Überblick

Überblick

Kompilieren eines C Programmes

- bevor Programm ausführbar ist, muss es von Compiler in Maschinensprache übersetzt werden

→ Neben Compiler werden für das Übersetzen des Quelltextes der Präprozessor und der Linker benötigt

Kompilieren eines C Programmes

- bevor Programm ausführbar ist, muss es von Compiler in Maschinensprache übersetzt werden

→ Neben Compiler werden für das Übersetzen des Quelltextes der Präprozessor und der Linker benötigt

Präprozessor: einfacher Textübersetzer der Makroanweisungen auswertet und ersetzt

Kompilieren eines C Programmes

- bevor Programm ausführbar ist, muss es von Compiler in Maschinensprache übersetzt werden

→ Neben Compiler werden für das Übersetzen des Quelltextes der Präprozessor und der Linker benötigt

Präprozessor: einfacher Textübersetzer der Makroanweisungen auswertet und ersetzt

- Präprozessor übergibt an Compiler, welcher in Maschinensprache übersetzt

Kompilieren eines C Programmes

- bevor Programm ausführbar ist, muss es von Compiler in Maschinensprache übersetzt werden

→ Neben Compiler werden für das Übersetzen des Quelltextes der Präprozessor und der Linker benötigt

Präprozessor: einfacher Textübersetzer der Makroanweisungen auswertet und ersetzt

- Präprozessor übergibt an Compiler, welcher in Maschinensprache übersetzt
- Linker verbindet einzelne Programmmodule miteinander, sodass ausführbare Datei entsteht

Kompilieren eines C Programmes

- bevor Programm ausführbar ist, muss es von Compiler in Maschinensprache übersetzt werden

→ Neben Compiler werden für das Übersetzen des Quelltextes der Präprozessor und der Linker benötigt

Präprozessor: einfacher Textübersetzer der Makroanweisungen auswertet und ersetzt

- Präprozessor übergibt an Compiler, welcher in Maschinensprache übersetzt
- Linker verbindet einzelne Programmmodule miteinander, sodass ausführbare Datei entsteht

Überblick

Datentyp const

```
1 #include<stdio.h>
2 // konstante Variable
3 const float MwStFaktor = 0.19;
4
5 int main() {
6     float betrag = 350.0;
7     float mwst = betrag * MwStFaktor;
8     printf("%.f% von %.2f Euro: %.2f Euro\n",
9     MwStFaktor * 100, betrag, betrag *
          MwStFaktor);
10    return 0;
11 }
```

Ausgabe

19% von 350.00 Euro: 66.50 Euro

Datentyp static

```
1 int zaehlen() {
2     static int counter = 0;
3     return ++counter;
4 }
5 int main() {
6     printf("Zaehler: %d\n", zaehlen());
7     printf("Zaehler: %d\n", zaehlen());
8     printf("Zaehler: %d\n", zaehlen());
9     return 0;
10 }
```

Ausgabe

Zähler: 1
Zähler: 2
Zähler: 3

- teilt dem Compiler mit, dass Variable durch Ereignisse außerhalb des Programms verändert werden kann

→ Variable muss vor jedem Zugriff neu aus Hauptspeicher eingelesen werden (darf nicht in Register

- teilt dem Compiler mit, dass Variable durch Ereignisse außerhalb des Programms verändert werden kann

→ Variable muss vor jedem Zugriff neu aus Hauptspeicher eingelesen werden (darf nicht in Register zwischengespeichert werden)

→ Compiler arbeitet ohne Optimierung

- teilt dem Compiler mit, dass Variable durch Ereignisse außerhalb des Programms verändert werden kann

→ Variable muss vor jedem Zugriff neu aus Hauptspeicher eingelesen werden (darf nicht in Register zwischengespeichert werden)

→ Compiler arbeitet ohne Optimierung

```
1 volatile char keyPressed;
2 int count = 0;
3
4 while (keyPressed != 'a') {
5     count++;
6 }
```

- teilt dem Compiler mit, dass Variable durch Ereignisse außerhalb des Programms verändert werden kann

→ Variable muss vor jedem Zugriff neu aus Hauptspeicher eingelesen werden (darf nicht in Register zwischengespeichert werden)

→ Compiler arbeitet ohne Optimierung

```
1 volatile char keyPressed;
2 int count = 0;
3
4 while (keyPressed != 'a') {
5     count++;
6 }
```

Überblick

Logische Operatoren wie in Java

Operator	Syntax
Addition	$a + b$
Subtraktion	$a - b$
Modulo	$a\%b$
Inkrement Prefix	$++a$
Dekrement Postfix	$a--$
Logical NOT	$!a$
Logical AND	$a\&\&b$
Logical OR	$a\|\|b$

Bitweise Operatoren

Operator	Syntax
Bitwise AND	$a \& b$
Bitwise OR	$a \mid b$
Bitwise NOT	$\sim a$
Bitwise XOR	$a \hat{} b$
SHIFT	$c = b << 1$

Beispiel 1: Bitwise AND

```
1 int main()
2 {
3     .
4     .
5     int a = 10, b = 7, c;
6     c = a & b;
7     .
8     .
9 }
```

Welchen Wert hat nun die Variable c?

Lösung des Beispiels 1

Die Variable hat nun den Wert $c = 2$, denn

$$10_{(10)} = 1010_{(2)}$$
$$7_{(10)} = 0111_{(2)}$$

$$\begin{array}{r} 1010_{(2)} \\ \&\,0111_{(2)} \\ \hline 0010_{(2)} \end{array} = 2_{(10)}$$

Lösung des Beispiels 1

Die Variable hat nun den Wert $c = 2$, denn

$$10_{(10)} = 1010_{(2)}$$
$$7_{(10)} = 0111_{(2)}$$

$$\begin{array}{r} 1010_{(2)} \\ \&\,0111_{(2)} \\ \hline 0010_{(2)} = 2_{(10)} \end{array}$$

Beispiel 2: Bitwise NOT

```
1 int main()
2 {
3     .
4     .
5     int a = 3;
6     .
7     .
8     a = ~a;
9 }
```

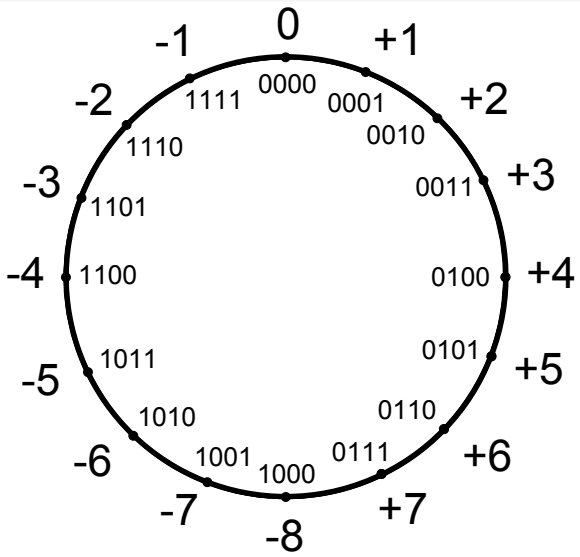
Welchen Wert hat die Variable a?

Lösung des Beispiels 2

Die Variable a hat den Wert a = -4, da

Bitwise NOT

bei dem bitweisen NOT jedes Bit invertiert wird. Möchte man eine Dezimalzahl negieren, muss danach noch eine 1 addiert werden.



Quelle: http://de.wikipedia.org/wiki/Arithmetischer_Ueberlauf