Grundkurs C

Nils Schmeißer
Abteilung Kommunikation und Datenverarbeitung
Forschungszentrum Rossendorf e.V.

Inhalt

- Einführung
- Programmstruktur
- Typkonzept und Ausdrücke
- Kontrollstrukturen
- Blockkonzept und Funktionen
- Modulkonzept und Speicherklassen
- Besonderes
- Richtlinien zur Programmentwicklung

Einführung

- “general-purpose” Programmiersprache
- typisierte Sprache
- “low-level” Sprache
  - keine Operationen zum direkten Umgang mit nicht-generischen Datentypen
  - keine Ein-/Ausgabeanweisungen
  - Nutzung von Bibliotheken
- C ist “klein”

Einführung

- von B (Thompson, 1970) abgeleitet
- 1978 Kernighan und Ritchie
  “The C Programming Language”
- 1988 ANSI Standard
- 1991 Stroustrup
  “The C++ Programming Language”

Compiler

- auf dem IBM/PC
  - Borland
  - Microsoft
  - GNU (freier Compiler)
- auf Workstation, Großrechner
  - alle Hersteller
  - GNU (freier Compiler)

Literatur

- “The C Programming Language”; B. Kernighan, D. Ritchie
- “C/C++” Dokumentationen” von
  - Borland, IBM, HP, ...
- “Programmieren in C”; Clauss/Fischer
- “The C++ Programming Language”; B. Stroustrup

Programmstruktur

- ein C-Programm besteht aus einer Menge von Funktionen, die (in sich geschachtelte) Anweisungen enthalten. Beispiel:

```
#include <stdio.h>
int main() {
     printf("Hello world.\n");
     return 0;
}
```

Datei: hello.c

Übersetzung 1

- ein C-Programm wird durch den Aufruf des Compilers übersetzt:

  cc hello.c

- unter UNIX produziert der Compiler ein Executable mit dem Namen “a.out”

Übersetzung 2

- zur Sicherung der ANSI Konformität erhält der Compiler meist eine Option

Borland C	PC	bcc -A
GNU C	alle	gcc -ansi
Convex	C-Serie	cc -std
HP, Convex	WS, SPP	cc -Aa
IBM	RS/6000	xlc

Beispiel

- aus folgendem “Rahmen” kann ein einfaches Programm entstehen

```
#include <stdio.h>
int main() {
  int a,b,c;            /* Deklaration */
  printf(”a b:”);           /* Ausgabe */
  scanf(”%d %d”,&a,&b);     /* Eingabe */
  c=a+b;                  /* Zuweisung */
  printf(”%d + %d = %d\n”,c);
  return 0;
}
```

Datei: ex1.c

Sprachbeschreibung 1

- für die nächsten Folien soll folgende Notation gelten:
  - Schlüsselwörter werden gelb dargestellt
  - um einen Bereich hervorzuheben, werden sie auch grün dargestellt
  - *Metavariable*, die beliebige andere syntaktische Strukturen verkörpern, werden kursiv gesetzt
  - in *[ ]* geklammerte Spracheinheiten sind optional zu verwenden
  - `C Code wird in Courier angegeben`

Sprachbeschreibung 2

- Beispiel: Deklaration einer Konstanten

const *[Modifizierer] Typ*
Konstantenname = *Wert* ;

```
const int i = 5 ;
const unsigned char = 'a' ;
```

Typkonzept

- alle Informationen werden im Rechner als Binärzahlen gespeichert
- zur Unterscheidung der Semantik werden Datenbereiche (Typen) eingeführt
- Daten einzelner Bereiche können ineinander überführt werden

Typkonzept - C Typen

- C verfügt über einige interne Typen:
  - Zahlen
    - ganze Zahlen
    - rationale Zahlen
  - Zeichen/Zeichenketten
  - strukturierte Datentypen
    - Felder und Strukturen
  - Aufzählungstyp
  - Zeigertyp

