Fakultät für Informatik

MNM TEAM
Munich Network Management Team

Institut für Informatik

LMU

Systempraktikum im Wintersemester 2009/2010 (LMU):
Vorlesung vom 29.10. – Foliensatz 2

Modularisierung (T)
Eigene Headerdateien und Bibliotheken (P)
Arten der Parameterübergabe (P)
Arrays und Strukturen (P)
Rekursive und iterative Funktionen (P)

Dr. Thomas Schaaf, Dr. Nils gentschen Felde

Modularisierung

MNM TEAM

- Deklaration: führt einen oder mehrere Bezeichner ein oder wiederholt sie
- Definition: Eine Definition ist eine Deklaration, die mindestens eine dieser Bedingungen erfüllt:
  - Reservieren von Speicherplatz für das deklarierte Objekt wird veranlasst
  - Im Falle einer Funktion wird ihr Rumpf angegeben
- Beispiele: Deklaration oder Definition?

```
int a = 3;
int b;
extern int c;
static int x;
int f(void);
```

Modularisierung

MNM TEAM

- Grundbegriffe
  - Bindung
    - Keine Bindung: Block-lokale Variablen
    - Interne Bindung: Variablen lokal zu einer Übersetzungseinheit
    - Externe Bindung: Objekte, die in verschiedenen Modulen benutzt werden können
  - Speicherdauer
    - Statisch: von Beginn bis Ende des Programmlaufs
    - Dynamisch: Anlegen und Zerstören mit speziellen Befehlen
    - Automatisch: vom Eintritt in bis Austritt aus dem Sichtbarkeitsbereich
  - Sichtbarkeit
    - Lokal: Bezeichner ist nur in einem bestimmten Bereich gültig
    - Global: Bezeichner ist überall gültig

Modularisierung

MNM TEAM

- Zusammenhänge zwischen Bindung, Speicherdauer und Sichtbarkeit
  - Interne Bindung → Lokale Sichtbarkeit
  - Externe Bindung → Globale Sichtbarkeit
  - Interne oder externe Bindung → Statische Speicherdauer

Modularisierung

MNM TEAM

- Modularisierung
  - Aufteilung des Gesamtsystems (Programms) in mehrere Teilsysteme (Teilprogramme/Module)
  - Teilprogramme sollen weitgehend unabhängig voneinander sein
  - Jedes Teilprogramm/Modul erhält eine Schnittstelle
    - Schnittstelle beschreibt, was das Modul macht, aber nicht, wie
    - Schnittstellen vereinfachen das Zusammensetzen von Modulen und die Kommunikation zwischen Modulen
  - Module können zu Bibliotheken zusammengesetzt (gebunden/gelinkt) werden:
    - Statisches Binden/Linken
    - Dynamisches Binden/Linken

- Beispiel: Interne und externe Bindung
  - Drei Programm-Module:
    - modul1.c:

```
void test(void) {
   printf("Test1\n");
}
```

    - modul2.c:

```
void test(void) {
   printf("Test2\n");
}
```

    - main.c:

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

extern void test(void);

int main(void) {
   test();
   return EXIT_SUCCESS;
}
```

`gcc -o test main.c modul1.c modul2.c`

verursacht einen Fehler:

```
/tmp/ccMZ6iMI.o(.text+0x0): In
function `test':
: multiple definition of `test'
```

- Verbesserung
  - Funktion `test()` in modul2.c intern binden (mit `static`):
    - modul1.c:

```
void test(void) {
   printf("Test1\n");
}
```

    - modul2.c:

```
static void test(void) {
   printf("Test2\n");
}
```

    - main.c:

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

extern void test(void);

int main(void) {
   test();
   return EXIT_SUCCESS;
}
```

Ausführung und Ausgabe:

Test1

Modulare Systemprogrammierung

MNM TEAM

- Wiederholung: Headerdateien
  - Was ist eine Headerdatei?
    - Beschreibung der Schnittstelle zu einem Modul
    - Modul kann Teil einer Bibliothek sein
  - Was kann eine Headerdatei enthalten?
    - Variablen- und Funktionsdeklarationen
    - Makro-Definitionen
    - Konstanten-Definitionen
    - Typdefinitionen
    - Aufzählungen
    - Namensdeklarationen
    - `#include`-Anweisungen
    - Direktiven für bedingte Kompilierung
    - Kommentare (Dokumentation der Schnittstelle)

- Beispiel: Eigene Headerdateien
  - -geometrie.h:

```
#ifndef geometrie
#define geometrie

#define PI (3.1415926)

float quadrat(float x);
float kreisflaeche (float radius);
float kugelvolumen (float radius);

#endif
```

  - -geometrie.c:

