

Kapitel 5: Der C-Präprozessor

C-Präprozessoranweisungen

- ▶ Präprozessor-Anweisungen beginnen mit #
- ▶ Präprozessor-Anweisungen werden vor dem Kompiliervorgang ausgeführt
- ▶ Präprozessor-Features:
 - ▶ Zusammenführen von Zeilen die durch \ aufgeteilt wurden
 - ▶ Definition und Ersetzung von Makros (`#define`)
 - ▶ Einschleusung von Dateiinhalten (`#include`)
 - ▶ Beeinflussung des Compilers (`#pragma`)
 - ▶ Bedingte Kompilierung (`#ifdef`)

https://de.wikibooks.org/wiki/C-Programmierung:_Präprozessor#.23pragma

Definition und Ersetzung von Makros

In C sind Makros mit und ohne Parameter zulässig

```
#define MAKRO $Ersatztext
#define MAKRO(<Parameterliste>) $Ersatztext
```

Beispiele:

```
#define MONTHS_PER_YEAR 12
#define MAXIMUM(a,b) ((a) > (b) ? (a) : (b))
```

Nutzung:

```
int foo=7, bar=10;
int monthly_costs = MAXIMUM(foo, bar); // (foo) > (
    bar) ? (foo): (bar)
int annual_costs = monthly_cost * MONTHS_PER_YEAR;
    // monthly_cost * 12
```

ANSI-C Macros

Fünf Makros die jeder ANSI-C-Compiler unterstützen muss.

__LINE__ : Zeilennummer der momentanen Quelldatei
 __FILE__ : Name der momentanen Quelldatei
 __DATE__ : Übersetzungsdatum der Quelldatei
 __TIME__ : Übersetzungszeit der Quelldatei
 __STDC__ : Erkennungsmerkmal für ANSI-C Compiler

```

1  #include <stdio.h>
2
3  int main() {
4      if(__STDC__) {
5          printf("%s_ %d:_", __FILE__, __LINE__);
6          printf("%s_(%s)\n", __DATE__, __TIME__);
7      }
8      return 0;
9  }
```

Rekursive Makrodefinition

Falls der Name eines Makros innerhalb seiner eigenen Definition auftaucht, wird nicht ersetzt, sondern übernommen.

Beispiele

```
#define sqrt(x) printf("sqrt(%f) = %f\n", x, sqrt(x))  
...  
sqrt(9.61); // printf("sqrt(%f) = %f\n", 9.61, sqrt(9.61));
```

Frage: Warum findet diese Ersetzung nicht statt?

Einschleusung von Dateiinhalten

- ▶ Üblicherweise werden mit der `#include` Anweisung Headerdateien (Endung `.h`) eingebunden
- ▶ Eigene Headerdateien werden in Anführungszeichen angegeben (z.B. `#include "foobar.h"`)
- ▶ Die folgenden im ANSI-C Standard festgelegten Dateien werden in spitzen Klammern angegeben (z. B. `#include<stdio.h>`)

<code><assert.h></code>	<code><ctype.h></code>	<code><complex.h></code>	<code><errno.h></code>
<code><fenv.h></code>	<code><float.h></code>	<code><iso646.h></code>	<code><inttypes.h></code>
<code><limits.h></code>	<code><locale.h></code>	<code><math.h></code>	<code><setjmp.h></code>
<code><signal.h></code>	<code><stdalign.h></code>	<code><stdarg.h></code>	<code><stdatomic.h></code>
<code><stdbool.h></code>	<code><stddef.h></code>	<code><stdio.h></code>	<code><stdint.h></code>
<code><stdlib.h></code>	<code><stdnoreturn.h></code>	<code><string.h></code>	<code><tgmath.h></code>
<code><threads.h></code>	<code><time.h></code>	<code><uchar.h></code>	<code><wchar.h></code>
		<code><wctype.h></code>	

Siehe https://en.wikipedia.org/wiki/C_standard_library

Bedingte Kompilierung

Mit Hilfe von `#ifdef`-Anweisungen kann festgelegt werden ob bestimmte Programmteile kompiliert werden oder nicht.