Typkonzept - ganzzahlige Typen

- ganzzahlige Typen

short int	2 Byte	-32’768..32’767
int	4 Byte	-2'147'483'648 ..2'147'483’647

Datei: ex2.c

Typkonzept - ganzzahlige Typen

- Integer Konstanten:
  - 1234567890,-12345 (dezimal)
    dürfen nicht mit 0 beginnen
  - 01234567 (octal)
    beginnen immer mit 0
  - 0x123456789abcdef (hexadzimal)
    beginnen immer mit 0x

Typkonzept - Gleitkommazahlen

- rationale Zahlen

float	4 Byte	$3.4*10^{-38}$ .. $3.4*10^{38}$
double	8 Byte	$1.7*10^{-308}$ .. $1.7*10^{308}$

Typkonzept - Gleitkommazahlen

- Gleitkommakonstanten:
  - 23.45e5 23.45×10^{6}
  - .0 0
  - 0. 0
  - 1. 1.0×10^{0}
  - -1.23 -1.23
  - 2e-5 2.0×10^{-5}
  - 3E+10 3.0×10^{10}
  - .09E34 0.09×10^{34}

Typkonzept - Variablen

- Variablen müssen in C mit ihrem Typ erklärt werden (Deklaration)
  - Beispiel:

```
int i , I , _j , _k1_2l3 ;
type ident_list ;
```

  - durch den Vorsatz “const” wird eine Variable als Konstante erklärt; in diesem Fall muß der Wert der Konstanten mit angegeben werden Beispiel:

```
const float PI=3.1415926 ;
```


Typkonzept - Modifizierer

- Modifizierer
  - unsigned vorzeichenlose Zahl
    - unsigned short int 0 .. 2^{16}-1
    - unsigned int 0 .. 2^{32}-1
  - long doppelte Genauigkeit
    - unsigned long int 0 .. 2^{64}-1
    - long double
    - ! die Implementationen können unterschiedlich sein
      - » 10 Byte (Borland C/C++)
      - » 12 Byte (GNU C/C++, Linux)
      - » 16 Byte (HP/Convex S-Class)

Typkonzept - Zeichentyp

- Zeichen werden durch ihren ASCII (American Standard Code for Information Interchange) dargestellt; dieser umfaßt die Werte 0..127

char	1 Byte	-128 .. 127
unsigned char	1 Byte	0 .. 255

Typkonzept - Zeichenketten

- Zeichenketten sind Folgen von Zeichen, die durch das Zeichen mit dem ASCII 0 abgeschlossen werden (sog. 0-terminierte Strings)
- Zeichenketten gibt es nicht als Datentyp
- Zeichenketten werden in " " eingeschlossen

```
"ABC"
```

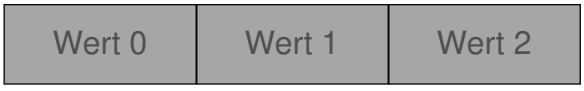
Typkonzept

C verfügt über eine Reihe interner Typen:

- Zahlen
  - ganze Zahlen
  - rationale Zahlen
- Zeichen/Zeichenketten
- strukturierte Datentypen
  - Felder und Strukturen
- Aufzählungstyp
- Zeigertyp

Typkonzept - Felder 1

- mehrere Daten gleichen Typs werden zu einem Feld zusammengefaßt
- Feldelemente werden mit 0 beginnend indiziert



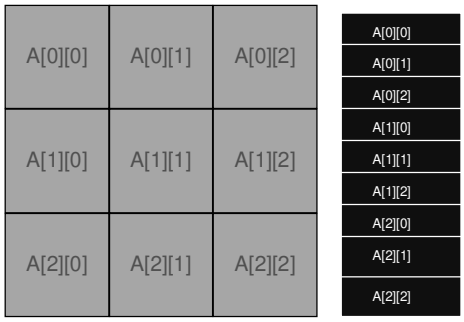
Feld

Typkonzept - Felder 2

- eine Feldvereinbarung kann in der Variablendeklaration stattfinden

Beispiel:

