

C-Programmieren: Präprozessor und Makros in C

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 1

C-Präprozessor

- In diesem Kapitel wollen wir zeigen:
 - ➔ welche Funktion der C-Präprozessor hat
 - ➔ welche wichtigen Anweisungen es für den C-Präprozessor gibt
 - ➔ was Makros sind
 - ➔ wie man eigene *Header*-Dateien vereinbart.
 - ➔ wie man mit C-Präprozessor-Anweisungen den Übersetzungsvorgang beeinflussen kann.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 2

C-Präprozessor-Anweisungen (1)

- Vor dem eigentlichen Aufruf des C-Compilers wird der **C-Präprozessor** ausgeführt.
- Der Präprozessor ist ein **reines Textersetzungsprogramm**. Damit kann zur **Laufzeit keine Änderungen** des Programmablaufs mehr erfolgen.
- Alle Präprozessor-Anweisungen beginnen mit dem ASCII-Zeichen **#**.
- Vor und nach dem **#** kann eine beliebige Anzahl von **white spaces** stehen.

Beispiel:

```
# define max 100 /* Vereinbarung: max*/
int main ( void ) /* Hauptprogramm */
{
    short i = max/2; /* Definition und Initialisierung: i = 50 */
    ...
}
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 3

C-Präprozessor-Anweisungen (2)

- Im Gegensatz zum C-Compiler ist der C-Präprozessor **zeilenorientiert**:
- Jede Präprozessor-Anweisung **endet mit dem Zeilenende-Zeichen (ASCII: CR)**.
Deshalb darf nach einer Präprozessor-Anweisung **ohne besonderen Grund niemals ein Semikolon** stehen.
- Jedoch kann mit Hilfe des **Zeilenabschlusszeichens** **** die Präprozessor-Anweisung in der nächsten Zeile fortgesetzt werden.
- Eine Präprozessor-Anweisungen kann an **beliebiger Stelle** im Quellcode eingeschoben werden.
- Zur besseren Übersicht stehen diese Anweisungen normalerweise **statisch zu Beginn des C-Quelltextes**.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 4

C-Präprozessor: Funktionen

- Mit dem Präprozessor kann man u.a.:
 - ➔ Dateien einfügen
 - ➔ Texte im Quellcode ersetzen
 - ➔ die Kompilierung von Bedingungen (z.B. vom Prozessor) abhängig machen
 - ➔ Strings in Parameter umwandeln
 - ➔ Makros definieren
 - ➔ eigene *Header*-Dateien definieren
 - ➔ prozessorspezifische Implementierungen definieren.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 5

C-Präprozessoranweisungen: *#include* (1)

➔ *# include <Dateiname>*

- Der Präprozessor **kopiert den Quelltext** des ursprünglichen Programms **in eine temporäre Datei** für die aktuelle Kompilierung.
- Dabei **ersetzt er sukzessive** alle *#include*-Anweisungen **durch die** in den Dateien enthaltenen **Quelltexte**.
- Die Klammern *<...>* bewirken, dass die *#include*-Dateien im **Standardverzeichnis des C-Compilers** gesucht werden, z.B. in *MSVC++*:

D:\Microsoft Visual Studio\VC98\Include

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 6

C-Präprozessoranweisungen: **#include** (2)

➡ **# include "Dateiname"**

- Die Anführungszeichen **"..."** bewirken, dass die Datei mit dem Namen **Dateiname** im **Verzeichnis des C-Quelltextes** gesucht werden.

Beispiele:

Die Datei **SOURCE.C** im Verzeichnis **usr\src\work** enthält u.a. folgende Präprozessoranweisungen:

include <stdio.h> /* **stdio.h** wird im **Standard Include-Verzeichnis** z.B. **D:\Microsoft Visual Studio\VC98\Include** gesucht. */

include "my_header.h"

/* Die Datei **my_header.h** wird im Verzeichnis **der C-Quellcodedatei** z.B. **I:\MS Visual C++\Projekt1** gesucht. */

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 7

C-Makros: **#define** (1)

➡ **# define Text Ersatztext**

- Die **define**-Anweisung leitet in C ein ➡ **Makro** ein.
- Dabei ersetzt der Präprozessor den Text **stur** 🚫 durch den Ersatztext.

Beispiel:

define max 500

/* Dem Text **max** wird der Ersatztext **500** zugewiesen. */

define min (max/10)

/* Dem Text **min** wird der Ersatztext **(500/10)** zugewiesen. */

...

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 8

C-Makros: **#define** (2)

➡ **# define Makroname(Par1 ,..., ParN) Ersatztext**

- Die **#define**-Anweisung leitet in C ein Makro ein, mit dem die **Parameter Par1 ,..., ParN übergeben** werden.
- Dabei darf zwischen **Makroname** und (**kein Leerzeichen (blank)** stehen.
- Ein Makro ist **keine Funktion**. Deshalb
 - können die Parameter **nicht dynamisch** (d.h. **während des Programmlaufs**) **verändert werden**.
 - enthalten die Funktionsparameter **keine Typbezeichnungen**.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 9

C-Makros: **#define** (3)

- Die **#define**-Anweisung leitet eine
➡ **Makro-Definition** ein:

➡ **# define Makroname(Par1 ,..., ParN) Ersatztext**

Beispiel:

```
# define  quadrat_zahl( x )  (x) * (x)
/* Das Makro quadrat_zahl mit dem Parameter x wird definiert. */
...
int main (void)
{
    int a = 3, y ;
    ...
    y = quadrat_zahl(a + 4) ;    /* quadrat_zahl(a + 4) ersetzt der
                                Präprozessor durch den Ersatztext (a+4) * (a+4) */
    ...
}
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 10



Aufgabe: C-Makros (1)

Untersuchen Sie das folgende Programm!

1. Was wird auf dem Bildschirm ausgegeben?

2. Welche Fehler enthält das Programm?

```
# include <stdio.h>
# define min      2 ;
# define max      5
# define x        max-3
# define hoch2( b )    b * b
int main (void)
{ short w, y, z;
  y = min ;
  ➔ z = -2 + min + max ;
  w = hoch2(x);
  printf ("\n x = %d\n y = %d\n z = %d\n w = %d\n\n", x, y, z, w);
  return 0; }
```

➔ Der MSVC-Compiler gibt **keine Fehler oder Warnungen** aus.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 11



Aufgabe: C-Makros (1)

1. Ausgabe:

```
x = 2
y = 2
z = 0
w = -13
Press any key to continue.
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 12

✓ Aufgabe: C-Makros (2)

1. Ausgabe:

Erwartete Bildschirmausgabe:

$x = 2$ $x = 2 = \max - 3 = 5 - 3 = 2$
 $y = 2$ $y = 2 = \min = 2$
 ➔ $z = 0$ ➔ $z = 5 = -2 + \min + \max = -2 + 2 + 5 = 5$
 ➔ $w = -13$ ➔ $w = 4 = (\max - 3) * (\max - 3) = (5 - 3) * (5 - 3) = (2 * 2) = 4$

• Tatsächliche Berechnung:

$x = \max - 3;$ $= 5 - 3 = 2$
 $y = \min ; ;$ $= 2$
 $z = -2 + \min ; + \max;$ ⚡: Das Semikolon wird ebenfalls ersetzt!
 $= -2 + 2; + 5; = 0$
 $w = \max - 3 * \max - 3 = 5 - 3 * 5 - 3 = 5 - 15 - 3$
 $= 5 - 15 - 3 = -13$ ⚡: reine Textersetzung ohne Klammern!



Eigene Header-Dateien (1)

➔ Header-Dateien

sind **normale C-Quelltextdateien**, welche die Dateierweiterung (extension) ***.h** aufweisen.

- Programmteile, die bei der **Programmentwicklung nicht verändert werden**, wie z.B.

- **#define**-Anweisungen
- Konstantendefinitionen
- Makros
- Funktionsprototypen

sollten in **eigenen Header-Datei** gespeichert werden.



Eigene Header-Dateien (2)

➡ Header-Dateien

sind **normale C-Quelltextdateien**, welche die Dateierweiterung (extension) ***.h** aufweisen.

- Die **Header-Dateien** werden in den C-Quelltext mit einer **#include** -Anweisung eingebunden.
- **Header-Dateien** werden wie normale C-Quelltextdateien kompiliert.
Sie erzeugen **kein ausführbares Programm (*.exe)**.

06.06.2004



Eigene Header-Dateien (3)

- Eigene **Header-Dateien** haben u.a. folgende Vorteile:
 - ➡ Das zu entwickelnde Programm ist **übersichtlicher**, weil es weniger Text enthält.
 - ➡ Die Programmteile im **Header** sind **nicht so anfällig für unbeabsichtigte Veränderungen**.
 - ➡ Wenn komplexe Programme von mehreren Entwicklern parallel entwickelt werden, so greifen alle auf **denselben Satz von Definitionen** zu.
 - ➡ Verändert ein Entwickler die Definitionen in einer **Header-Datei**, so ist dies sofort am Speicherdatum der Datei zu erkennen.

06.06.2004



Eigene Header-Dateien: Beispiel

- Ausschnitt aus einer Header-Datei **SPIEL.H**:

```

/*-----*/
/* Definitionen für ein BildschirmSpiel          */
/*-----*/
#define MAXx      640 /* maximal 640 Pixel in x-Richtung */
#define MAXy      480 /* maximal 480 Pixel in y-Richtung */
#define SPIELFELDbx MAXx-MAXx/10 /* Spielfeldausdehnung x */
#define SPIELFELDIx MAXy-MAXy/10 /* Spielfeldausdehnung y */
#define BALLradius MAXy/20 /* Ballradius definieren */
...

```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 17

C-Präprozessoranweisungen: **#undef**

➔ **# undef Makroname**

- Die **#undef**-Anweisung hebt eine vorherige **#define**-Anweisung auf.
- Damit kann man die Gültigkeit eines Makros auf den Programmabschnitt zwischen der **#define**-Anweisung und der **#undef**-Anweisung begrenzen.

Beispiel:

```

#define min 10 /* ab hier gilt: min ist ... */
int main (void) /*... statisch definiert: min = 10 */
{ short z, max = 1000 ;
  z = -2 + min + max ;
...
#define min /* ab hier ist min nicht mehr definiert */
{
  int min = max / 10 ; /* nun ist min in diesem Block ... */
... } } /* ... dynamisch definiert */

```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 18

**Aufgabe: dynamische und statische Variable**

Untersuchen Sie das folgende Programm!

1. Welchen Fehler enthält das folgende Programm?

```
# define  min 10    /* ab hier gilt: min ist statisch definiert: min = 10 */
#include <stdio.h> /* Bibliothek stdio.h einbinden */

...
int main (void)    /* Hauptprogramm */
{ short z, max = 1000 ;    /* Variablendefinition*/
  z = -2 + min + max ;
  printf ( "\n min = %d,  max = %d  z = %d\n", min, max, z) ;
  min = 2*min ;
...
# undef  min      /* ab hier ist min nicht mehr definiert */
{
  int min = max / 10 ; /* min neu definieren */
  printf ( "\n min = %d,  max = %d\n", min, max ) ;
}
return 0 ; }
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 19

**Aufgabe: dynamische und statische Variable****1. Programmfehler:**

```
# define  min 10    /* ab hier gilt: min ist statisch definiert: min = 10 */
#include <stdio.h> /* Bibliothek stdio.h einbinden */

...
int main (void)    /* Hauptprogramm */
{  short z, max = 1000 ;    /* Variablendefinition*/
  z = -2 + min + max ;
  printf ( "\n min = %d,  max = %d  z = %d\n", min, max, z) ;
  ➡ min = 2*min ;
...
# undef  min      /* ab hier ist min nicht mehr definiert */
...
➡ Hier ist min statisch definiert, d.h. zur Laufzeit des Programms
konstant. Deshalb darf min hier kein neuer Wert zugewiesen werden.
MSVC++ gibt für die Zeile min = 2*min ;
die Warnung Linker Operand muss ein L-Wert sein aus.
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 20

C-Präprozessoranweisungen: bedingte Compilierung



