EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT
Arbeitsbereich Symbolisches Rechnen

C-Einführung

Praktikum Betriebssysteme (INF4122)
WS 17/18

20.02.2018, Jens Haußmann, M. Sc.

Übersicht

- Compiler und IDE
- Syntax
- Variablen, Datentypen und Operatoren
- Kontrollstrukturen
- Funktionen und Programmstruktur
- Zeiger und Arrays
- Strukturen
- Ein- und Ausgabe
- Makefiles

Compiler und IDE

- In simpelster Form wird ein normaler Editor (z.B. VIM) zum Erstellen der Datei und die GNU Compiler Collection (gcc) für das Kompilieren verwendet. Um Informationen über einen Kommandozeilenbefehl zu erhalten, können wir die Man-Pages nutzen.
  Beispiel: `man gcc`
  Ausgabe: `gcc [-c|-S|-E] [-std=standard] [-g] [-pg] [-Olevel] [-Wwarn...] [-Wpedantic] [-Idir...] [-Ldir...] [-Dmacro[=defn]...] [-Umacro] [-foption...] [-mmachine-option...] [-o outfile] [@file] infile...`
  Für uns relevante Parameter:
  - `O0`: Optimierungslevel des Compilers.
  - `g`: Erstellt Debbuging-Informationen
  - `Wall`: Zeigt Warnungen und Fehler des beim Kompilieren an
  - `std=c99`: Welcher C-Standard soll genutzt werden
  - `o` outputfile

Compiler und DIE
Hello-World

1. Öffnen Sie VIM:
   ```
   vi helloWorld.c
   ```
2. Drücken Sie „i“ für insert und tippen Sie das nachfolgende Programm ab:
   ```
   #include<stdio.h>
   int main() {
   printf("Hello World\n");
   return 0;
   }
   ```
3. Speichern und schließen Sie den Editor mit „`esc`“, „`:wq`“.
4. Führen Sie den folgenden Befehl aus:
   ```
   gcc -O0 -g -Wall -std=c99 -o helloWorld helloWorld.c
   ```
5. Starten Sie Ihr Programm:
   ```
   ./helloWorld
   ```

Syntax

- Die Syntax von C hat u.a. Java beeinflusst.
- Anweisungen werden mit „ `;` “ getrennt.
- Anweisungen können mit „ `,` “ zusammengefasst werden.
- Viele Syntax-Konstrukte von Java gibt es auch in C:

```
if(foo==bar){
printf(„foo is bar“);
}else{
printf(„bar is not foo“);
}
```

- Funktionsdefinitionen so wie Methoden bei Java:

```
int faculty(int n){...}
```

Syntax
Hello World

```
#include<stdio.h>
int main(int argc, char* argv[]){
printf(“Hello World! \n”);
return 0;
}
```

- Mit #include <stdio.h> wird das Modul stdio.h geladen welches u.a. printf zur Verfügung stellt.
- Jedes ausfuhrbare C-Programm enthält eine Main-Funktion. Sie gibt einen int Wert zurück, welcher den Exit-Code angibt.
- C kennt keine Klassen! Wir haben es in C mit Funktionen und nicht mit Methoden zu tun!
- Die Parameter der main-Funktion werden später erklärt! (Ähnlich String[] in Java)

Variablen, Datentypen und Operatoren

- Variablennamen bestehen aus Buchstaben und Ziffern, wobei das erste Zeichen ein Buchstabe sein muss. Zudem sind sie case-sensitive.
- In C gibt es nur wenige elementare Datentypen:
  - int: Ganzzahliger Wert, üblicherweise in der für die Maschine „natürlichen“ Größe, d.h. 32 Bit oder 64 Bit.
  - float: Ein einfach genauer Gleitpunktwert, 4 Byte.
  - double: Ein doppelt genauer Gleitpunktwert, 8 Byte.
  - char: Ein Zeichen aus dem Zeichensatz der Maschine, 1 Byte.
- Wichtig: In C gibt es kein Boolean und kein String. Alles ungleich 0 ist wahr und Strings entstehen durch Char-Arrays.
- Gültige Operatoren: `+ - * / % << >> & ^ |`

