Technisches Programmieren in C++

Der C++-Präprozessor

Karl Riedling
Institut für Sensor- und Aktuatorsysteme

Der C++-Präprozessor

- Die Funktionen des C++-Präprozessors
- Präprozessor-Direktiven

Der C++-Präprozessor

- **Die Funktionen des C++-Präprozessors**
- Präprozessor-Direktiven

Der C++-Präprozessor

- Die Funktionen des C++-Präprozessors
  - **Allgemeines**
  - Die Rolle des Präprozessors in C++

Die Funktionen des C++-Präprozessors

- C++- (und Standard-C-) Programme werden einer Vorverarbeitung im *Präprozessor* unterworfen.
- Dieser modifiziert den *Quellcode* als *Textprozessor*:
  - Ersatz symbolischer Namen durch andere oder durch Konstanten;
  - Expansion von *Makros*;
  - Einbinden anderer Quelldateien;
  - Ausblenden von Teilen des Quellcodes in Abhängigkeit vom Ergebnis der Prüfung einer Bedingung;
  - Einstellung gewisser (implementierungsspezifischer) Compilerparameter.

Die Funktionen des C++-Präprozessors

- Ziel dieser Vorverarbeitung ist es, Quellprogramme einfacher, leichter wartbar und portabler zu gestalten durch:
  - Festlegung oder Änderung konstanter Programmparameter an einer leicht zugänglichen Stelle;
  - Kompaktere Darstellung oft gebrauchter Code-Sequenzen durch Makros;
  - Replizieren gewisser Deklarationen oder Definitionen mit verringerter Fehleranfälligkeit in einer beliebigen Anzahl von Übersetzungseinheiten;

Die Funktionen des C++-Präprozessors

- Übersetzung von Programmen mit unterschiedlichen Parametern oder auch in unterschiedlichen Teilen.
- Adaptieren eines Programms an möglichst wenigen Stellen mit möglichst globaler Wirkung.

- Präprozessor-Direktiven werden ausschließlich vom *Präprozessor* interpretiert und aus seiner Ausgabedatei entfernt.
- Sie müssen grundsätzlich mit einem "#" beginnen, das das erste nicht-leere Zeichen der Zeile sein muss.

Die Funktionen des C++-Präprozessors

- Präprozessor-Befehle müssen in *einer* Zeile stehen.
- Sie können sich dann über *mehrere* Zeilen erstrecken, wenn das *letzte* Zeichen in der Zeile unmittelbar vor dem Zeilenvorschub ein "\" ist.
- Sie können Kommentare in "`/*  */`" oder nach "`//`" enthalten.
- Die folgenden Präprozessor-Direktiven sind definiert:

`#define`	`#endif`	`#ifdef`	`#line`
`#elif`	`#error`	`#ifndef`	`#pragma`
`#else`	`#if`	`#include`	`#undef`

Die Funktionen des C++-Präprozessors

- Der Präprozessor *expandiert manifeste Konstanten* und *Makros* in jenen Teilen des Programms, die *keine* Präprozessor-Direktiven sind.
- Manifeste Konstanten und Makros werden mit der `#define`-Direktive definiert.
- Mit `#define` definierte Namen, die für eine Konstante stehen, sind *manifeste Konstante*; solche, die für einen Ausdruck oder Befehl stehen, sind *Makros*.
- *Makros* können eine beliebige Anzahl von Argumenten haben.

Der C++-Präprozessor

- Die Funktionen des C++-Präprozessors
  - Allgemeines
  - **Die Rolle des Präprozessors in C++**

Die Rolle des Präprozessors in C++

- In Standard-C unerlässliche Funktionen des Präprozessors wurden in C++ durch Funktionen des Compilers ersetzt oder dupliziert:

Standard-C:	**C++:**
manifeste Konstanten (mit `#define` definiert)	Konstante (mit dem Schlüsselwort `const` deklariert)
Makros	`inline`-Funktionen

Die Rolle des Präprozessors in C++

- Vorteile der C++-Lösungen:
  - Möglichkeit einer strikten Typenprüfung;
  - Verfügbarkeit von Konstanten oder Enumeratoren mit ihren symbolischen Namen innerhalb eines symbolischen Debuggers;
  - keine Probleme mit den Nebenwirkungen von Makro-Aufrufen.

