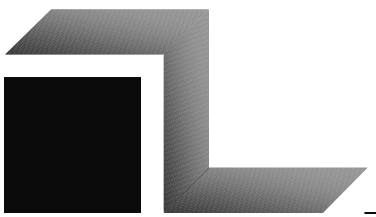
C / C++ für Java-Programmierer

Der Präprozessor

Prof. Dr. Wolfgang Schramm



FACHHOCHSCHULE
MANNHEIM

Hochschule für Technik und Gestaltung

Präprozessor

- Ist dem Compiler vorgeschaltet („Prä“-).
- Spezielle Anweisungen im Quelltext, die sog. Präprozessoranweisungen werden vom Präprozessor erkannt und verarbeitet.
- Der Präprozessor verändert den Quelltext, bevor er vom Compiler übersetzt wird.

Vorteile von Präprozessor-Anweisungen

- Die Präprozessoranweisungen dienen i.w. dazu das Quellprogramm kompakter und damit übersichtlicher zu gestalten→ Bessere Lesbarkeit
- Bessere Portierbarkeit von Programmen; geschicktes Verstecken von maschinen-abhängigen Details durch Präprozessor-Anweisungen

allgemeine Syntax

C/C++

- i.d.R. # in der ersten Spalte
- unmittelbar nach #: gewünschte Präprozessor-Anweisung, Leerzeichen sind nicht zulässig
- darf i.d.R. nicht über mehrere Zeilen hinausgehen, muss in der Zeile, in der sie begonnen wurde, beendet werden
- sind mehrzeilige Präprozessor-Anweisungen erforderlich: \ als letztes Zeichen in der Zeile

Arten von Präprozessor-Anweisungen

- Kopieren von externen Dateien in die zu übersetzende Datei (#include)
- Textuelles Ersetzen von symbolischen Konstanten und Makros durch zuvor definierte Werte (#define)
- Bedingte Übersetzung (#ifdef ...)

Präprozessor-Anweisungen: **Einbinden von Dateien**

C/C++
...
...
...
...
...

Syntax: **#include <**dateiname**>** **oder** **#include "**dateiname**"**

Datei wird in speziellem Verzeichnis gesucht: Standard-include-Verzeichnis

Datei wird in aktuellem Verzeichnis gesucht

Semantik: An der Stelle, an der die include-Anweisung steht, wird die angegebene externe Datei in die zu übersetzende Datei kopiert.

Verwendung: Informationen, die für mehrere Programme von Interesse sind an zentraler Stelle verfügbar machen.

wichtigste Anwendung: Einbindung von Standard-Headerdateien (Nutzung der mitgelieferten Laufzeitbiliotheken).

Präprozessor-Anweisungen: **Einbinden von Dateien - Beispiel**

Quelldatei:

```
Zeile 1
#include "b.h"
Zeile 2
#include "c.h"
Zeile 3
```

Datei b.h:

```
Zeile b1
Zeile b2
```

Datei c.h:

```
Zeile c1
Zeile c2
```

```
Zeile 1
Zeile b1
Zeile b2
Zeile 2
Zeile c1
Zeile c2
Zeile 3
```

Eingabe für Compiler nach Präprozessorlauf

Präprozessor-Anweisungen: **Konstantendefinition**

Einfachste Form der #define-Anweisung.

C/C++

Beispiele:

```
#define EINS 1
```

Alle Vorkommen von EINS im Bezeichnerkontext eines Programms werden durch 1 ersetzt.

```
#define forever for (;;)    /* Endlosschleife */
```

vernünftige Konstanten:

```
#define TRUE 1
#define FALSE 0
#define PI 3.1415926535
#define AND &&
#define OR ||
#define NOT !
```

kompliziertes Beispiel:

C/C++
…
…
…
…
…

```
#define IF if (
#define THEN ) {
#define ELSE ; } else {
#define END ; }
```

Quellprogramm

```
IF 5 <= 10 THEN
   printf ("5 kleiner gleich 10")
ELSE
   printf ("5 größer 10")
END
```

```
if ( 5 <= 10 ) {
   printf ("5 kleiner gleich 10")
; } else {
   printf ("5 größer 10")
; }
```

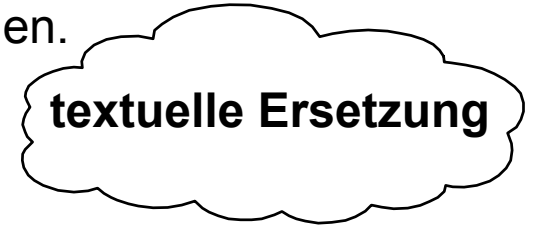
Vom Compiler modifiziertes Quellprogramm

Präprozessor-Anweisungen: **Makrodefinition**

Syntax: **#define** name ["(" Parameterliste ")"] [Wert]

C/C++
...
...
...
...
...

Semantik: Man gibt Zeichenketten an, die vor der Übersetzung des Programms in andere Zeichenketten (aus-)getauscht werden sollen.



Darf an jeder beliebigen Stelle im Programm erfolgen.
Sichtbarkeit ab Definitionsende bis Dateiende.