Variablen, Datentypen und Operatoren
String

- Strings werden durch Char-Arrays repräsentiert, wobei jedes Element einen Buchstaben enthält.
- Problem: Wie kann das Programm feststellen, wann der Text zu Ende ist? Hier hilft die binäre Null als „Endzeichen“. Die binäre Null hat nichts mit der Ziffer 0 (ASCII-Code 48) zu tun, sondern ist das Zeichen mit dem ASCII-Code 0. Um dieses Zeichen eingeben zu können, wird die Schreibweise ‚\0‘ verwendet.
- Beispiel:

```
char text[10];
text[0]=‚a‘;
text[1]=‚b‘;
text[2]=‚\0‘;
```

Variablen, Datentypen und Operatoren
String

- Bequemere Möglichkeit:
  char text[10]=„ab“;

- Speicherdarstellung:

Wert	a	b	\0
Adresse	100	101	102

- Die Funktion „printf“ wird verwendet, um eine Konsolenausgabe zu erzeugen:
  printf(„Erster Buchstabe: %c\n“, text[0]);

Variablen, Datentypen und Operatoren
String

- Platzhalter für printf:

Format-element	Typ	Ausgabe
%d	int	Dezimalzahl
%f	float, double	Gleitkommazahl
%c	char	Zeichen
%s	char*	Zeichenkette
%p	*	Pointeradresse

- Nützliche String-Funktionen:
  - get_s(text, size): Liest einen Text von der Kommandozeile ein.
  - strcpy(dest, src): Kopiert einen String.
  - strcat(text1, text2): Verkettet zwei Strings.
  - strcmp(text1, text2): Vergleicht zwei Strings.

Variablen, Datentypen und Operatoren
Bitoperatoren

- In C ist es möglich Operationen auf Bitebene vorzunehmen. Hierfür können wir die folgenden Bitoperationen auf Daten anwenden:
  int A=60; //0011 1100
  int B=13; //0000 1101

Operation	Bezeichnung	Beispiel
&	Binärer UND Operator kopiert ein Bit in das Ergebnis, wenn es in beiden Operanten existiert	(A&B) = 0000 1100 = 12
|	Binärer ODER Operator kopiert ein Bit in das Ergebnis, wenn es in einem der beiden Operanten existiert	(A|B) = 0011 1101 = 61
^	Binärer XOR Operator kopiert ein Bit in das Ergebnis, wenn es in einem der beiden Operanten existiert	(A^B) = 0011 0001 = 49
~	Binäres Komplement, bildet das Komplement.	~A = 1100 0011 = 61
<<	Binäre Linksverschiebung	A<<2 = 1111 0000 = 240
>>	Binäre Rechtsverschiebung	A>>2 = 0000 1111 = 15

Variablen, Datentypen und Operatoren
Bitoperatoren

```
int A=60; //0011 1100
int x=1;
```

Bit setzen	A |= 1 << x	0011 1110
Bit löschen	A &= ~(1 << x)	0011 1100
Bit toggeln	A ^= 1 << x	0011 1110
Bit checken	bit = (A >> x) & 1	0

Kontrollstrukturen
if-else / switch

- if (expression){}else{}
- switch(expression { case 1: statements; break; case 2:….}

  Beispiel:

```
char answer = 0;
printf("Enter Y or N: ");
scanf(" %c", &answer);
switch (answer){
   case 'y': case 'Y':
   printf("\n affirmative.");
   break;
   case 'n': case 'N':
   printf("\n  negative.");
   break;
   default:
   printf("\n  not respond correctly...");
   break;
   }
```

Kontrollstrukturen
Switch - ASCII

Beispiel:

```
char answer = 0;
printf("Enter Y or N: ");
scanf(" %c", &answer);
switch (answer){
   case 121: case 89:
   printf("\n affirmative.");
   break;
   case 110: case 78:
   printf("\n  negative.");
   break;
   default:
   printf("\n  not respond correctly...");
   break;
   }
```

