

Kapitel 9

Programmgrobstruktur

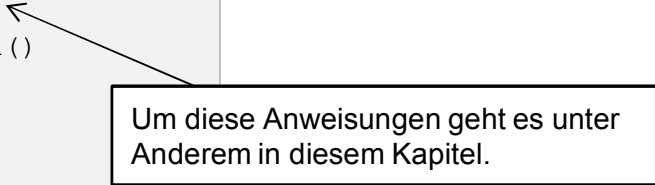
Der Preprocessor

Anweisungen, die am Zeilenanfang mit **#** beginnen, richten sich an den **Preprocessor**.

Der Preprocessor verarbeitet diese Anweisungen bevor der Compiler mit der Übersetzung des Programms beginnt. Der Compiler übersetzt dann den durch den Preprocessor vorverarbeiteten Quellcode.

```
# include <stdio.h>
# include <stdlib.h>

void main()
{
    ...
    ...
    ...
}
```



Um diese Anweisungen geht es unter Anderem in diesem Kapitel.

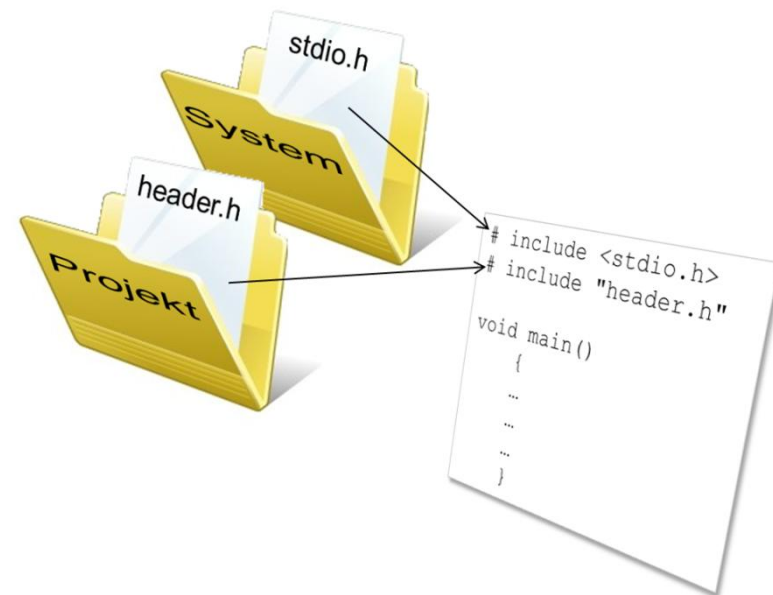
Der Preprocessor erzeugt keinen ausführbaren Code sondern führt nur Textersetzungen im Quellcode durch.

Include Anweisung

Mit einer Include-Anweisung können komplette Dateien vor der Übersetzung virtuell in den Quellcode eingefügt (includiert) werden. Üblicherweise handelt es sich dabei sogenannte Header-Dateien. Diese Dateien erkennt man an der Dateinamenserweiterung ".h".

System-Headerdateien sind Dateien, die mit dem Compiler oder mit speziellen System- oder Entwicklungskomponenten geliefert werden und auf dem Entwicklungsrechner bereits vorhanden sind. Diese Dateien liegen in speziellen Systemverzeichnissen, die der Entwicklungsumgebung bekannt sind.

Projekt-Headerdateien sind Headerdateien, die Sie in Ihrem Projekt selbst erstellen. Diese Dateien liegen zusammen mit den von Ihnen ebenfalls erstellten Quellcodedateien im Projektordner Ihres Projekts.