Begriffe:

`#define ...`	Makrodefinition
`#undef ...`	Makrodefinition (ab diesem Punkt) außer Kraft setzen
späteres Austauschen	Makrosubstitution
definierte Zeichenkette	Makro

Präprozessor-Anweisungen: **Makros**

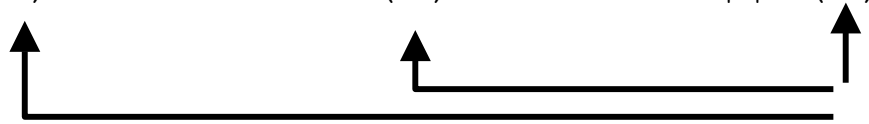
C/C++
…
…
…
…
…

```
#define KLEINER_NULL(a) a < 0

if (KLEINER_NULL(i) || KLEINER_NULL(x))

if ( i < 0 || x < 0)

#define SCHALTJAHR(a)\
(a)%4 == 0 && (a)%100 !=0 || (a)%400 ==0
```



Wichtig: Klammerung des Parameters

```
if (SCHALTJAHR(2001)) . . .
if (SCHALTJAHR(2002)) . . .
if (SCHALTJAHR(2003)) . . .
if (SCHALTJAHR(2004)) . . .
if (SCHALTJAHR(2001+x)) . . .

(2001+x)%4 == 0 && (2001+x)%100 !=0 || (2001+x)%400 == 0

2001+(x %4) == 0 && 2001+(x % 100) !=0 || 2001+(x %400) ==
```

Makros vs. Funktionen

C/C++
...
...
...
...
...

- Argumente von Makros können beliebige Datentypenannehmen. Es ist sogar möglich, dass Makros lokale Variablen vereinbaren können.
- Makros sind effizienter (vermeiden Laufzeit-Overhead).
- Der Quelltext wird umfangreicher, da jeder Makroaufruf eine textuelle Expansion bedingt.
- Rekursion ist nicht möglich.
- Komplizierte Makros → fehleranfällig.
- I.d.R. sollten Makros für sehr kurze Funktionen eingesetzt werden.

Bedingte Übersetzung 1/4

C/C++
...
...
...
...
...

- Problem in der Praxis: Ein Programm wird für verschiedene Plattformen (Software, Hardware – z.B. das zugrundeliegende Betriebssystem ist entweder Unix oder Windows) mit plattformabhängigen (man sagt auch maschinenabhängigen) Programmteilen entwickelt.
- Man möchte trotzdem 2 Versionen des Quelltextes vermeiden, da sonst die Gefahr besteht, dass bei Änderungen nicht synchron verfahren wird.

➔ Bedingte Übersetzung

- Hierbei werden bestimmte Teile des Quellcode mittels Präprozessordirektiven von der Compilation ausgeschlossen.
- Benötigte Direktiven:
  - ifdef
  - ifndef
  - elif
  - else
  - endif

Bedingte Übersetzung 2/4

C/C++

...

...

...

...

...

Syntax:

```
#ifdef Bezeichner1
Programmtext1
[#elif Bezeichner2
Programmtext2]
. . . .
[#elif Bezeichnern
Programmtextn]
[#else Programmtextn+1]
#endif
```

Semantik:

Falls $Bezeichner_1$ vorher mit der define-Direktive definiert wurde, wird $Programmtext_1$ in die Übersetzung einbezogen. Ist das nicht der Fall, wurde aber $Bezeichner_2$ vorher mit der define-Direktive definiert , wird $Programmtext_2$ in die Übersetzung einbezogen etc.

Ist keiner der Bezeichner definiert, wird $Programmtext_{n+1}$ in die Übersetzung einbezogen.

Bedingte Übersetzung 3/4

C/C++

```
#define UNIX  /* nur an dieser Stelle muss geändert werden */

#ifdef W2K
        Programmtext1 mit Windows2000 spezifischen Anweisungen
#elif UNIX
        Programmtext2 mit Unix spezifischen Anweisungen
#endif
```

Bedingte Übersetzung 4/4

C/C++
...
...
...
...
...

Problem in der Praxis: Beim Testen streut man eigene Debug-Anweisungen in den Programmtext ein (print Anweisungen).

Diese nur zum Testen des Programm notwendigen Ausgabeanweisungen kann man mittels der ifdef-Direktive ein- bzw. ausschalten (dann wenn das Programm in Produktion geht).

```
#define DEBUG  /* nur an dieser Stelle muss geändert werden */

......

#ifdef DEBUG

   printf (....);

   printf (....);

#endif
```

#-Operator (stringizing operator)

- Dem Parameter kann im Makroersatztext der #-Operator voran-gestellt werden.
- Beim Aufruf wird das entsprechende Argument in Anführungszeichen (“”) gesetzt, also in einen String konvertiert.

```
#define printint(i) printf(“Wert “ #i “ = %d“, i)


int a, b;

. . .

printint (a-b); →     printf(“Wert “ “a-b“ “ = %d“, a-b);


                      printf(“Wert a-b = %d“, a-b);
```

##-Operator (token past operator)

- Dem Parameter kann im Makroersatztext der ##-Operator vor- oder nachgestellt werden.
- Beim Aufruf zieht der Präprozessor die Zeichenfolgen links und rechts vom ##-Operator zusammen. Leerzeichen werden ignoriert. Entsteht dabei wieder ein Makro-Aufruf, dann wird auch dieses Makro expandiert.

```
#define show(var, nr) printf(#var #nr " =%.1f\n", var ## nr)


float x5 = 16.4;

. . .

show (x,5); → printf("x" "5" " =%.1f\n", x5);
```

Ausgabe: x5 = 16.4