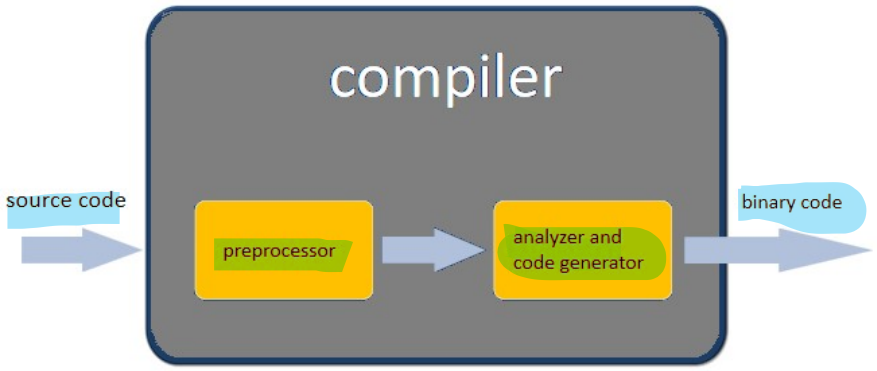
16 Der Präprozessor

Bevor der eigentliche Compiler an die Übersetzung eines C/C++ -Quelltextes geht, wird dieser zunächst von dem C/C++ -Präprozessor bearbeitet. Der C/C++ -Präprozessor ist ein einfacher Makro Prozessor, mit dem sich Spracherweiterungen implementieren lassen.



In den bislang geschriebenen Programmen haben wir von einer Funktionalität des C/C++ -Präprozessors schon oft Gebrauch gemacht, nämlich von der des Einfügens eines Quelltextes in einen anderen durch die `#include`-Anweisung. Präprozessoranweisungen beginnen immer mit dem Doppelkreuz (#). Dieses Doppelkreuz muss immer das erste Zeichen einer Zeile sein, wobei optional führende Leerzeichen erlaubt sind. Der C/C++ -Präprozessor kennt u.a. Direktiven

- zum Textersatz
- und zur bedingten Übersetzung
- zum Einfügen eines Quelltextes in einen anderen

16.1 Textersatz – #define / #undef

Der Präprozessor kann einfache Textersetzungen in C/C++ -Quellen vornehmen. Der Textersatz in Programmen basiert auf der #define-Direktive. Mit dieser Direktive lassen sich so genannte Präprozessor - Makros (oder kurz Makros) definieren. Tritt ein solcher Makro dann später in einer C/C++ -Quelle auf, so wird er durch seine Definition ersetzt. Wird etwa der Makro ZEILENLAENGE durch die Präprozessor Direktive

```
#define ZEILENLAENGE 80
```

definiert, so wird im folgenden C/C++ -Quelltext jedes Auftreten des Wortes ZEILENLAENGE außerhalb von Zeichenkettenkonstanten durch 80 ersetzt. Die Deklaration eines char - Vektors zur Aufnahme einer Textzeile kann dann also durch

```
char textzeile [ ZEILENLAENGE ];
```

erfolgen.

Darüberhinaus gestattet der C/C++ -Präprozessor es, parameterisierte Makros zu definieren. Ein Beispiel eines parameterisierten Makros zur Berechnung des Maximums zweier Werte A und B wäre etwa

```
#define MAX(A,B) ((A) > (B) ? A : B)
```

Nach der Definition des Präprozessormakros MAX wird nun jedes Auftreten der Zeichenfolge

```
MAX(Argument1,Argument2)
```

in einem C/C++ -Programm durch die Zeichenfolge

```
((Argument1) > (Argument2) ? Argument1 : Argument2)
```

ersetzt. Bei der Definition und Verwendung von parameterisierten Präprozessormakros ist darauf zu achten, dass sehr leicht unerwünschte Effekte entstehen können, wie das folgende Beispiel zeigt:

```
ergebnis = MAX ( x++, y++ );
```

Die Textersetzung hierbei ergibt:

```
ergebnis = ((x++) > (y++) ? x++ : y++);
```