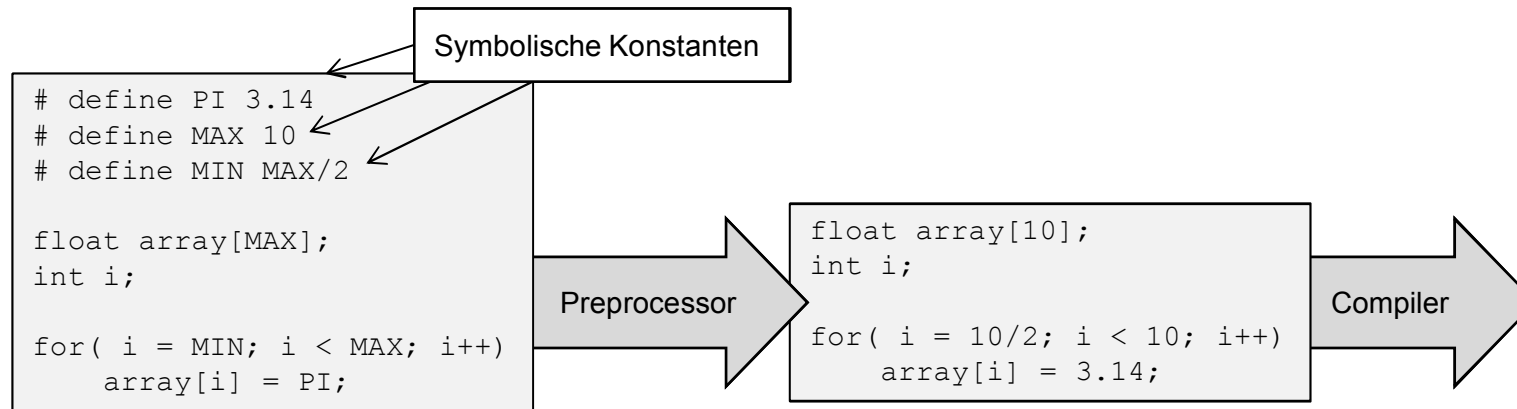


In einer Include-Anweisung werden System-Headerdateien in spitze Klammern (<...>) und Projekt-Headerdateien in Anführungszeichen ("...") gesetzt.

In Headerdateien stehen Informationen, die einer oder mehreren Quellcodedateien von zentraler Stelle aus einheitlich zur Verfügung gestellt werden sollen.

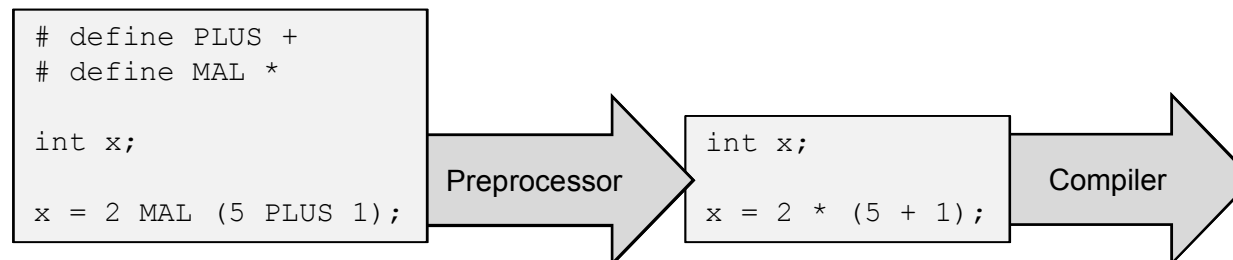
Symbolische Konstanten

Durch **symbolische Konstanten** können Werte, die an unterschiedlichen Stellen im Quellcode einheitlich verwendet werden sollen, an zentraler Stelle festgelegt und gepflegt werden.



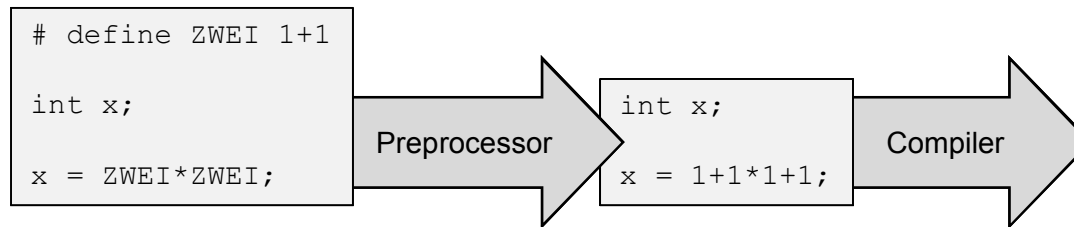
**Symbolische Konstanten sind keine Variablen
sondern nur Platzhalter für einen Ersatztext!**