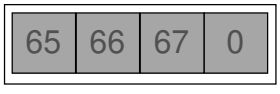
```
int index[100];
double A[3][3];
```



Typkonzept - Zeichenketten

- nochmal Zeichenketten:
  da Zeichenketten eine Folge von Elementen des Typs “char” darstellen, können sie auch als Feld aufgefasst werden:

```
const char text[4]="ABC";
```



(text)

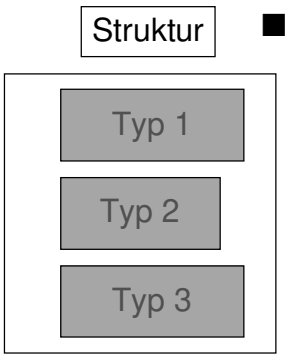
- die Qualifizierung erfolgt über den Index, Beispiel: `text[2], x[index[k]]`

Typkonzept - Feldkonstanten

```
#include < stdio.h>
int main() {
  const float preis[]={0.35,0.79,1.79};
  int ware,anzahl;
  printf("Broetchen (0)\n");
  printf("    Milch (1)\n");
  printf("    Wurst (2): ");
  scanf("%d",&ware);
  printf("       Anzahl: ");
  scanf("%d",&anzahl);
  printf("Preis=%f\n",
         (float)anzahl*preis[ware]);
  return 0;
}
```

Datei: ex3.c

Typkonzept - Strukturen 1



- Objekte (Dinge) werden durch verschiedenste Eigenschaften charakterisiert; diese Eigenschaften lassen sich zu einem Typ zusammenfassen

Typkonzept - Strukturen 2

- die Elemente einer Struktur heißen Komponenten; Komponenten werden über einen '.' qualifiziert

```
struct {
    float x,y;
    float radius;
    int color;
} a_circle ;
```

```
a_circle.color
a_circle.radius=3.45;
a_circle.x=a_circle.x+10;
```

Datei: ex4.c

Typkonzept - Strukturen 3

- `struct circle { ... };` definiert einen neuen Datentyp "circle"

```
struct point { float x,y; };
struct circle {
  struct point center;
  float radius;
};
struct circle a_circle;
a_circle.center.x=1.0;
a_circle.radius=5.2;
```

Typkonzept - union

- im Gegensatz zu struct stellen unions einen Container für verschiedenartige Informationen dar

```
union data {          a_data.idata[0]
   int idata[2];      a_date.idata[1]
   double ddata;
} a_date ;            a_date.ddata
```


Typkonzept - Aufzählungstyp

- C bietet die Möglichkeit eine geordnete Menge von Symbolen zu definieren

```
enum { Symbolliste } Variablenname ;
```

Beispiel:

```
enum { EIN,AUS } zustand;
...
zustand=AUS;
```

Typkonzept - Beispiel Strukturen

```
struct Circle
struct Rectangle
union GeometryElement {
      struct Circle circle;
      struct Rectangle rectangle;
};
struct Geometry {
      enum { CIRCLE,RECTANGLE } type;
      union GeometryElement element;
};
struct Geometry geometry
geometry.type=CIRCLE;
geometry.element.circle.center.x=-1;
```


Typkonzept - Typdefinition 1

- selbstdefinierter Datentyp:

```
typedef Typbeschreibung  Typname ;
```

Beispiel:

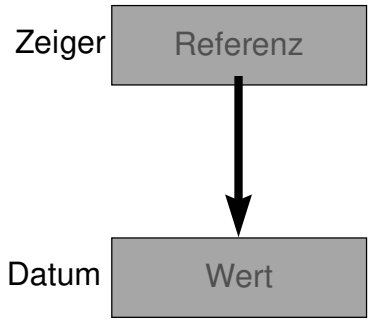
```
typedef unsigned short int word;
typedef struct { float x,y; } point;
typedef enum { EIN,AUS } zustand ;
```

