15, bitweise Negation `~val`, bitweises Und `val&15`, bitweises Oder `val|15`, bitweises Exklusiv Oder `val^15`, bitweises Schieben nach links `val<<2` und bitweises Schieben nach rechts `val>>2`.

Hinweise: Verwenden Sie für die 2. bis 5. Spalte entsprechend die Platzhalter %hd, %hu, %ho und %hX. Für die binäre Ausgabe verwenden Sie die zuvor erstellte Funktion *printBinary()*. Auch wenn die Funktion *printBinary* den Datentyp *unsigned short* erwartet, können Sie ihr auch den Datentyp *short* übergeben – das ausgegebene Bitmuster bleibt unverändert.

2.5 Programm testen

Überprüfen Sie Ihr Programm mit dem dezimalen Wert 20, indem Sie zuvor alle Werte theoretisch bestimmen und dann mit Ihrem Programm vergleichen. Legen Sie dafür auf einem separaten Zettel eine Tabelle an, und füllen Sie diese zuvor per Hand aus.

2.6 Quellcode gestalten

Gehen Sie bei der Gestaltung des Quellcodes wie bei den letzten Praktika vor. Des Weiteren:

- Verwenden Sie keine globalen Variablen
- Halten Sie die übliche Reihenfolge ein: Include-Befehle, Makros (wenn vorhanden), Funktionsdeklarationen, Hauptprogramm, Funktionsdefinitionen
- Fügen Sie zur besseren Gliederung Leerzeilen und ggf. Linien ein

3 Optionale Zusatzaufgabe

Überlegen Sie sich weitere Zahlenbeispiele und testen Sie damit Ihr Programm. Implementieren Sie das Zweierkomplement binär unter Zuhilfenahme des Inkrement-Operators.

Happy Computing!