Es können sehr allgemeine Ersetzungen durchgeführt werden. Wichtig ist, dass nach der Verarbeitung durch den Preprocessor gültiger Quellcode entsteht:



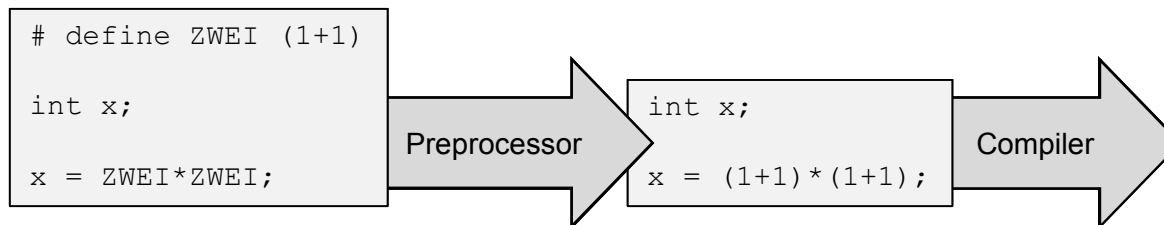
Auflösung symbolischer Konstanten

Bei der Auflösung von symbolischen Konstanten können unerwünschte Effekte auftreten:



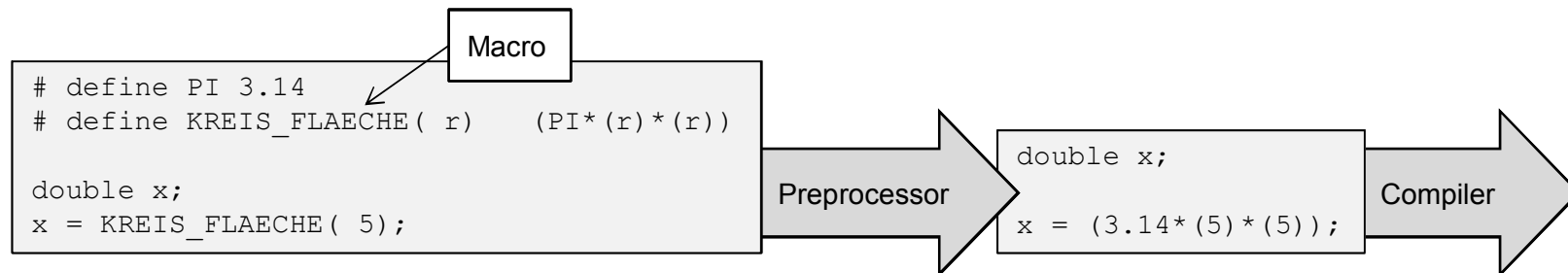
Ausdrücke werden nicht ausgewertet, vereinfacht oder ausgerechnet. Es findet eine reine Textersetzung statt.

Setzen Sie um Ausdrücke "Sicherheitsklammern", da Sie nicht wissen, in welchem Kontext die Auflösung erfolgt:



Preprocessor-Makros

Makros ermöglichen es, Textersetzungen über Parameter zu steuern:

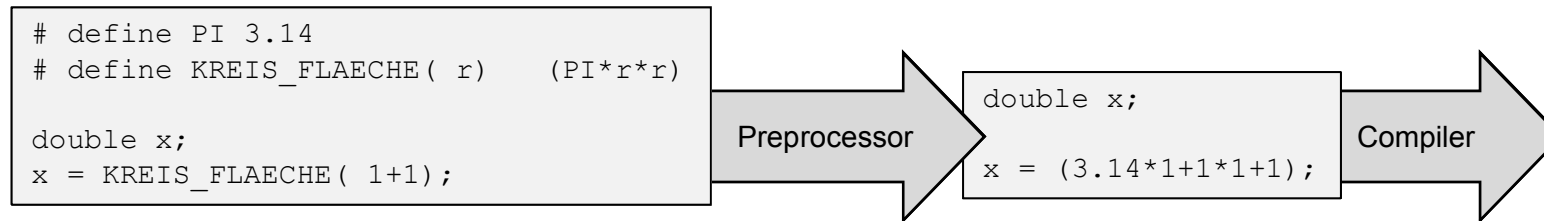


Bei der Ersetzung durch den Preprocessor werden keine Auswertungen, Vereinfachungen oder Berechnungen durchgeführt, auch hier handelt es sich um eine reine Textersetzung.