Der C++-Präprozessor

- Die Funktionen des C++-Präprozessors
- **Präprozessor-Direktiven**

Der C++-Präprozessor

- Präprozessor-Direktiven
  - **Die `#define`-Direktive**
  - Operatoren für `#define`
  - Die `#undef`-Direktive
  - Vordefinierte Makros
  - Die `#include`-Direktive
  - Bedingte Übersetzung
  - Sonstige Direktiven

Die `#define`-Direktive

- Definition eines symbolischen Namens für eine Konstante, einen Ausdruck oder einen Befehl (generell, für ein *Token*):

  `#define` ***`Name Token-String`***
- Alle auf diese Definition folgenden Vorkommen von ***`Name`*** werden durch ***`Token-String`*** ersetzt, wenn ***`Name`*** tatsächlich ein *Token* bildet. Dies ist *nicht* der Fall
  - wenn ***`Name`*** *Teil* eines anderen Namens ist;
  - wenn ***`Name`*** innerhalb eines Strings vorkommt;
  - wenn ***`Name`*** innerhalb eines Kommentars steht.

Die `#define`-Direktive

- Es ist zulässig, *keinen Token-String* in einer `#define`-Direktive anzugeben; in diesem Fall werden alle Vorkommen von ***`Name`*** entfernt. Trotzdem gilt ***`Name`*** aber als definiert (und wird von den Direktiven `#if defined` oder `#ifdef` als existent betrachtet).
- *Formale Argumente* eines Makros, die beliebig oft in ***`Token-String`*** vorkommen können, können *unmittelbar anschließend* an ***`Name`*** (ohne Leerzeichen) in Klammern und durch Kommas getrennt angeführt werden.

Die `#define`-Direktive

- Funktionsartige Makros (wie `max(a, b)`) werden überall dort im Programmcode expandiert, wo ihrem Namen eine Parameterliste in Klammern folgt. Jedes formale Argument wird im Makro durch das korrespondierende aktuelle Argument ersetzt.
- Eine weitere Definition desselben Namens mit `#define` bewirkt eine Fehlermeldung, ausgenommen, beide Definitionen sind identisch.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die `#define`-Direktive
  - **Operatoren für `#define`**
  - Die `#undef`-Direktive
  - Vordefinierte Makros
  - Die `#include`-Direktive
  - Bedingte Übersetzung
  - Sonstige Direktiven

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Operatoren für `#define`
    - **Der "Stringmacher"-Operator "#"**
    - Der "Tokenverbindungs"-Operator "##"

Der "Stringmacher"-Operator "#"

- Dieser Operator kann im "Körper" eines Makros einem formalen Argument vorangestellt werden.
- Bei der Expansion des Makros wird das aktuelle Argument an dieser Stelle des Makros in einen String umgewandelt (also mit "'" "eingeklammert").
- Leerzeichen vor dem ersten und nach dem letzten *Token* des Strings werden ignoriert.
- Wenn im aktuellen Argument die Zeichen "'" oder "\" vorkommen, wird ihnen ein "\" vorangestellt.
- Das folgende Makro gibt das den Namen und den Wert einer Variablen oder eines Ausdrucks (von fundamentalem Typ) aus:

Der "Stringmacher"-Operator "#"

```
#include <iostream.h>
#define ZeigeWert(x) cout << #x " = " \
  << (x) << endl;              // Fortsetzungszeile!
int main ()
{
  int i = 13;
  double x = 3.141592;
  ZeigeWert (i);
  ZeigeWert (x);
  ZeigeWert (i * x);
}
```


Der "Stringmacher"-Operator "#"