Kontrollstrukturen while/for

- while(expression){statement}
- for(expr1; expr2; expr3){}
  Wichtig: Variableninitialisierung im Loop ist erst ab c99 möglich.
  D.h. <c99

```
int i;
for(i=0;i<10;i++)
```

ab c99

```
for(int i=0;i<10;i++)
```

```
=>-std=c99
```

Funktionen und Programmstruktur

```
#include<stdio.h>

amazingFunction(int val){
printf(“This function is amazing. Val: %d\n”,val);
}

int main(int argc, char* argv[]){
printf(“Hello World!\n”);
amazingFunction(2);
return 0;
}
```

Funktionen und Programmstruktur

```
#include<stdio.h>

int main(int argc, char* argv[]){
printf("Hello World!\n");
amazingFunction(2);
return 0;
}

amazingFunction(int val){
printf("This function is amazing. Val: %d\n",val);
}
```

Funktioniert so nicht!
In ‚main' ist ‚amazingFunction' noch unbekannt!

Funktionen und Programmstruktur

```
#include<stdio.h>

amazingFunction(int val);

int main(int argc, char* argv[]){
printf("Hello World!\n");
amazingFunction(2);
return 0;
}

amazingFunction(int val){
printf("This function is amazing. Val: %d\n",val);
}
```

Funktionen und Programmstruktur
Prototypen

- Bei Programmen mit vielen Funktionen ist es sinnvoll diese auszugliedern.

```
#include<stdio.h>
#include „amazing.h“

int main(int argc, argv*[]){
amazingFunction(2);
…
}
```

main.c

```
amazingFuntion(int val);
```

amazing.h

```
#include<stdio.h>
#include „amazing.h“ //not
necessary
amazingFuntion(int val){
printf(„amazing %d\n“,val);
}
```

amazing.c

```
gcc -o my_app main.c amazing.c
```

Zeiger und Arrays

- Bisher haben wir Variablen immer mit ihrem Namen angesprochen. Eine weitere Möglichkeit bieten sogenannte Pointer. Sie spiegeln die Adresse einer Variablen wieder.
- Um den Pointer zu erhalten nutzen wir den Referenzoperator „&“
- Um an den Wert eines Pointers zu gelangen nutzen wir den „*“ Operator.
- Variablen welche einen Pointer enthalten werden zusätzlich mit „*“ gekennzeichnet.
- Beispiel:

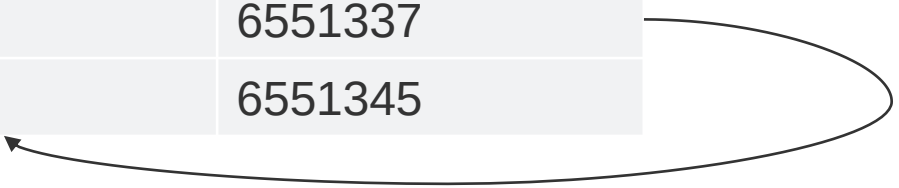
```
int i=5;
int* ptr=&i;
printf(„i has the address %p and the value %d\n“, ptr, i);
```

Zeiger und Arrays

- Beispiel:

```
int i=5;
int* ptr=&i;
printf(„i has the address %p and the value %d\n“, ptr, *ptr);
```

Variable	i	ptr
Wert	5	6551337
Adresse	6551337	6551345



- *ptr bedeutet also: Gehe zu Speicherstelle 6551337 und nimm den Wert, der sich dort befindet

Zeiger und Arrays

- Wozu benötige ich diese komplizierten Pointer?
  Funktionsparameter sind immer nur Kopien. Die folgende Funktion „swap“ bringt in der main-Funktion also rein garnichts

```
swap(int a, int b){
        int tmp=a;
        a=b;
        b=tmp;
}
int main(int argc, char* argv[]){
        int a=10, b=20;
        swap(a,b);
        printf(“a is %d b is %d\n”,a,b);
        return 0;
}
```

Zeiger und Arrays

- Deshalb müssen wir den Funktionen die Adressen der Variablen übergeben.

```
swap(int* a, int* b){
        int tmp=*a;
        *a=*b;
        *b=tmp;
}

int main(int argc, char* argv[]){
        int a=10, b=20;
        swap(&a,&b);
        printf("a is %d b is %d\n",a,b);
        return 0;
}
```

Zeiger und Arrays

- Auf Pointern können auch Rechenoperationen angewandt werden!

```
main(int argc, char* argv[]){
int x=1337;
int* ptr;
ptr=&x;
printf(„Wert von ptr: %p\n“,ptr);
ptr++;
printf(„Wert von ptr: %p\n“,ptr);
Ausgabe:
Wert von ptr 6554242
Wert von ptr 6554250
```

- Pointerarithmetik findet automatisch immer mit der Größe des angegebenen Typs statt.