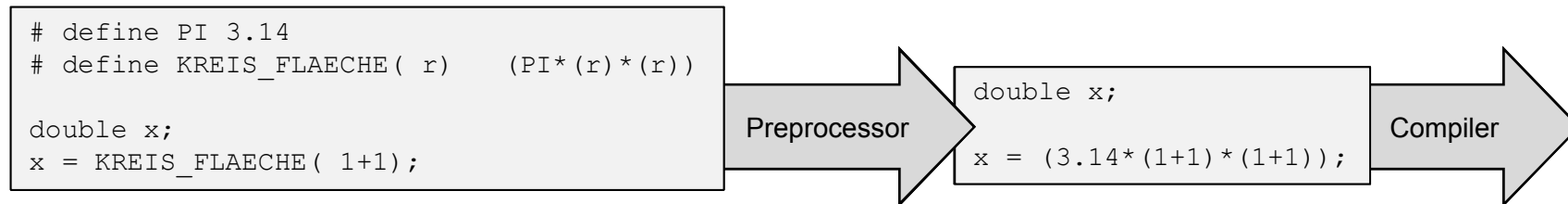
Makros sind keine Funktionen, sondern nur parametrisierte Platzhalter für einen Ersatztext!

Auflösung von Makros

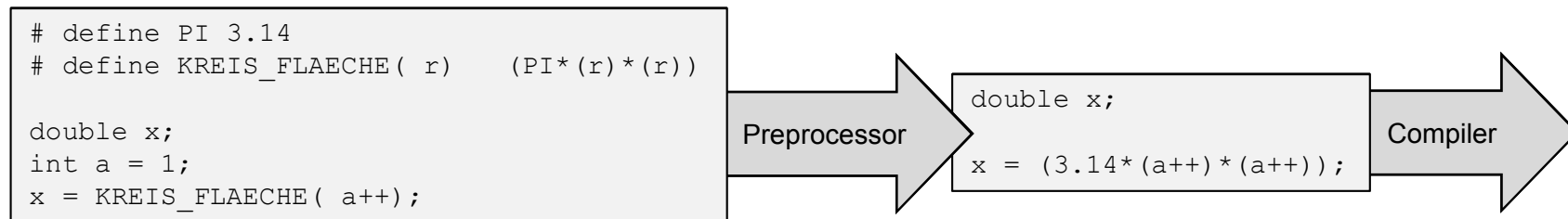
Bei der Auflösung von Makros kann es zu unerwünschten Effekten kommen:



Setzen Sie Klammern um Parameter, um ungewollte Effekte zu vermeiden.