Schlüsselwort	Bedeutung
<code>#if \$bedingung</code>	Test ob eine Bedingung erfüllt ist
<code>#ifdef \$name</code>	Test ob ein Makro definiert wurde
<code>#ifndef \$name</code>	Test ob ein Makro noch nicht definiert wurde
<code>#else</code>	Leitet den <code>else</code> -Programmteil ein
<code>#endif</code>	Beendet eine bedingte Kompilierungskonstruktion

Bedingte Kompilierung: Beispiel

```
1  #include<stdlib.h>
2  #include<stdio.h>
3
4  #define BAR
5
6  int main(){
7  #ifdef FOO
8      puts("Macro_FOO_is_defined.");
9
10 #elif defined BAR
11     puts("Macro_BAR_is_defined.");
12
13 #else
14     puts("Neither_FOO_nor_BAR_is_defined.");
15 #endif
16
17     return EXIT_SUCCESS;
18 }
```


Endianness (byte order)

Je nach Rechner-Architektur werden Ganzzahlen, die aus mehreren Bytes bestehen, unterschiedlich im Speicher abgelegt.

- **Big-Endian**: Das höchstwertigste Byte wird zuerst gespeichert.

int x = 0xAABBCCDD;			
AA	BB	CC	DD

- **Little-Endian**: Das niederwertigste Byte wird zuerst gespeichert.

int x = 0xAABBCCDD;			
DD	CC	BB	AA

Bedingte Kompilierung: Real-World-Beispiel

```
1  #if __BYTE_ORDER == __LITTLE_ENDIAN
2      #define TO_LITTLE_ENDIAN_64(n) (n)
3      #define TO_LITTLE_ENDIAN_32(n) (n)
4
5  #elif __BYTE_ORDER == __BIG_ENDIAN
6      #define TO_LITTLE_ENDIAN_64(n) bswap_64(n)
7      #define TO_LITTLE_ENDIAN_32(n) bswap_32(n)
8
9  #else
10     #warning "byte_order_couldn't_be_detected".
11     #define TO_LITTLE_ENDIAN_64(n) (n)
12     #define TO_LITTLE_ENDIAN_32(n) (n)
13
14 #endif
```

Beeinflussung des Compilers

Mit der `#pragma`-Anweisung kann die Codegenerierung beeinflusst werden. Im Folgenden werden die zwei wichtige Pragmas besprochen die der `gcc` unterstützt.

- ▶ `#pragma pack`: Speicheralignment
- ▶ `#pragma once`: Include-Guard

Speicherausrichtung (Memory Alignment)

- ▶ Bei dem Zugriff auf eine Speicheradresse lesen moderne Rechner immer mehrere Bytes (**Wort**).
- ▶ Die **Wort**länge von moderenen CPUs ist meist 4 byte oder 8 byte.
- ▶ Anzahl gelesener Bytes ist eine Zweierpotenz: 1, 2, 4, 8,
- ▶ Größe der C-Datentypen auf einem 32-Bit Prozessor.

char: 1-Byte

short: 2-Byte




int: 4-Byte

long: 4-Byte

void: 4-Byte

- ▶ **Frage:** Wie sind die Größen auf einem 64-Bit Prozessor?

Misalignment (Falsch ausgerichteter Speicher)

- ▶ Geg: 32-Bit Umgebung.
- ▶ Auf dem dem Stack werden nacheinander Variablen der folgende Typen gepusht:
 - ▶ char: 
 - ▶ short: 
 - ▶ int: 



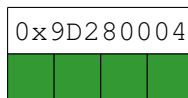
Der Speicher ist Misalignment, da der Zugriff auf die `int`-Variable Bitshifts benötigt.

(**Beispiel:** $\star (0xD928000000) \ll 24 \quad || \quad \star (0xD9280004) \gg 8$)

Speicherausrichtung (Memory Alignment)

- ▶ Einige Prozessoren werfen bei dem Zugriff auf eine misalignte Variable eine `alignment` Exception (Alignment Fault).
- ▶ eine solche Exception führt in der Regel zum Programmabbruch.
- ▶ Compiler versuchen oftmals misalignment durch Padding (■) zu vermeiden.
- ▶ Beispiel: Pushen von Variablen auf den Stack a 32-Bit CPU.

