RLP 2005 – Handwerkszeug

Bengen, Mallon, Röhricht

Fakultät für Informatik
IPD Goos

10. Mai 2005

Grundlagen

Der C Präprozessor

- ist ein einfaches Textersetzungssystem
- hat Mechanismen zur bedingten Abarbeitung
- arbeitet zeilenweise
- ist nicht Turing-mächtig

#include

#include <datei>

- Ersetzt die Zeile durch den Inhalt der Datei
- Sucht in Systempfaden

#include "datei"

- Suchpfade sind implementierungsabhängig
- Zumeist Pfade, die per `-I` übergeben wurden und aktuelles Verzeichnis

#define

#define FALSE 0

- Definiert ein Makro mit Namen `FALSE`
- Alle späteren Vorkommnisse von `FALSE` werden durch `0` ersetzt
- Linke Seite besteht aus genau einem Wort
- Rechte Seite ist beliebig

#define BLA

- Spezialfall: Leeres Makro

#define

Ersetzungen sind nicht selbst-rekursiv

#define bla bla

- ersetzt `bla` durch `bla`
- Ersetzung terminiert damit

#define blub bli
#define bli blub

- ersetzt `blub` durch `bli`
- ersetzt `bli` durch `bli`

#define mit Parametern

```
#define EINS_MEHR(x) ((x) + 1)
#define SUMME(x, y) ((x) + (y))
```

- Makros können beliebig viele Parameter haben
- Makros sind nicht überladbar

```
#define HALLO() printf("Hallo, Welt")
```

- Spezialfall: Leere Parameterliste

#undef

#undef BLA

- Löscht ein zuvor definiertes Makro (hier BLA)

#if

```
#if BEDINGUNG
/* wahr */
#else
/* falsch */
#endif
```

- Wertet die Bedingung aus und verarbeitet nur entsprechenden Abschnitt weiter

#if

Bedingungen können bestehen aus

- Vergleichsoperatoren
- Arithmetischen Operatoren
- Bitweise Operatoren
- Logische Operatoren
- Makros
- defined()

#ifdef/#ifndef

#ifdef MAKRO

- Kurzform von #if defined(MAKRO)

#ifndef MAKRO

- Kurzform von #if !defined(MAKRO)

#elif

```
#if BEDIGUNG1
#elif BEDINGUNG2
#endif
```

- Kurzform von

```
#if BEDIGUNG1
#else
  #if BEDINGUNG
  #endif
#endif
```

#warn/#error

#warn Hoffenentlich geht das gut...

- Erzeugt eine Warnung

#error Mein Leben!

- Bricht die Verarbeitung mit einem Fehler ab

# #

#define MAGIE(x) #x

- Verwandelt den Parameter in eine Zeichenkette
- Beispiel: MAGIE(Hallo) wird zu "Hallo"

##

#define MAGIE(x) x ## _Welt

- Verbindet zwei Token zu einem
- Beispiel: MAGIE(Hallo) wird zu Hallo_Welt

Tücken

Tücken des Präprozessors

Operatorpräzedenz

```
#define SECHS 1 + 5
#define NEUN 8 + 1
#define ANTWORT SECHS * NEUN
```

Operatorpräzedenz

```
#define SECHS (1 + 5)
#define NEUN (8 + 1)
#define ANTWORT (SECHS * NEUN)
```

- Lösung: Vollständige Klammerung
- Insbesondere Parameter klammern

Assoziativität von Schlüsselwörtern

```
#define DEBUG(x) if (debug > 0) printf("%s", x)
if (i < 3)
  DEBUG("i kleiner 3");
else
  f(i);
```

Assoziativität von Schlüsselwörtern

```
if (i < 3)
  if (debug > 0) printf("%s", "i kleiner 3");
else
  f(i);
```

Assoziativität von Schlüsselwörtern

#define DEBUG(x) do { if (debug > 0) printf("%s", x); } while(0)