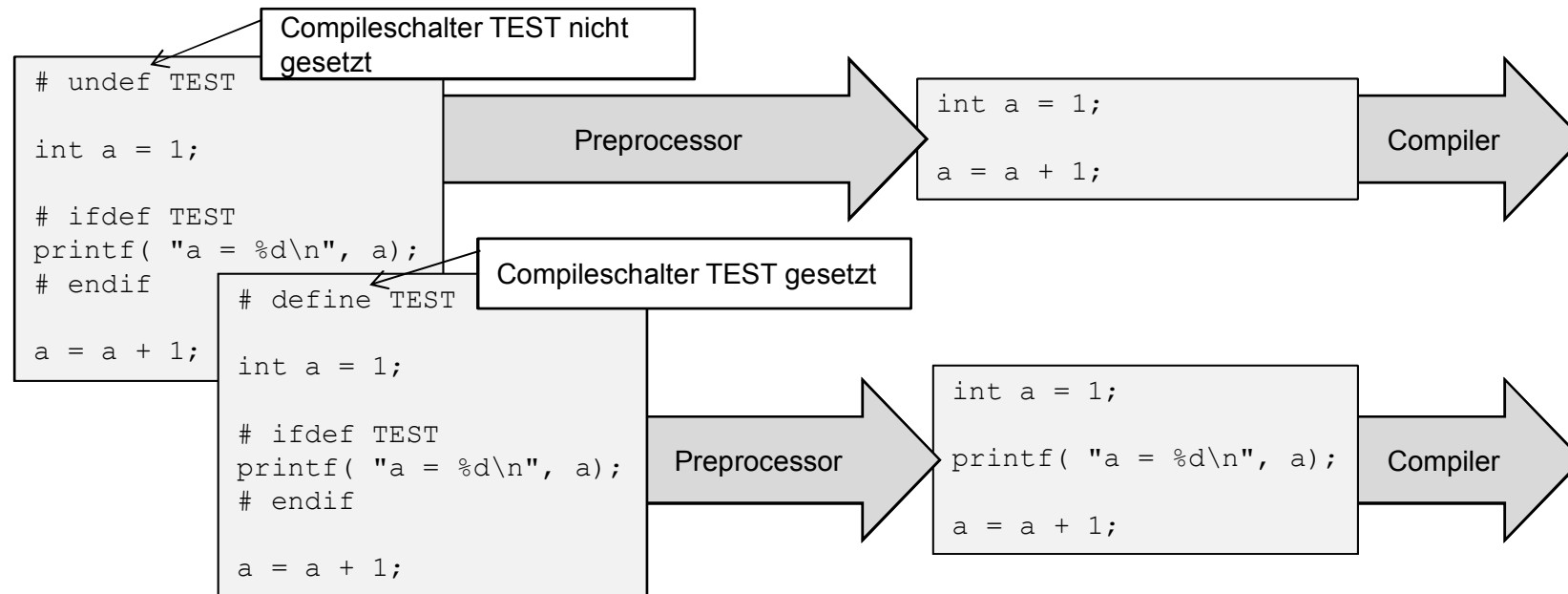
Achten Sie auf Seiteneffekte bei Formelausdrücken in Makros



Nach der Auflösung des Makros wird im letzten Beispiel `a` zweimal inkrementiert, was wahrscheinlich nicht beabsichtigt war.

Compileschalter

Durch **Compileschalter** können Teile des Codes von der Übersetzung ausgeschlossen werden oder verschiedene Varianten des Quellcodes aus einer Quelle erzeugt werden:



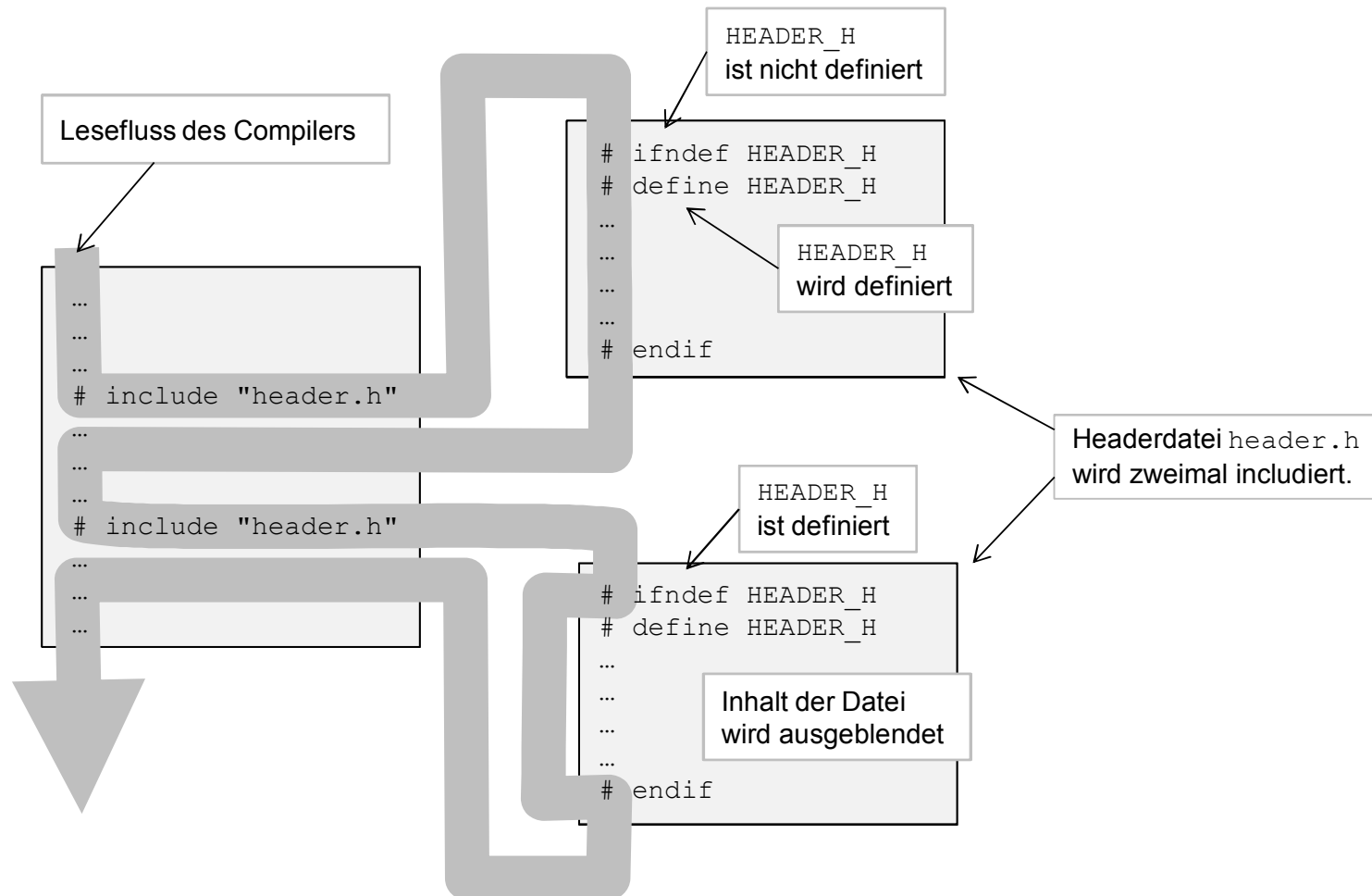
Wenn der Compileschalter TEST gesetzt ist, sind zusätzlich Prüfdrucke im Code vorhanden. Ist der Compileschalter nicht gesetzt, sind die Prüfdrucke nicht vorhanden.