Typkonzept - Typdefinition 2

- Beispiel “Personaldaten”

```
typedef char String[256];
typedef struct {
  unsigned short day,month,year;
} Date;
typedef enum {VERWALT,RZ,INST1,INST2} Departement;
typedef struct {
  String Name,Vorname;
  Departement departement;
  Date birthday;
} Personaldata;
...
Personaldata Mitarbeiter[2000];
```


Typkonzept - Zeiger 1

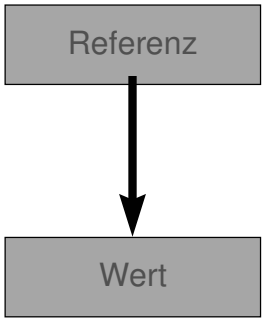


- Zeiger (Referenzen) verweisen auf die Stelle, an der sich ein Datum befindet

Beispiel:

```
int *zeiger_auf_int;
```

Typkonzept - Zeiger 2

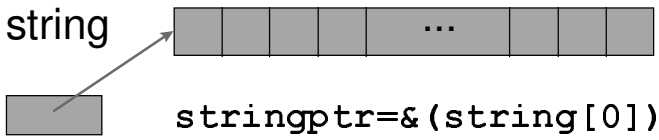



```
int *zx;

x = *zx;  (Dereferenzierung)
zx = &x;  (Referenzierung)

int x;
```


Typkonzept - Zeiger und Felder

- da die Elemente eines Feldes zusammenhängend abgelegt werden, kann auch eine Referenz auf das erste Element das Feld repräsentieren

```
char string[256]; char *stringptr;
```




```
stringptr=&(string[0]);
```

Typkonzept - Zeiger und Strukturen

- auch Strukturen können referenziert werden
- Beispiel:

```
a_circle   == *(&a_circle)
a_circle.x ==  (&a_circle)->x
struct circle *cptr;
cptr=&a_circle; ... cptr->x=...;
cptr=&b_circle; ... cptr->y=...;
```


Typkonzept - Zusammenfassung

- Zusammenfassung (einfache Typen)
  - Zahlentypen
    - short int, int, long int, unsigned short int, unsigned int, unsigned long int
    - float, double, long double
    - Aufzählungstyp
  - Zeichen
    - char, unsigned char
  - Zeiger
    - type '*'

Typkonzept - Zusammenfassung

- Zusammenfassung (strukturierte Typen)

  Felder

  - type array_name ‘[‘ dim_0 ‘]’ .. ‘[‘ dim_n ‘]’
    Beispiel:

```
double Matrix[10][3]
```

  Strukturen

  - struct struct_name ‘{‘
    $comp_0$
    ...
    $comp_n$
    ‘}’ a_structure;

```
struct Matrix {
    int n,m;
    double e[10][10];
} a_matrix;
```


Typkonversion 1

```
#include <stdio.h>
int main() {
  int i;
  float x;

  x=3.0;
  i=x/4.0;
  printf("> %d\n",i);
  return 0;
}
```

Ausgabe:

```
> 0
```

Datei: conv1.c

Typkonversion 2

- die Zuweisung von Werten an Variablen erfolgt mit dem Zuweisungsoperator '=' Beispiel: `x=1.0;`
- eine Zuweisung kann nur an Variablen gleichen Typs erfolgen Beispiel:

```
float x;
x=1; /* "int 1" */
```

hier muß die ganze Zahl 1 in die FP Zahl 1.0 konvertiert werden

Typkonversion 3

- die Konversion kann implizit erfolgen

```
int x;  x=1.75;
```

oder explizit

```
int x;  x=(int)1.75;
```

zulässig sind:

- short ↔ int ↔ long ↔ float ↔ double
  - ← unter Verlust von Informationen
- char ↔ int
  - ← nur wenn 0<int<255