Man erkennt leicht, dass, wenn x größer als y ist, die Variable x zweimal inkrementiert wird. Im anderen Fall wird die Variable y doppelt inkrementiert. Dieser Effekt ist vom Programmierer sicherlich nicht beabsichtigt!
Des Weiteren ist es immer sinnvoll, die Parameter von Präprozessormakros einzuklammern, wie das folgende Beispiel eines unsauber geklammerten Präprozessormakros zeigt:

```
#define MULT(X,Y) X*Y
```

Die Verwendung des Makros MULT könnte dann etwa so aussehen:

```
z = MULT( 2 + 3, 4 + 5);
```

Nach der Expansion des Makros, also nach dem Textersatz ergibt sich:

```
z = 2 + 3 * 4 + 5;
```

Das Ergebnis ist 19 und nicht, wie vom Programmierer sicherlich erwartet 45. Dies liegt daran, dass nach der Expansion des Makros die Multiplikation 3 * 4 einen höheren Vorrang hat als die beiden Additionen! Ein sauber definierter Präprozessormakro MULT sollte also etwa so aussehen:

```
#define MULT(X,Y) ((X)*(Y))
```

Jetzt ergibt sich nach der Expansion der Ausdruck

```
z = ((2 + 3) * (4 + 5));
```

Dieser ergibt nach der Bewertung dann das erwartete Ergebnis.

Aufhebung von Makrodefinitionen
Die Definition eines mittels der `#define`-Direktive vereinbarten Makros kann durch die

```
#undef <Makroname>
```

Direktive aufgehoben werden.

Beispiel für undef:

```
int add(int x) {
      return x + 1;
}

int main(void) {
      int i = 100;
      i = add(i);
#define add(x)    (2 * (x))
      i = add(i);
#undef add
      i = add(i);
      printf("%d",i);
      return 0;
}
```

⇨

```
int add(int x) {
    return x + 1;
}

int main(void) {
    int i = 100;
    i = add(i);

    i = (2 * ( i )) ;

    i = add(i);
    printf("%d",i);
    return 0;
}
```

Vordefinierte Makros:

_ _LINE_ _ aktuelle Zeilennummer
_ _FILE_ _ Dateiname der Quelldatei
_ _DATE_ _ Datum der Compilierung
_ _TIME_ _ Uhrzeit der Compilierung

Beispiel:

```
printf("this is line %d\n", __LINE__);
printf("Hello from the source file named "__FILE__"\n");
printf("The program was successfully compiled on " __DATE__ "\n");
printf("I was compiled at " __TIME__);
```

16.2 Bedingte Übersetzung

Der C/C++ -Präprozessor kann auch dazu benutzt werden, abhängig von bestimmten Bedingungen Teile des Quelltextes von der Übersetzung durch den C/C++ -Compiler auszunehmen oder gezielt einzuschließen. Der C/C++ -Präprozessor kennt die folgenden Direktiven zum Testen von Bedingungen:

`#if` <Ausdruck>: Der hiervon abhängige Quelltext wird übersetzt, falls <Ausdruck> von 0 verschieden ist.

`#ifdef (<Makroname>)`: Der hiervon abhängige Quelltext wird übersetzt, falls der Präprozessormakro <Makroname> definiert ist.

`#ifndef (<Makroname>)`: Der hiervon abhängige Quelltext wird übersetzt, falls der Präprozessormakro <Makroname> nicht definiert ist.

Diese Direktiven können durch eine der folgenden beiden Direktiven fortgesetzt werden:
`#else`: Der hiervon abhängige Quelltext wird übersetzt, falls der vom zugeordneten `#if`... abhängige Quelltext nicht übersetzt wird.

`#elif <Ausdruck>`: Der hiervon abhängige Quelltext wird übersetzt, falls <Ausdruck> von 0 verschieden ist.