- ▶ `char`: 
- ▶ `short`: 
- ▶ `int`: 



Speicheralignment von structs

```
1  #include <stdio.h>
2  #include <stdint.h>
3
4  typedef struct { char a;  short b;  int c; } foo;
5  typedef struct { char a;  short b; } bar;
6
7  int main () {
8      const uint64_t clen = sizeof(char);
9      const uint64_t slen = sizeof(short);
10     const uint64_t ilen = sizeof(int);
11
12     uint64_t reglen = clen + slen + ilen;
13     printf("Foo:_%lu_/_%lu\n", reglen, sizeof(foo));
14
15     reglen = clen + slen;
16     printf("Bar:_%lu_/_%lu\n", reglen, sizeof(bar));
17 }
```

Frage: Was ist die Ausgabe des Programms?

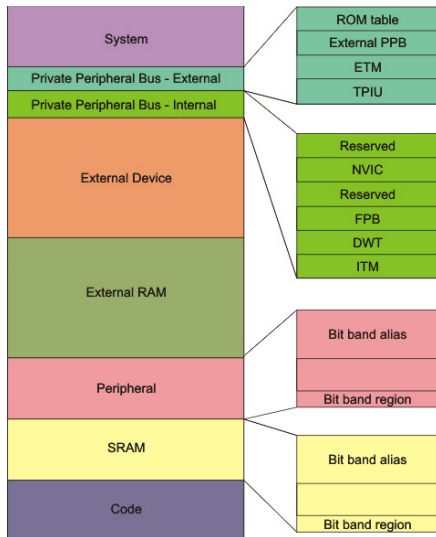
Speicheralignment mittels pragma pack

Mit der Anweisung `#pragma pack` kann das Speicheralignment für Verbundstypen (structs) bestimmt werden.

```
1  #include <stdio.h>
2  #pragma pack(4)
3  struct foo { char a; int b; char c; };
4
5  #pragma pack(1)
6  struct bar { char a; int b; char c; };
7
8  int main() {
9      printf("sizeof_foo:_%lu\n", sizeof(struct foo));
10     printf("sizeof_bar:_%lu\n", sizeof(struct bar));
11     return 0;
12 }
```

Frage: Was ist die Ausgabe des Programms?

Use-Cases: Memory Mapped Hardware Interface



Quelle: <https://www.cnblogs.com/shangdawei/p/3319632.html>

Systeminteraktion

- ▶ Das Betriebssystem setzt ein bestimmtes Alignment voraus.
- ▶ `#pragma pack()`
Setzt das Alignment auf den Startzustand zurück.
- ▶ `#pragma pack(push)`
Legt das aktuelle Alignment auf einen Stack
- ▶ `#pragma pack(pop)`
Nimmt das oberste Alignment vom Stack.
- ▶ `-fpack-struct[=n]`
Compiler-Flag mit dem der Startzustand angepasst werden kann.
- ▶ `#pragma pack()` sollte immer von einem `#pragma pack(push)` und `#pragma pack(pop)` eingerahmt werden.

Beispiel: Nutzung des Pragmas pack

```
1  #include <stdio.h>
2
3  #pragma pack(push)
4  #pragma pack(1)
5  struct foo { int a; char b; int c; };
6  #pragma pack(pop)
7
8  struct bar { int a; char b; int c; };
9
10
11 int main() {
12     printf("sizeof_foo:_%lu\n", sizeof(struct foo));
13     printf("sizeof_bar:_%lu\n", sizeof(struct bar));
14     return 0;
15 }
```

Include-Guard-Makro

Wird eine Headerdatei von mehreren C-Dateien, welche Teile eines Programmes sind, genutzt, kann es zu unerlaubten doppelten Definitionen kommen.

Daher verwendet man sogenannte *Include-Guards*.

Beispiel:

```
// foo.h
#ifndef FOO_H
#define FOO_H

#define FNORD 23

#endif /* FOO_H */
```

Alternative: Anweisung `#pragma once` zu Beginn einer Headerdatei.

Zusammenfassung

Sie sollten ...

- ▶ ... C-Präprozessor-Makros schreiben können.
- ▶ ... die C-Präprozessor-Anweisungen zur bedingten Kompilierung beherrschen.
- ▶ ... wissen was es mit der Anweisung `#pragma pack` auf sich hat.
- ▶ ... verstanden haben, warum es in C Include-Guards gibt.