- Der Präprozessor macht aus `main()`:

```
int main ()
{
  int i = 13;
  double x = 3.141592;
  cout << "i" " = " << ( i ) << endl;  ;
  cout << "x" " = " << ( x ) << endl;  ;
  cout << "i * x" " = " << ( i * x ) << endl;  ;
}
```

- Bei seiner Ausführung gibt das Programm aus:

```
i = 13
x = 3.14159
i * x = 40.8407
```


Der C++-Präprozessor

- Präprozessor-Direktiven
  - Operatoren für `#define`
    - Der "Stringmacher"-Operator "#"
    - **Der "Tokenverbindungs"-Operator "##"**

Der "Tokenverbindungs"-Operator "##"

- Dieser Operator erlaubt das Verbinden zweier getrennter *Tokens* zu einem einzigen.
- Die sich dabei ergebenden *Tokens* können ihrerseits die Namen von Makros oder manifesten Konstanten sein, die entsprechend umgesetzt werden.
- Da "##" zwei benachbarte *Tokens* verbinden muss, kann dieser Operator weder das erste noch das letzte *Token* in der Definition des Makros sein.
- Leerzeichen vor oder nach "##" sind zulässig.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die `#define`-Direktive
  - Operatoren für `#define`
  - **Die `#undef`-Direktive**
  - Vordefinierte Makros
  - Die `#include`-Direktive
  - Bedingte Übersetzung
  - Sonstige Direktiven

Die `#undef`-Direktive

- Die `#undef`-Direktive hebt die Definition eines mit `#define` definierten Namens wieder auf.
- Bei der Aufhebung der Definition eines funktionsartigen Makros mit `#undef` ist *nur der Name* des Makros anzugeben und *keine Argumente*!
- `#undef` kann auch verwendet werden, um Namen ungültig zu machen, die Elemente der Sprache selbst sind.
- Die Aufhebung der Definition eines Namens kann verwendet werden, um implementierungsspezifische Schlüsselworte zu entfernen oder zu ändern.

Die `#undef`-Direktive

- Reservierte Schlüsselworte haben keine besondere Bedeutung für den Präprozessor und werden genauso behandelt wie irgendein anderer Name.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die `#define`-Direktive
  - Operatoren für `#define`
  - Die `#undef`-Direktive
  - **Vordefinierte Makros**
  - Die `#include`-Direktive
  - Bedingte Übersetzung
  - Sonstige Direktiven

Vordefinierte Makros

- Einige Makros sind standardmäßig in jeden C- oder C++-Compiler eingebaut; sie können ohne vorherige explizite Definition verwendet werden.
- Weitere teils implementierungsspezifische vordefinierte Makros geben Type und Version des Compilers, Prozessor-Typ und Betriebssystem, für welche das Programm übersetzt wurde, und ähnliche Informationen an.

Vordefinierte Makros

`__cplusplus`	Wird von einem C++-Compiler definiert, wenn er ein C++-Programm übersetzt.
`__DATE__`	Übersetzungsdatum der aktuellen Übersetzungseinheit als String, z.B. "`Oct 07 2005`".
`__FILE__`	Name der Quelldatei als String-Konstante.
`__LINE__`	Aktuelle Zeilennummer in der Quelldatei als `int`-Konstante.
`__STDC__`	Mit dem Wert `1` definiert, wenn der Compiler den ANSI-C-Standard voll erfüllt.
`__TIME__`	Übersetzungszeit der aktuellen Übersetzungseinheit als String-Konstante.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die `#define`-Direktive
  - Operatoren für `#define`
  - Die `#undef`-Direktive
  - Vordefinierte Makros
  - **Die `#include`-Direktive**
  - Bedingte Übersetzung
  - Sonstige Direktiven

Die `#include`-Direktive

- Die Direktive "`#include`" bewirkt, dass der Präprozessor den Inhalt der mit `#include` angegebenen Datei an der Stelle der `#include`-Direktive einbindet.
- Für den eigentlichen Compiler erscheint der Inhalt der *Include*-Datei (und weiterer von dieser mit `#include` eingebundener Dateien) als integraler Bestandteil der aktuellen Übersetzungseinheit.
- Anwendung von *Include*-Dateien ist üblicherweise das Einbinden der Definitionen von Konstanten oder Makros, von Definitionen von Funktionen (*Prototypen*) und Klassen oder von externen Variablen und Objekten.