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

#include "geometrie.h"

float quadrat(float x) {
    return x * x;
}

float kreisflaeche(float radius) {
    return PI * quadrat(radius);
}

float kugelvolumen(float radius) {
    return (4/3) * PI *
    quadrat(radius) * radius;
}
```

- Beispiel: Eigene Headerdateien (Forts.)
  - main.c:

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

#include "geometrie.h"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Aufruf: start <radius>\n\n");
        return EXIT_FAILURE;
    }

    float radius = atof(argv[1]);
    printf("Kreisfläche = %f.\n", kreisflaeche(radius));
    printf("Kugelvolumen = %f.\n", kugelvolumen(radius));
    return EXIT_SUCCESS;
}
```

- Beispiel: Bibliotheken (libraries)
  - geometrie.h, geometrie,c, main.c unverändert
  - Zusätzlich algebra.h:

```
#ifndef algebra
#define algebra

#define E (2.7182818)

float mittelwert(float a, float b);

#endif
```

  - algebra.c:

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

#include "algebra.h"

float mittelwert(float a, float b) {
    return (a+b) / 2;
}
```

  - Idee: Bibliothek für Geometrie und Algebra → mymath.a
    - Schritt 1: Objektdateien erstellen

```
gcc -c algebra.c geometrie.c
```

    - Schritt 2: Objektdateien zu Bibliothek linken

```
ar mymath.a algebra.o geometrie.o
```

    - Schritt 3: Übersetzen unter Verwendung der Bibliothek

```
gcc -o start main.c mymath.a
```

Arten der Parameterübergabe

MNM TEAM

- Arten der Parameterübergabe bei Funktionsaufrufen
  - Wertübergabe (call by value)
  - Adressübergabe (call by reference)
- Beispiel 1: Call by reference (falsch!)
  - cbrfalsch.c:

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

int plus1(int *x) {
    *x = *x + 1;
    return *x;
}

int main(void) {
   int a, *x = malloc(sizeof(int));
   *x = 5;
   printf("Wert von x vorher: %i\n", *x);
   a = plus1(*x);
   printf("Wert von x nachher: %i\n", *x);
   free(x);
   return 0;
}
```

```
gcc -o test cbrfalsch.c
```

verursacht folgende Warnung:

```
cbrfalsch.c:13: warning: passing
arg 1 of `plus1' makes pointer
from integer without a cast
```

```
./test
```

liefert:

```
Wert von x vorher: 5
Segmentation fault
```

Arten der Parameterübergabe

MNM TEAM

- Beispiel 2: Call by reference
  - main.c:

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

extern int plus1(int *x);
extern int minus1(int *x);

int main(void) {
   int *x = malloc(sizeof(int));
   *x = 5;

   printf("x is %i\n", *x);
   printf("%i + 1 = %i\n", *x, plus1(x));
   printf("%i - 1 = %i\n", *x, minus1(x));

   free(x);
   return 0;
}
```

  - plusminus.c:

```
int plus1(int *x) {
    *x = *x + 1;
    return *x;
}

int minus1(int *x) {
    *x = *x - 1;
    return *x;
}
```

Ausführung und Ausgabe:
x is 5

  - Problem hier: Verändern von Speicherinhalten durch aufgerufene Funktionen kann unerwünscht sein (Vgl. Ausgabe)

    → Deklaration mit `const`

Arten der Parameterübergabe

MNM TEAM

- Beispiel 3: const-deklarierte Parameter
  - main.c:

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

extern int plus1(int *x);
extern int minus1(int *x);

int main(void) {
   int *x = malloc(sizeof(int));
   *x = 5;

   printf("x is %i\n", *x);
   printf("%i + 1 = %i\n", *x, plus1(x));
   printf("%i - 1 = %i\n", *x, minus1(x));

   free(x);
   return 0;
}
```

  - plusminus.c:

```
int plus1(const int *x) {
    *x = *x + 1;
    return *x;
}

int minus1(const int *x) {
    *x = *x - 1;
    return *x;
}
```

```
gcc -o test main.c plusminus.c
```

verursacht folgende Warnungen:

```
plusminus.c: In function `plus1':
plusminus.c:2: warning: assignment of read-only location
plusminus.c: In function `minus1':
plusminus.c:7: warning: assignment of read-only location
```

Arten der Parameterübergabe

MNM TEAM

- Beispiel 4: `const`-deklarierte Parameter
  - main.c:

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

extern int plus1(int *x);
extern int minus1(int *x);

int main(void) {
   int *x = malloc(sizeof(int));
   *x = 5;

   printf("x is %i\n", *x);
   printf("%i + 1 = %i\n", *x, plus1(x));
   printf("%i - 1 = %i\n", *x, minus1(x));

   free(x);
   return 0;
}
```

  - plusminus.c:

