„Error-handling in C“

Proseminar C-Programmierung

24.6.2011

- errno
- Exceptions
- GError
- Zusammenfassung

Wofür brauchen wir „Error-handling“?

```
int main()
{
	int x;

	scanf("%d", &x);

	printf("x*x=%d", x*x);

	return 0;
}
```

Was passiert bei der Eingabe „3.14“ oder „xyz“?

errno.h

- Thread-lokale Variable „errno“
- Makros
  - EDOM „Domain error“
  - EILSEQ „Illegal sequence“
  - ERANGE „Range Error“
- Textdarstellung durch „strerror(errno)“
- Vorsicht: strerror() ist nicht threadsicher
- Die sichere Methode strerror_r() ist nicht im C99-Standard

Benutzung

```
#include <errno.h>

double kehrwert(int x)
{
	double ergebnis = 0;
	if(x != 0)
	{
		ergebnis = 1.0/x;
	}
	else
	{
		errno = EDOM;
		ergebnis = 0;
	}
	return ergebnis;
}
```

Demo

Vorteile von errno

- Viele Bibliotheksfunktionen nutzen errno
- Einfache Benutzung
- Textdarstellung einfach zugänglich

Nachteile von errno

- Globale Variable
- Manuelle Überprüfung nötig
- Unbemerkte Änderungen möglich
- Keine Angaben über Ursprung des Fehlers
- Fehleranfällig bei Multithreading

Exceptions

- Sprachkonstrukt zum Behandeln von Ausnahmen
- Häufig werden Schlüsselwörter „try“, „catch“, „throw“ und „finally“ verwendet (z.B. in Java)
- Bei Fehlern werden Exceptions „geworfen“
- Aktuelle Methode wird abgebrochen
- Aufrufende Methode kann Fehler behandeln oder weiterreichen

Beispiel

- Exceptions in Java

```
try
{
    x = kehrwert(15);
}
catch(Div0Exception e)
{
    System.out.println("Fehler: Division durch 0");
}
finally
{
    System.out.println("Done.");
}
```

Regeln

- Methoden geben bekannt, welche Exceptions sie werfen können
- Aufrufende Methode darf Exceptions nicht ignorieren
- Darf aber die Exception „weiterreichen“

```
double funktion(int x) throws Div0Exception
{
    return kehrwert(x);
}
```

Exceptions in C

- Exceptions sind nicht Teil des C99-Standards
- Viele Implementationen bekannt
- Leider keine einheitliche Lösung

Beispiel: cexcept.h [5]

```
define_exception_type(int);
struct exception_context the_exception_context[1];

void werfer(int x)
{
        Throw x;
}

void faenger()
{
        int e;
        Try
        {
                werfer(0);
        }
        Catch(e) printf("Fehler %d", e);
}
```

Vorteile von Exceptions

- Syntaktische Trennung von der Programmlogik
- Platzsparend
- Lokale Fehlerbehandlung möglich
- Ursprung des Fehlers oft nachvollziehbar

GError

- Die Struktur GError [3]

```
struct GError {
    GQuark       domain;
    gint         code;
    gchar        *message;
};
```

Wichtige Funktionen

- GError* g_error_new(GQuark domain, gint code,
  const gchar *format, ...)
- void g_set_error(GError **err, GQuark domain,
  gint code, const gchar *format, ...)
- void g_error_free(GError *error)
- void g_propagate_error(GError **dest, GError *src)

Benutzung (1)

```
#include <glib.h>

int main()
{
	GError *error = NULL;
	double x = kehrwert(3, &error);

	if(error != NULL)
	{
		printf("%s", error->message);
		g_error_free(error);
	}
	else
	{
		printf("Der Kehrwert von 3 ist %f", x);
	}
}
```

Benutzung (2)

- Auch zulässig:

```
double x = kehrwert(3, NULL);
```

- Methode sollte Fehler über den Rückgabewert melden

Demo

Vergleich zu Exceptions

- Lokale Fehlerbehandlung möglich
- Detaillierte Informationen verfügbar
- Methoden haben trotzdem einen Rückgabewert
- Funktionen benötigen zusätzlichen Parameter
- Kein syntaktischer Unterschied zur Programmlogik

Zusammenfassung

- errno.h liefert eine einfache Möglichkeit, Fehler zu melden
- Exceptions sind zuverlässiger, aber kein Bestandteil von C
- GError bietet Fehlerbehandlung in C an, ist aber syntaktisch nicht vom Anwendungscode getrennt

Quellen

[1] http://openbook.galileocomputing.de/c_von_a_bis_z/030_c_anhang_b_004.htm (13.6.2011)

[2] http://de.wikipedia.org/wiki/Ausnahmebehandlung (13.6.2011)

[3] http://developer.gnome.org/glib/unstable/glib-Error-Reporting.html (14.6.2011)

[4] http://pubs.opengroup.org/onlinepubs/009695399/functions/errno.html (11.6.2011)

[5] http://www.nicemice.net/cexcept/ (17.6.2011)