Den Abschuss bildet in jedem Fall die Direktive `#endif`. Die in den Präprozessordirektiven zur bedingten Übersetzung verwendeten Ausdrücke werden aus den üblichen logischen, arithmetischen und Vergleichsoperatoren (+, -, *, /, ..., ==, !=, ..., &&, ||, !, ...) sowie dem Operator `defined (<Makroname>)` gebildet. Der Operator `defined (<Makroname>)` liefert genau dann den Wahrheitswert wahr, wenn der Präprozessormakro `<Makroname>` definiert ist, und falsch sonst. Demzufolge sind die folgenden Präprozessordirektiven äquivalent:

```
#ifdef <Makroname>

#if defined (<Makroname>)
```

beziehungswiese

```
#ifndef <Makroname>

#if !defined (<Makroname>)
```

Alle Präprozessordirektiven zur bedingten Übersetzung dürfen ineinander geschachtelt werden.

Beispiel:

```
#define GERMAN  1
#define ENGLISH 2
#define FRENCH   3

#define LANGUAGE GERMAN

int main ()
{
#if LANGUAGE == GERMAN
    printf ("Hallo Welt\n");
#elif LANGUAGE == ENGLISH
    printf ("Hello world\n");
#else
    printf ("Bonjour monde\n");
#endif
}
```

⇨

```
int main ()
{
    printf ("Hallo Welt\n");
}
```

16.3 Textimport #include bzw. Headerfiles

Mittels #include-Direktive werden explizit Dateien an der angegebenen Stelle in den Programmcode eingefügt. Der Inhalt der eingefügten Datei wird daraufhin ebenfalls geparst.

Die grösser- und kleiner-Klammern <> weisen den Preprozessor an, nach der angegebenen Datei in bestimmten Suchpfaden zu suchen, die vom System her bekannt sind. Dies beinhaltet insbesondere die Standard-Bibliotheken sowie möglicherweise dem System hinzugefügte Bibliotheken.
Um selbst hergestellte Dateien mittels der `#include`-Direktive einzufügen, werden die Anführungs- und Schlusszeichen `""` benötigt. Diese weisen den Preprozessor an, zuerst im Verzeichnis der aktuellen Datei zu suchen.

Beispiel mit 3 Sourcecode-Dateien: main.c, bib.c, bib.h

main.c

```
#include <stdio.h>
#include "bib.h"

int main(void)
{
    int a=4,b;
    b=quadrat(a);
    printf("Quadrat von %d: %d\n",a,b);
    return 0;
}
```

bib.c

```
#include "bib.h"

int quadrat(int nr)
{
    return nr*nr;
}
```

bib.h

```
#ifndef BIB_H_INCLUDED
#define BIB_H_INCLUDED

int quadrat(int nr);

#endif // BIB_H_INCLUDED
```

16.4 weitere Präprozessor-Operatoren

String-Operator (#)

Da der Präprozessor Inhalte in Zeichenketten nicht ersetzt, ist folgende Präprozessor-Anweisung nicht möglich:

```
#define MESSAGE(X) "X"
```

Stattdessen ist hier der #-Operator anzuwenden:

```
#define MESSAGE(X) #X
```

Beispiel:

```
#define MESSAGE(X) printf("%s = %d\n",#X,X)

int main(void) {
      int i = 2;
      int counter = 12;
      int sum = 14;

      MESSAGE(i);
      MESSAGE(counter);
      MESSAGE(sum);

      return 0;
}
```

Ausgabe:

⇨

```
i = 2
counter = 12
sum = 14
```

Verbindungsoperator (##)

Möchte man Parameter verbinden, ist folgende Präprozessor-Anweisung nicht möglich:

```
#define CONCAT(X,Y) XY
```

Stattdessen ist hier der ##-Operator anzuwenden:

```
#define CONCAT(X,Y) X##Y
```

Beispiel:

```
#define STRTONUMBER(func, args) ato##func (args)

int main(){
  char str[]="123.456";
  printf("PI = %f\n", STRTONUMBER(f, str));
  printf("PI = %d\n", STRTONUMBER(i, str));
  return 0;
}
```

⇩

```
int main(){
  char str[]="123.456";
  printf("PI = %f\n", atof (str));
  printf("PI = %d\n", atoi (str));
  return 0;
}
```

⇨

Ausgabe:

```
PI = 123.456000
PI = 123
```