Seiteneffekte

```
#define MIN(a, b) ((a) < (b) ? (a) : (b))
MIN(i++, j)
```

Seiteneffekte

((i++) < (j) ? (i++) : (j))

- i++ wird evtl. 2 mal ausgewertet

Anwendung

Beispiele für Verwendung des Präprozessors

Konstanten

```
#define PI 3.1415926535
#define ANTWORT 42
```

Abkürzungen

```
#define POWER_OF_2(n) (((n) & ((n) - 1)) == 0 && (n) != 0)
```

Wächter

```
#ifndef HEADER_H
#define HEADER_H
  /* Deklarationen */
  typedef unsigned int uint;
#endif
```

Auskommentieren

```
#if 0
/* Beliebiges */
#endif
```

Platformspezifika

```
#if defined(__GNUC__)
  /* Kram für den GCC */
#elif defined(_MSVC_VER)
  /* Kram für MSVC */
#endif
```

Altlasten

```
int* __error(void);
#define errno (*__error())
```

Befehle im Eigenbau

```
#define TRAVERSIERE(x) for (; (x) != NULL; (x) = (x)->succ)
TRAVERSIERE(liste)  summe += liste->wert;
```

assert()

```
void assert(const char* condition, const char* file, int line);
#define assert(x) ((x) ? (void)0 : assert(#x, __FILE__, __LINE__))
```

assert()

dinge.c, Zeile 23:
assert(zeiger != NULL);

- wird zu

((zeiger != NULL) ? (void)0 : assert("zeiger != NULL", "dinge.c", 23);

enum mit Namen

```
enum {
  EINS = 1,
  ZWEI = 2,
  ANTWORT = 42
};
```

enum mit Namen

x.def:
x(EINS, 1)
x(ZWEI, 2)
x(ANTWORT, 42)

enum mit Namen

```
enum {
  #define x(a, b) a = b,
  #include "x.def"
  #undef x
};
const char* EnumName(int n) {
  switch (n) {
    #define x(a, b) case b: return #a;
    #include "x.def"
    #undef x
  }
}
```

enum mit Namen

```
enum {
EINS = 1,
ZWEI = 2,
ANTWORT = 42,
};
const char* EnumName(int n) {
  switch(n) {
case 1: return "EINS";
case 2: return "ZWEI";
case 42: return "ANTWORT";
  }
}
```

Was ist ein Versionsverwaltungssystem?

- »Version control is the art of managing changes to information«[1]
- Software zur Versionierung und Verwaltung von Information (Quelltexte bspw.)
- Erfassen von Änderungen mit Zeitstempel und Benutzerkennung
- Ermöglicht leichte Zugänglichkeit zur aktuellsten Version und das Rückspielen einer alten Version
- Zentrales Archiv (Repository) meist in Datenbank oder eigenem Dateiformat
- Lokale Arbeitskopie des Verzeichnisbaums (Aktualisierung der Arbeitskopie mittels checkout)
- Gleichzeitige Entwicklung mehrerer Zweige (Branches)

[1] Ben Collins-Sussman u.a.: Version Control with Subversion.

RCS – Revision Control System

- Urvater der Versionskontrollsysteme
- Walter F. Tichy (Prof. Universität Karlsruhe)
- Besteht aus mehreren Programmen
- Jede verwaltete Datei erhält eigene Revisionsdatei
- Zusatzinformationen: Autor, Erstellungszeitpunkt, Locker, Status, Kommentare zu Änderungen
- Kann nur Dateien verwalten, keine Verzeichnisse
- Keine Umbenennung oder Verschiebung von Dateien oder Verz. möglich

CVS – Concurrent Versioning System

- Bei vielen großen Opensource Projekten im Einsatz (Gnome, Mozilla)
- Von vielen OS-Plattformen unterstützt, bspw. Sourceforge.net
- Sehr gut dokumentiert
- Viele grafische Applikationen und Add-Ons
- Aber limitiert:
  - Unterstützt kein Umbenennen von Dateien oder Verzeichnissen
  - Probleme mit binären Dateien (Kein Diff möglich)
  - Kein verteiltes System
  - Keine atomaren Commits

