karriere tutor®
Gemeinsam. Digital. Erfolgreich.

C++

Funktionen und Bit-/Schiebeoperationen

Aufgabe 0: Aussagen einordnen

Wahr oder falsch? Begründe deine Antwort.

a) Bitweise Operatoren wie `&`, `|` und `^` sind nur auf Ganzzahlen (int, short, etc.) anwendbar, nicht auf Gleitkommazahlen (float, double).

b) Der Schiebeoperator `<<` verschiebt die Bits nach links und verdoppelt somit den Wert der Zahl.

c) Der Schiebeoperator `>>` verschiebt die Bits nach rechts und halbiert den Wert der Zahl für alle Datentypen.

d) Funktionen in C++ können keine Parameter per Referenz übergeben, sondern nur per Wert.

e) Der Ausdruck `x << 1` ist äquivalent zu einer Multiplikation von `x` mit 2, solange `x` eine positive Ganzzahl ist.

f) Es ist möglich, benutzerdefinierte Datentypen (wie Strukturen oder Klassen) mit bitweisen Operatoren zu verwenden, indem man die Operatoren überlädt.

g) Funktionen in C++ dürfen keine Funktionszeiger als Parameter akzeptieren.

h) Der Bitweiser Operator `^` (XOR) ergibt 1, wenn beide Bits unterschiedlich sind, und 0, wenn sie gleich sind.

i) Die Standardbibliothek von C++ bietet eine Funktion namens `bitset`, die zur Manipulation einzelner Bits in einem Datentyp verwendet werden kann.

j) Der Schiebeoperator `<<` kann nur zur Bitmanipulation verwendet werden, nicht zur Ausgabe von Texten mit `std::cout`.

Aufgabe 1: Bit-Setzer mit Validierung und flexibler Maskierung

Erstelle eine Funktion, die nicht nur ein Bit an einer angegebenen Position setzt, sondern auch eine Validierung für ungültige Positionen durchführt und es ermöglicht, zwischen dem Setzen eines Bits (auf 1) und dem Zurücksetzen eines Bits (auf 0) zu wählen. Die Funktion soll außerdem mit beliebig großen Ganzzahlen (z. B. 64-Bit oder 32-Bit) funktionieren und eine entsprechende Fehlermeldung zurückgeben, falls eine ungültige Position gewählt wurde.

Anforderungen:

- Das Bit an der angegebenen Position kann entweder auf 1 gesetzt oder auf 0 zurückgesetzt werden.
- Falls die Bit-Position außerhalb des Bereichs der Ganzzahl liegt, soll eine Fehlermeldung ausgegeben werden.

```
int setBit(int number, int position, bool setToOne);
```

Erweiterungen:

- Die Funktion berücksichtigt sowohl das Setzen (setToOne = true) als auch das Zurücksetzen (`setToOne = false`) eines Bits.
- Sie prüft, ob die angegebene Position innerhalb des gültigen Bitbereichs liegt. Wenn nicht, wird eine Fehlermeldung ausgegeben.

Aufgabe 2: Bit-Umkehrer für große Zahlen mit dynamischer Bitbreite

Schreibe eine Funktion, die die Bits einer Ganzzahl umkehrt, aber zusätzlich berücksichtigt, dass die Größe der Zahl (z. B. 32-Bit oder 64-Bit) dynamisch angegeben werden kann. Implementiere die Funktion so, dass du nicht nur die Bits umkehrst, sondern auch sicherstellst, dass die führenden Nullen korrekt behandelt werden.

Anforderungen:

- Die Funktion soll mit verschiedenen Bitbreiten (z. B. 8, 16, 32, 64 Bit) arbeiten können.
- Es soll sichergestellt werden, dass die führenden Nullen korrekt berücksichtigt und nicht umgekehrt werden.
- Die Funktion soll die Bitbreite als zusätzlichen Parameter akzeptieren und die Umkehrung darauf beschränken.

```
unsigned long long reverseBits(unsigned long long number, int bitWidth);
```

Erweiterungen:

- Die Funktion unterstützt verschiedene Bitbreiten (8, 16, 32, 64 Bit).
- Die Validierung stellt sicher, dass nur zulässige Bitbreiten verwendet werden.
- Die führenden Nullen werden korrekt behandelt, indem die Umkehrung nur auf die tatsächliche Bitbreite beschränkt wird.

Aufgabe 3: Bit-Paar-Tausch

Erstelle eine Funktion, die eine Ganzzahl akzeptiert und die benachbarten Bit-Paare der Zahl tauscht. Beispielsweise soll Bit 0 mit Bit 1, Bit 2 mit Bit 3, usw. getauscht werden. Gib die veränderte Zahl zurück.

```
int swapBitPairs(int number);
```

Hinweis: Verwende bitweise Operatoren, um die geraden und ungeraden Bits zu isolieren und dann mit den Schiebeoperatoren zu tauschen.

Aufgabe 4: Anzahl der gesetzten Bits (Popcount)

Schreibe eine Funktion, die eine Ganzzahl als Eingabe akzeptiert und die Anzahl der gesetzten Bits (Bits, die `1` sind) in der Zahl zurückgibt.

```
int countSetBits(int number);
```

Hinweis: Verwende eine Schleife und den bitweisen AND-Operator, um jedes Bit zu überprüfen.

Aufgabe 5: Zirkuläre Bitverschiebung

Schreibe eine Funktion, die eine Ganzzahl und eine Anzahl von Bits als Parameter akzeptiert. Die Funktion soll die Bits der Zahl zirkulär nach links verschieben, d.h. die Bits, die links herausgeschoben werden, sollen rechts wieder hineingeschoben werden. Gib die veränderte Zahl zurück.

```
int rotateLeft(int number, int bits);
```

Hinweis: Verwende eine Kombination aus Links- und Rechtsverschiebungen sowie bitweisen OR-Operationen, um die Bits zu zirkulieren.

Aufgabe 6: Bitbereich auslesen

Schreibe eine Funktion, die eine Ganzzahl, eine Startposition und eine Länge akzeptiert. Die Funktion soll die Bits aus der Zahl, die an der Startposition beginnen und über die angegebene Länge reichen, extrahieren und zurückgeben.

```
int extractBitRange(int number, int start, int length);
```

Diese Aufgaben stellen höhere Anforderungen an das Verständnis von bitweisen Operationen und erfordern eine durchdachte Nutzung der Bitmasken, Schiebeoperationen und logischen Operatoren in C++. Sie helfen dabei, ein tieferes Verständnis der Manipulation von Bits in realen Anwendungen zu erlangen. Viel Erfolg!