```
# if
# elif
# else
# endif
# ifdef
# ifndef
```

- Diese Präprozessoranweisungen für die **bedingte Kompilierung** werden benutzt, um beispielsweise
 - C-Programme an **unterschiedliche Betriebssysteme** anzupassen.
 - C-Programme für **unterschiedliche Prozessoren** zu entwickeln.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 21

Präprozessoranweisungen: **#if, #else, #elif, #endif**

- Diese Präprozessoranweisungen entsprechen den **if - else**-Anweisungen in C.
- Die Bedingungen werden genau so ausgewertet wie in C:
wahr entspricht einem Wert $\neq 0$,
falsch einem Wert = 0.
- **#else** und **#elif** sind optionale Parameter für die **Wahl zwischen 2 und mehr Alternativen** (2-seitige Bedingungsprüfung).
- Der Präprozessor ist **zeilenorientiert** und **kennt keine Verbundanweisungen**, wie z.B. **{ ... }** in C.

Deshalb schließt man eine Folge von
#if .. #else-Anweisungen mit **#endif** ab.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 22

Unterschied zwischen **#else** und **#elif**

- **#else** und **#elif** sind optionale Parameter für die **Wahl zwischen 2 und mehr Alternativen**.
- Bei **#elif** wird im Unterschied zu **#else** die Bedingungsprüfung **abgebrochen**, **nachdem der logische Wert **wahr** erkannt worden ist**.

Insofern entspricht die Präprozessoranweisung **#elif** einer **else { ... break; }**-Verbundanweisung in C.

06.06.2004



Bedingte Kompilierung: Anpassen an unterschiedliche Prozessoren

- Man möchte gegebenenfalls Programme
➡ **prozessorunabhängig**
schreiben.
- Deshalb ist es u.U. erforderlich, bestimmte Systemfunktionen **rechnerabhängig** zu implementieren.
- Ein Beispiel für diese Anwendung ist die
➡ **Anpassung der Datentypen** an unterschiedliche Prozessoren.

06.06.2004



Bedingte Kompilierung

- Beispiel: Typanpassung

```
# define LINT          /* Das Programm wird für INT = long übersetzt */
/* # define SINT */    /* Das Programm wird für INT = short übersetzt */
# ifdef LINT
    typedef long INT ;
# endif
# ifdef SINT
    typedef short INT ;
# endif
...
int main (void)        /* Hauptprogramm */
{ INT z, max = 1000 ; /* Variablendefinition: INT ist hier long */
  z = -2 + max ;
  ...
  return 0 ; }
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 25

Bedingte Kompilierung:

unterschiedliche Längen der Datentypen

- Präprozessor-Anweisungen sind statisch, d.h. sie werden **vor der Kompilierung des Programms** durchgeführt.
 - Wenn man die **Datentypen mit Präprozessor-Anweisungen** beeinflussen möchte, so darf man dabei keine **dynamischen** Befehle, wie z.B. **sizeof ...** verwenden.
- C verwendet deshalb die *Header-Datei* **limits.h** zur Festlegung der **prozessorspezifischen Länge** der ganzzahligen Datentypen und **float.h** für die Gleitpunkt-Datentypen.

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 26

Bedingte Kompilierung: **limits.h** (1)

- Definitionen in **limits.h** in MSVC++ (1):

```
#ifndef _INC_LIMITS /* Mehrfacheinbindung von limits.h verhindern */
#define _INC_LIMITS
#if !defined(_WIN32) && !defined(_MAC)
#error ERROR: Only Mac or Win32 targets supported!
#endif
#define CHAR_BIT 8 /* number of bits in a char */
#define SCHAR_MIN (-128) /* minimum signed char value */
#define SCHAR_MAX 127 /* maximum signed char value */
#define UCHAR_MAX 0xff /* maximum unsigned char value */
#ifndef _CHAR_UNSIGNED
#define CHAR_MIN SCHAR_MIN /* minimum char value */
#define CHAR_MAX SCHAR_MAX /* maximum char value */
#else
/* ... */
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 27

Bedingte Kompilierung: **limits.h** (2)

- Definitionen in **limits.h** in MSVC++ (2):

```
/* ... */
#define CHAR_MIN 0
#define CHAR_MAX UCHAR_MAX
#endif /* _CHAR_UNSIGNED */
#define MB_LEN_MAX 2 /* max. # bytes in multibyte char */
#define SHRT_MIN (-32768) /* minimum (signed) short value */
#define SHRT_MAX 32767 /* maximum (signed) short value */
#define USHRT_MAX 0xffff /* maximum unsigned short value */
#define INT_MIN (-2147483647 - 1) /* minimum (signed) int value */
#define INT_MAX 2147483647 /* maximum (signed) int value */
#define UINT_MAX 0xffffffff /* maximum unsigned int value */
#define LONG_MIN (-2147483647L - 1) /* minimum (signed) long value */
#define LONG_MAX 2147483647L /* maximum (signed) long value */
#define ULONG_MAX 0xffffffffUL /* maximum unsigned long value */
/* ... */
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 28

Definition der Konstanten im Header *limits.h* (MSVC++)