Subversion

- Sollte CVS ersetzen und dessen Schwächen beseitigen
- Orientiert sich an CVS' Konventionen (Kommandos)
- Unterstützt u.a.:
  - Kopieren und Umbenennen von Dateien und Verz.
  - Echte atomare Commits
  - Effizienter Umgang mit Binärdateien
  - Verfügbarkeit als Apache2-Modul (WebDAV mit Benutzerauth., Verschlüsselung, Komprimierung, etc.)
  - Nativer Win32 Client und Server
- Nutzt die Berkeley DB
- Limitierte Unterstützung fürs ≫Mergen≪
- Relativ ressourcenhungrig (vor allem bei großen Operationen)
- Sehr gut dokumentiert

Beispiel zu Subversion

1. Repository erstellen: `svnadmin create /home/user/svnroot`
2. Bestehendes Projekt importieren: `svn import /tmp/projekte file:///home/user/svnroot -m 'Initialer Import'`
3. Check-out eines Repos: `svn checkout svn://anonsvn.kde.org/home/kde/trunk/`
4. Änderungen einpflegen: `svn commit datei`
5. Arbeitskopie aktualisieren: `svn update`
6. Änderungen zw. Revision 1 und 2 betrachten: `svn diff -r 1:2`
7. Änderungen einpflegen: `svn {add | delete | copy | move}`

Was gibt es noch?

- GNU Arch
  - Ursprünglich Sammlung von Shell-Skripten, heute in C, intelligentes Mergen, verteiltes System
- OpenCM
  - Unterstützt kryptografische Authentifikation, nicht verteilt, eigenes Protokoll
- Monotone
  - Starke Kryptographie (SHA1 und RSA), verteilter Ansatz mit zentralisierten Änderungssätzen
- Bitkeeper
  - Kommerziell, closed source, ehemals bei Teilen des Linux Kerns eingesetzt, viele Features
- Guter Vergleich vieler Systeme: *http://better-scm.berlios.de/comparison/comparison.html*

diff

- diff vergleicht zwei Textdateien und gibt Ergebnis nach stdout
- Erkennt neu eingefügte Textabschnitte automatisch
- Erläuterungen (unified output format):
  - `--- datei.c` → originale Datei
  - `+++ neueDatei.c` → ≫neue≪ Datei
  - `@@ -1,13 +1,12 @@` → bedeutet, dass die folgenden 13 Zeilen ab Zeile 1 der Originaldatei mit den 12 Zeilen ab Zeile 1 der anderen Datei ersetzt wurden
  - `- bla` → wurde gelöscht
  - `+ blub` → wurde hinzugefügt
- `-q` teilt lediglich mit, ob sich zwei Dateien unterscheiden; `-u` erstellt ein Diff im Unified Output Format (für Patch-Dateien); `-B` ignoriert Leerzeilen; ...
- Erzeugt Patch-Dateien ...

patch

- Wendet eine Diff-Datei als Patch an
- Beispiel: Linux Kernel Patches mit den Veränderungen zur letzten Version
- Nicht für Binärdateien geeignet
- Einfachster Fall: `patch < patchDatei`
- Da ganze Pfade in Patch-Datei einthalten sein können, muss darauf geachtet werden, dass sich die Verzeichnisebenen gleichen: `-p<Zahl>` gibt an, wieviele Unterverzeichnisse in der Patch-Datei abgeschnitten werden sollen.
- Bevor patch eine Datei modifiziert, legt es eine Sicherheitskopie mit der Endung .orig an

Ein Fallbeispiel(1)

Die Originaldatei datei.c

```
#include <stdio.h>

int main() {
  int zahl = 42;
  char wort[255];

  printf("\nWort eingeben: ");

  scanf("%s", wort);
  printf("Wort ist: %s\n", wort);

  return 0;
}
```