Ausdrücke

- durch die Zusammensetzung von Werten und Variablen eines Typs mittels Operatoren entstehen Ausdrücke
  Beispiel:
  - » x+y
  - » i+10+a_circle.color
  - » a[0]*b[0]+a[1]*b[1]
  - » (a+b)*c

Ausdrücke - Operatoren 1

- C kennt nur Operatoren für interne Datentypen
- es gibt verschiedene Klassifikationen für Operatoren
- die Operatoren haben verschiedene Wertigkeiten
- die Evaluierungsreihenfolge eines Ausdrucks ist nicht spezifiziert

Ausdrücke - Operatoren 2

- Nach Anzahl der Argumente
  - unäre (präfix: op arg, postfix: arg op)
  - binäre (infix: arg op arg)
- nach Basistyp
  - Bitoperatoren
  - arithmetische Operatoren (für Zahlen)
  - logische Operatoren

Ausdrücke - Bit-Operatoren

- Bitoperationen:
  - & bitweises UND (AND)
  - | bitweises ODER (OR)
  - ^ bitweises EXKLUSIVES ODER (XOR)
  - << bitweise Linksverschiebung, von rechts wird mit 0 gefüllt
    >> bitweise Rechtsverschiebung, von links wird mit 0 gefüllt
  - ~ bitweise Negation

Datei: expr1.c

Ausdrücke - arithmetische Op.

- arithmetische Operatoren
  - unäre Operatoren
    - » Präfix "`-`" (Vorzeichen)
    - » Präfix "`++`" und "`--`" (Inkrementieren und Dekrementieren um 1, nur ganzzahlige Typen) der Wert von "`++i`" ist `i+1`, i erhält den Wert i+1
    - » Postfix "`++`" und "`--`" der Wert von "`i++`" ist i, i erhält den Wert i+1
  - `+`, `-`, `*`, `/` die klassischen arith. Operatoren
  - `%` Modulo

Datei: expr2.c

Ausdrücke - Zuweisungen

- Zuweisungsoperatoren
  - `+=`, `-=`, `*=`, `/=`, `%=`, `bitop=`
    x op= y ist identisch zu x=(x) op (y); Ausdruck x wird nur einmal berechnet, Beispiel: `x[i+1]+=4;`
- "Bedingungsoperator"
  - `?:`

```
X = C ? A1 : A2
```

X erhält den Wert von A1, falls C einen Wert ungleich 0 hat, den Wert von A2 sonst

Ausdrücke - logische Operatoren

- einen Bool'schen Datentyp gibt es in C nicht, dafür werden alle Ausdrücke, deren Wert gleich 0 ist als "FALSCH", alle anderen als "WAHR" interpretiert
- logische Operatoren
  - Vergleichsoperatoren
    - » <, <=, ==, =>, >
  - logische Operatoren
    - » && logisches UND
    - » || logisches ODER
    - » ! logische Negation

Ausdrücke - Klammerung

- Klammerausdruck:

  ein beliebiger Ausdruck kann geklammert werden, die Evaluierung erfolgt dann nach den üblichen Regeln
- Beispiel:

  `(A+B)*C`

  zuerst wird der Wert von "**`A+B`**" bestimmt, dann der von "**`...*C`**"

Ausdrücke - “,” Operator

- mehrere Ausdrücke können mit einem “ , ” zu einem “,”-Ausdruck zusammengefaßt werden; die Abarbeitung erfolgt von links nach rechts, der Wert des Ausdrucks ist der Wert des am weitesten rechts stehenden Ausdrucks; Beispiel:

```
( A , B ) == B
```


Ausdrücke - Operatorrangfolge

```
()   []   ->   .
 ~   ++   --   -.
(type)   *.   &.   sizeof
 *    /    %
 +    -
<<   >>
 <   <=   >=   >
==   !=
 &
 ^
 |
&&
||
?:
 =   +=   -=   *=   ...
 ,
```