```
int plus1(const int *x) {
    return *x + 1;
}

int minus1(const int *x) {
    return *x -1;
}
```

Ausführung und Ausgabe:
x is 5

Arrays

MNM TEAM

- Arrays (Felder) in C
  - Eigenschaften:
    - Feste Länge
    - Eindeutiger Datentyp → Jedes Element im Array hat den gleichen Datentyp
    - Definition eines Arrays mittels `[]`-Operator liefert die Adresse des ersten Elements des Arrays (Pointer auf das Array)
  - Arten von Arrays:
    - Eindimensionale Arrays – z.B.:

```
int a[4];
```

    - Mehrdimensionale Arrays – z.B.:

```
char s[3][10];
```

  - Zusammenhang: Arrays und Adressen:
    - Es gilt:

```
&a[0] == a
&s[0][0] == s
```

Arrays

MNM TEAM

- Beispiel: Definition und Zugriff auf ein Array
  - array.c:

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

char s[3][5] = {
   {'a', 'b', 'c', 'd', 'e'},
   {'f', 'g', 'h'},
   {'i', 'j', '\0', 'k', 'l'}
};

int a[] = {1, 2, 3, 4, 5};

int main(int argc, char *argv[]) {
   printf("%c \n", s[0][0]);
   printf("%p \n", s[0]);
   printf("%p \n", &s[0]);
   printf("%p \n", &s[0][1]);
   printf("%s \n", s[0]);
   printf("%s \n", s[1]);
   printf("%s \n", s[2]);
```

Ausführung und Ausgabe:

Arrays

- Beispiel: Definition und Zugriff auf ein Array (Forts.)
  - array.c:

```
char *pointer = &s[0][0];
   printf("%c \n", *pointer);
   pointer +=10;
   printf("%c \n", *pointer);

   printf("%i \n", a[3]);
   printf("%i \n", *(a+3));
   printf("%i \n", *(3+a));
   printf("%i \n", 3[a]);
   printf("%p \n", &a[3]);
   printf("%p \n", a+3);
   printf("%p \n", &a[4]);

   return EXIT_SUCCESS;
}
```

Ausführung und Ausgabe:

Strukturen

MNM TEAM

- Strukturen in C
  - Eigenschaften:
    - Feste Länge (Vgl. Array)
    - Kein eindeutiger Datentyp → Elemente einer Struktur können unterschiedlichen Typs sein
    - Strukturen können selbst als eine "Art Datentyp" aufgefasst werden
    - Häufig: Deklaration von Strukturen in Headerdateien

- Beispiel: Definition und Zugriff auf eine Struktur
  - structure.c:

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

struct myStructure {
   int value1;
   float value2;
   char text[10];
};

int main(int argc, char *argv[]) {
   struct myStructure s1;

   s1.value1 = 3;
   s1.value2 = 0.77;
   strcpy(s1.text, "abc");

   printf("%i, %f, %s\n", s1.value1, s1.value2, s1.text);

   return EXIT_SUCCESS;
}
```

Iterative und rekursive Programme

MNM TEAM

- Beispiel: Fibonacci-Zahlen
  - Fibonacci-Folge: $f_n = f_{n-1} + f_{n-2}$ für $n > 1$
    mit den Anfangswerten $f_0 = 0$, $f_1 = 1$
  - Fibonacci-Zahlen: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...
- Rekursive Implementierung einer Funktion zur Berechnung der x-ten Fibonacci-Zahl

```
long fib_rec(long x) {
  if (x <= 1)
    return(x);
  else
    return (fib_rec(x-1) + fib_rec(x-2));
}
```

Iterative und rekursive Programme

MNM TEAM

- Iterative Implementierung einer Funktion zur Berechnung der x-ten Fibonacci-Zahl

```
long fib_it(long x) {
  long f0, f1, tmp, i;
  f0 = 0;
  f1 = 1;

  if (x <= 1)
    return( x);

  for(i=2; i<=x; ++i) {
    tmp = f1;
    f1  = f0 + f1;
    f0  = tmp;
  }
  return(f1);
}
```

Zusammenfassung

MNM TEAM

- Grundlagen modularer Systemprogrammierung
- Schnittstellen, Module, Bibliotheken
- Deklaration und Definition
- Bindung, Speicherdauer, Sichtbarkeit
- Statisches und dynamisches Binden

- Bindung, Speicherdauer und Sichtbarkeit in C
- Eigene Headerdateien und Bibliotheken
- Parameterübergabe: Call by value und Call by reference
- Arrays und Strukturen
- Iterative und rekursive Programme/Funktionen