Compileschalter sind keine if-Anweisungen

Eine if-Anweisung wird zur Laufzeit ausgeführt, ein Compileschalter wird durch den Preprocessor aufgelöst und ist zur Laufzeit nicht mehr im Code vorhanden.

Schutz vor rekursivem Include

Mit Compileschaltern kann verhindert werden, dass eine Headerdatei mehrfach inkludiert wird:



Wird die Headerdatei erstmalig inkludiert ist der Compileschalter noch nicht gesetzt. Die Headerdatei ist also für den Compiler sichtbar. In der Headerdatei wird dann der Compileschalter gesetzt, sodass die Datei, bei weiteren Includes ausgeblendet wird.

Ein kleines Projekt mit drei Dateien

maximum.h

```
# ifndef MAXIMUM_H
# define MAXIMUM_H

extern int absolute_maximum;
extern int maximum( int a, int b);

# endif
```

Hier werden die globale Variable `absolute_maximum` und die Funktion `maximum` **deklariert**.

maximum.c

```
# include <limits.h>
# include "maximum.h"

int absolute_maximum = INT_MIN;

int maximum( int a, int b)
{
    int max;

    if( a > b)
        max = a;
    else
        max = b;
    if( max > absolute_maximum)
        absolute_maximum = max;
}
```

Hier werden die globale Variable `absolute_maximum` und die Funktion `maximum` **definiert**.

main.c

```
# include <stdio.h>
# include <stdlib.h>
# include "maximum.h"

void main()
{
    int a, b, c;

    a = maximum( 1, -5);
    b = maximum( -10, 17);
    c = absolute_maximum;
    printf( "%d\n", c);
}
```

Hier werden die globale Variable `absolute_maximum` und die Funktion `maximum` **verwendet**.

`INT_MIN` ist eine symbolische Konstante, die in `limits.h` als der kleinstmögliche `int`-Wert festgelegt ist.

Headerdateien enthalten Deklarationen, die in unterschiedlichen Quelldateien konsistent verwendet werden sollen. Headerdateien enthalten keine Definitionen und keinen Code, sie ermöglichen nur die Aufteilung von Definitionen und Code auf mehrere Dateien.