Die `#include`-Direktive

- Dies ist in den folgenden Fällen zweckmäßig:
  - Wenn die Definitionen von Bibliotheksfunktionen benötigt werden, die mit dem Compiler ausgeliefert werden;
  - Wenn ein Programm aus mehreren Übersetzungseinheiten besteht, und diese Übersetzungseinheiten Zugriff auf gemeinsame Konstanten oder Makros, auf globale Daten oder auf die Definition benutzerdefinierter Datentypen benötigen.

Die `#include`-Direktive

- Gelegentlich ist es zweckmäßig, auch Teile des Programmcodes mit `#include` einzubinden, beispielsweise, wenn mehrere Programme auf gemeinsame Code-Module zugreifen müssen, aber die Erstellung einer Funktionsbibliothek nicht sinnvoll oder möglich ist.
- Es existieren zwei Varianten der `#include`-Direktive:
  - `#include "`*`Dateiname`*`"` und
  - `#include <`*`Dateiname`*`>`

Die `#include`-Direktive

- Diese beiden Varianten unterscheiden sich durch die Strategie, mit der der Präprozessor nach der einzubindenden Datei sucht:
  - Bei der Form "`#include "`*`Dateiname`*`"`" sucht der Präprozessor (in der angegebenen Reihenfolge)
    - im aktuellen Verzeichnis;
    - in den Verzeichnissen, die mit einer Umgebungsvariablen definiert wurden, in der Reihenfolge ihrer Definition;
    - in den Verzeichnissen, die mit dem Parameter `-I`*`Verzeichnis`* beim Aufruf des Compilers (oder Präprozessors) spezifiziert wurden.

Die `#include`-Direktive

  - Bei der Form "`#include <`*`Dateiname`*`>`" sucht der Präprozessor
    - in den Verzeichnissen, die mit einer Umgebungsvariablen definiert wurden, in der Reihenfolge ihrer Definition;
    - in den Verzeichnissen, die mit dem Parameter `-I`*`Verzeichnis`* beim Aufruf des Compilers (oder Präprozessors) spezifiziert wurden;
    - *nicht* im aktuellen Verzeichnis.

Die #include-Direktive

- Für Dateien, die mit #include eingebunden werden sollen, hat sich die Dateitype ".h" (für *Header*) eingebürgert. Es ist die Verwendung dieser Konvention aber keinesfalls zwingend.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die #define-Direktive
  - Operatoren für #define
  - Die #undef-Direktive
  - Vordefinierte Makros
  - Die #include-Direktive
  - **Bedingte Übersetzung**
  - Sonstige Direktiven

Bedingte Übersetzung

- Präprozessor-Direktiven erlauben eine *bedingte Übersetzung*: Teile eines Programms werden in Abhängigkeit vom Definitions-Status eines Namens (definiert oder nicht) oder dem ihm zugeordneten Wert verwendet oder übersprungen.
- Zwei Direktiven — #if und #elif ("*else if*") — prüfen den *Wert* eines *Ausdrucks* (der zum Zeitpunkt der Bearbeitung des Programms durch den Präprozessor definiert sein muss).

Bedingte Übersetzung

- Wenn dieser Wert von Null verschieden ist, wird der Teil des Quellcodes bis zur nächstgelegenen #else-, #elif- oder #endif-Direktive an den Compiler weitergeleitet, anderenfalls ausgeblendet.
- Der *Definitions-Status* eines Namens kann mit den Direktiven #ifdef und #ifndef sowie mit #if und dem Präprozessor-Operator defined getestet werden. Nur wenn die Bedingung erfüllt ist, wird der Teil des Programms bis zur nächsten #else-, #elif- oder #endif-Direktive bearbeitet.