Ein Fallbeispiel(2)

```
#include <stdio.h>

int main() {
  char wort[255];

  printf("Wort eingeben: ");

  scanf("%s", wort);
  printf("Wort war: %s\n", wort);

  return 0;
}
```

Ein Fallbeispiel(3)

```
martin@bart:~$ diff -u datei.c neueDatei.c > dateiDiff
martin@bart:~$ cat dateiDiff
--- datei.c     2005-05-09 16:37:50.152260552 +0200
+++ neueDatei.c 2005-05-09 16:38:16.043324512 +0200
@@ -1,13 +1,12 @@
 #include <stdio.h>

 int main() {
-  int zahl = 42;
   char wort[255];

-  printf("\nWort eingeben: ");
+  printf("Wort eingeben: ");

   scanf("%s", wort);
-  printf("Wort ist: %s\n", wort);
+  printf("Wort war: %s\n", wort);
```

Ein Fallbeispiel(4)

Den Patch einspielen.

```
martin@bart: $ patch -p0 < dateiDiff
patching file datei.c
```

- Der Patch in dateiDiff wird eingespielt
- Das Programm patch entnimmt der Datei dateiDiff, welche Datei geändert werden soll
- Die Option -p0 gibt an, dass patch nur im aktuellen Verzeichnis suchen soll

make
Funktion

- Bestimmt Reihenfolge von (Neu-)Übersetzungen
- Grundlagen
  - Zeitstempel
  - Abhängigkeiten:

```
foo : bar.o baz.o quux.o
    cc -o foo bar.o baz.o quux.o
bar.o : bar.c defs.h
    cc -c -g -pg bar.c
⋮
clean :
    rm -f foo bar.o baz.o quux.o
.PHONY : clean
```

make

Variablen, Wildcards

- ```
  foo : bar.o baz.o quux.o
      cc -o foo bar.o baz.o quux.o
  ```
  - Schlecht! Umständlich, anfällig für Fehler.
- Besser:
  ```
  OBJS = bar.o baz.o quux.o
  foo : $(OBJS)
      cc -o foo $(OBJS)
  ```
- Wildcard-Ersetzung:
  ```
  OBJS = *.o
  foo : $(OBJS)
      cc -o foo $(OBJS)
  ```
  ```

make

Bedingungen, Funktionen, Muster

- Typisches Beispiel aus `debian/rules`:

```
ifneq (,$(findstring noopt,$(DEB_BUILD_OPTIONS)))
    OPT = -O0
else
    OPT = -O2
endif
```

- Patterns:

```
OBJS = bar.o baz.o quux.o
foo : $(OBJS)
    $(LD) -o foo $(OBJS)
%.o: %.c
    $(CC) -c $(CFLAGS) $(CPPFLAGS) $< -o $@
```

autoconf

- Warum?
  - Unix != Unix
  - Konfiguration soll automatisch ermittelt werden.
- Funktion
  - `configure.ac` → `configure`
  - Aufruf von `./configure` führt Tests durch und erzeugt schließlich `Makefile` aus `Makefile.in`, `config.h` aus `config.h.in`
- *autoscan* erzeugt Grundgerüst für `configure.ac`

automake

- Warum?
  - make != make
  - Abhängigkeiten über mehrere Verzeichnisse nicht trivial
- Funktion
  - `Makefile.am` → `Makefile.in`
  - Einfaches Beispiel für `Makefile.am`

```
bin_PROGRAMS = foo
foo_SOURCES = main.c foo.c bar.c baz.c
```

doxygen

- Generierung von Quellcode-Dokumentation aus Kommentaren
- Visualisierung von Code-Strukturen
- Unterstützung für C++, C, Java, Objective-C, IDL, PHP
- Kommentare:
  - C:
    `/**...*/`
  - C99/C++:
    `///...`
  - Qt:
    `/*!...*/`