Konstante	Bedeutung	Wert
CHAR_BIT	Anzahl der bits in der kleinsten Variablen, die kein Bitfeld ist	8
SCHAR_MIN	Minimum einer signed char Variablen	-128
SCHAR_MAX	Maximum einer signed char Variablen	127
UCHAR_MAX	Maximum einer unsigned char Variablen	255 (0xff)
CHAR_MIN	Minimum einer char Variablen	-128; 0 bei der /J-Compileroption
CHAR_MAX	Maximum einer char Variablen	127; 255 bei der /J-Compileroption
MB_LEN_MAX	maximale Anzahl von Bytes in einer Multicharacter Konstanten.	2
SHRT_MIN	Minimum einer short Variablen	-32768
SHRT_MAX	Maximum einer short Variablen	32767
USHRT_MAX	Maximum einer unsigned short Variablen	65535 (0xffff)
INT_MIN	Minimum einer int Variablen	-2147483647 - 1
INT_MAX	Maximum einer int Variablen	2147483647
UINT_MAX	Maximum einer unsigned int Variablen	4294967295 (0xffffffff)
LONG_MIN	Minimum einer long Variablen	-2147483647 - 1
LONG_MAX	Maximum einer long Variablen	2147483647
ULONG_MAX	Maximum einer unsigned long Variablen	4294967295 (0xffffffff)

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 29

Bedingte Kompilierung: Mehrfacheinbindung (1)

- **#include** -Anweisungen werden ggf. von **mehreren verschiedenen** Bibliotheken oder **Headern** aufgerufen
- Um Fehler zu vermeiden, wird empfohlen **keine Bibliothek mehrfach** einzubinden
- Eine **Mehrfacheinbindung** des Headers **limits.h** verhindert **MSVC++** automatisch auf folgende Weise:

```
/* ... */
#ifdef _INC_LIMITS      /* limits.h schon eingebunden ? */
#define _INC_LIMITS    /* nein: Bibliothek limits.h einbinden */
/* ... hier stehen weiter Header-Anweisungen von limits.h! */
#endif /* _INC_LIMITS: Ende von #ifdef _INC_LIMITS */
```

- Falls die Präprozessorkonstante **_INC_LIMITS** noch nicht definiert ist, wird sie definiert und **die weiteren Präprozessoranweisungen werden eingebunden**.

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 30

Parameter in Strings umwandeln:

- Präprozessor-Parameter, vor denen **#** steht, werden in einen **konstanten String** umgewandelt.

- Beispiel:

```
# define out( x )    printf ( #x " = %f\n", x)
                    /* x in konstanten String umwandeln */

...

int main ( void )   /* Hauptprogramm */
{
    float y = 1.2345 ;
    out(y) ;         /* erzeugt wird der konstante String:
                       printf ( "y" = %f\n", y) ; */

    return 0 ;
}
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 31

Strings verknüpfen:

- Der Präprozessor-Operator **##** **verkettet Strings**.
- Das Ergebnis ist ein **konstanter String**.
- Dabei werden **alle Leerzeichen** in der Umgebung von **##** entfernt.

- Beispiel:

```
# define link( x, y )      x ## y
                          /* x und y zu einem konstanten String verketteten */

# include <stdio.h>

...

int main ( void )   /* Hauptprogramm */
{
    int k1 = 1, k2 = 2 ;
    printf ( " k1 = %d \n k2 = %d \n", link( k,1 ), link( k,2 ) ) ;
    /* erzeugt wird der konstante String:
       printf ( " k1 = %d \n k2 = %d \n", k1, k2 ) ; */

    return 0 ;
}
```

06.06.2004

Hochschule für angewandte Wissenschaften Hamburg
Prof. Dr.-Ing. U. Vogelsang, Prof. Dr.-U. Sauvagerd



Programmieren in ANSI C:
K15_Präprozessor & Makros.ppt, Seite 32

Präprozessor-Anweisung: **#error**

- Der Präprozessor-Operator
error **Fehlertext**
erzeugt beim Kompilieren **einen Fehlertext**.
- Damit kann man beispielsweise verhindern, dass ein Programm für eine **Hardware oder Software übersetzt wird, auf der es nicht läuft**:

- Beispiel:

```
...  
# ifndef      WINDOWS  
    /* Der C-Code ist für Windows nicht geeignet */  
# error Das Programm laeuft unter MS WINDOWS nicht!
```

- Das Beispiel setzt voraus, dass Windows-Programme die folgende Präprozessor-Anweisung enthalten:

```
# define      WINDOWS
```