Bedingte Übersetzung

- Jedem #if, #ifdef oder #ifndef muss genau *ein* #endif entsprechen; dazwischen dürfen beliebig viele #elif-Direktiven und — als letzte — maximal eine #else-Direktive liegen.
- Innerhalb eines durch #if, #ifdef oder #ifndef einerseits und #endif andererseits "aufgespannten" Blocks dürfen weitere #if – #endif-Blöcke liegen; jede #elif- oder #else-Direktive in diesen verschachtelten Blöcken bezieht sich auf das nächstgelegene vorhergehende #if.

Bedingte Übersetzung

- Es gelten die folgenden Einschränkungen für die mit #if oder #elif verwendeten Ausdrücke:
  - Sie müssen ganzzahlig sein und dürfen nur ganzzahlige Konstanten, Zeichenkonstanten und den Operator defined enthalten.
  - Die Verwendung von *Type Casts* sowie des Operators sizeof ist unzulässig.
  - Numerische Werte werden intern als long oder unsigned long dargestellt.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Die `#define`-Direktive
  - Operatoren für `#define`
  - Die `#undef`-Direktive
  - Vordefinierte Makros
  - Die `#include`-Direktive
  - Bedingte Übersetzung
  - **Sonstige Direktiven**

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Sonstige Direktiven
    - **Die Direktive `#line`**
    - Die Direktive `#error`
    - Die `#pragma`-Direktiven

Die Direktive `#line`

- Die vordefinierten Makros `__FILE__` und `__LINE__` repräsentieren den Namen der aktuellen Datei und die Nummer der aktuellen Zeile in dieser Datei.
- Diese Information kann mit der Direktive `#line` gezielt verändert werden.
- Die Zeilennummern werden ausgehend von der mit `#line` angegebenen Nummer weitergezählt.
- Für diese Direktive sind die folgenden beiden Formen möglich:
  - `#line` *`GanzeZahl`* und
  - `#line` *`GanzeZahl`* `"`*`Dateiname`*`"`

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Sonstige Direktiven
    - Die Direktive `#line`
    - **Die Direktive `#error`**
    - Die `#pragma`-Direktiven

Die Direktive `#error`

- Mit der Direktive `#error`, der ein beliebiger String folgen kann, wird die Übersetzung eines Programms abgebrochen, wobei der mit der Direktive angegebene String ausgegeben wird.
- Typisch wird diese Direktive dazu verwendet, um nach Prüfung eines vordefinierten Makros die Übersetzung bei Bedarf abzubrechen:

```
#if defined __unix__
#error Tu nix mit UNIX!
#endif
```


Die Direktive `#error`

- Mit einem Unix-basierten C++-Compiler (z.B. GNU C++) übersetzt, bewirkt dieses Programm tatsächlich einen Abbruch und die Ausgabe des Kernspruchs:

```
7_02_09.cc:5: #error Tu nix mit UNIX!
```

- Hingegen wird das selbe Programm mit einem Windows-basierten Compiler, für den der Name `__unix__` nicht definiert ist, fehlerfrei übersetzt.

Der C++-Präprozessor

- Präprozessor-Direktiven
  - Sonstige Direktiven
    - Die Direktive `#line`
    - Die Direktive `#error`
    - **Die `#pragma`-Direktiven**

Die `#pragma`-Direktiven

- `#pragma`-Direktiven haben grundsätzlich die Form
  `#pragma` *`Token-String`*
- Sie dienen dazu, um implementierungsspezifische Anpassungen der Funktionen des Compilers vorzunehmen.
- Im Gegensatz zu Befehlszeilen-Parametern des Compilers, die immer die gesamte Übersetzungseinheit beeinflussen, können sie auch lokale Veränderungen des Verhaltens des Compilers in Teilen einer Übersetzungseinheit erlauben.