Zeiger und Arrays

- Zeiger und Arrays sind in C sehr eng miteinander verwandt. Der Name eines Arrays ist ein Pointer auf das erste Element des Arrays. Dies bedeutet aber nicht, dass Pointer und Arrays gleich sind.
  Die letzten Zeilen sind identisch:

```
int arr[10];
arr;
&arr[0];
```

Zeiger und Arrays

- Auf den Arraypointer kann ebenfalls die Pointerarithmetik angewandt werden.
  Beispiel:

```
int arr[3]={1,2,3};
printf(„arr[0] ist %d\n“, *arr);
printf(„arr[1] ist %d\n“, *(arr+1));
printf(„arr[2] ist %d\n“, *(arr+2));

printf(„arr[0] ist %d\n“, arr[0]);
printf(„arr[1] ist %d\n“, arr[1]);
printf(„arr[2] ist %d\n“, arr[2]);
```

Zeiger und Arrays

- Wichtig: Arrays können nur ab c99 variabel initialisiert werden. Beispiel

```
int arr[n];
//nur in c99 möglich, davor nur mit Zahlen, welche
zur Kompilezeit feststehen.
```

Zeiger und Arrays

- Wie wurden vor c99 variable Arrays erstellt?
  Antwort: Mit Pointern und malloc
  Beispiel:

```
int* arr=malloc(sizeof(int)*3);
*arr=1;
*(arr+1)=2;
*(arr+2)=3;
printf("Zweiter Wert ist %d\n",*(arr+1));
printf("Zweiter Wert ist %d\n",arr[1]);
free(arr);
```

Zeiger und Arrays

- C bietet auch die Möglichkeit für mehrdimensionale Arrays. Beispiel:

```
#include <stdio.h>
int main () {
/* an array with 5 rows and 2 columns*/
int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
int i, j;
/* output each array element's value */
for ( i = 0; i < 5; i++ ) {
       for ( j = 0; j < 2; j++ ) {
       printf("a[%d][%d] = %d\n", i,j, a[i][j] );
       }
}
return 0;
}
```

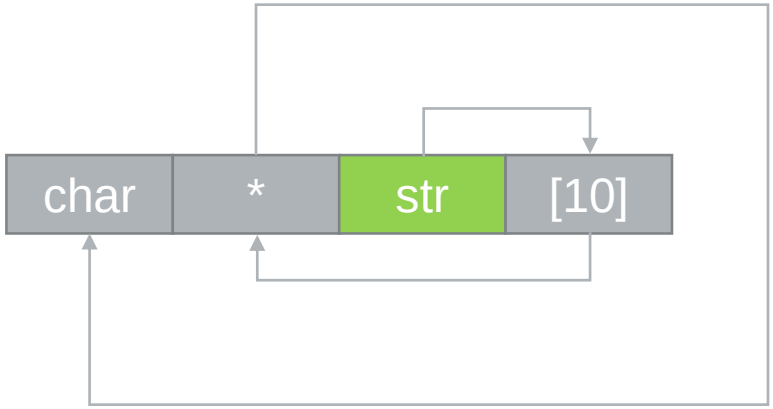
Zeiger und Arrays

- Variable mehrdimensionale Arrays können ebenfalls über malloc erzeugt werden.
  Beispiel:

```
int main () {
int c=3;
int r=2;
int** ptr = malloc(sizeof(int*)*r);
int i;
for(i=0;i<r;i++){
*(ptr+i)=malloc(sizeof(int)*c);
}
ptr[1][2]=10;
printf("val %d\n",ptr[1][2]);
return 0;
}
```

Zeiger und Arrays

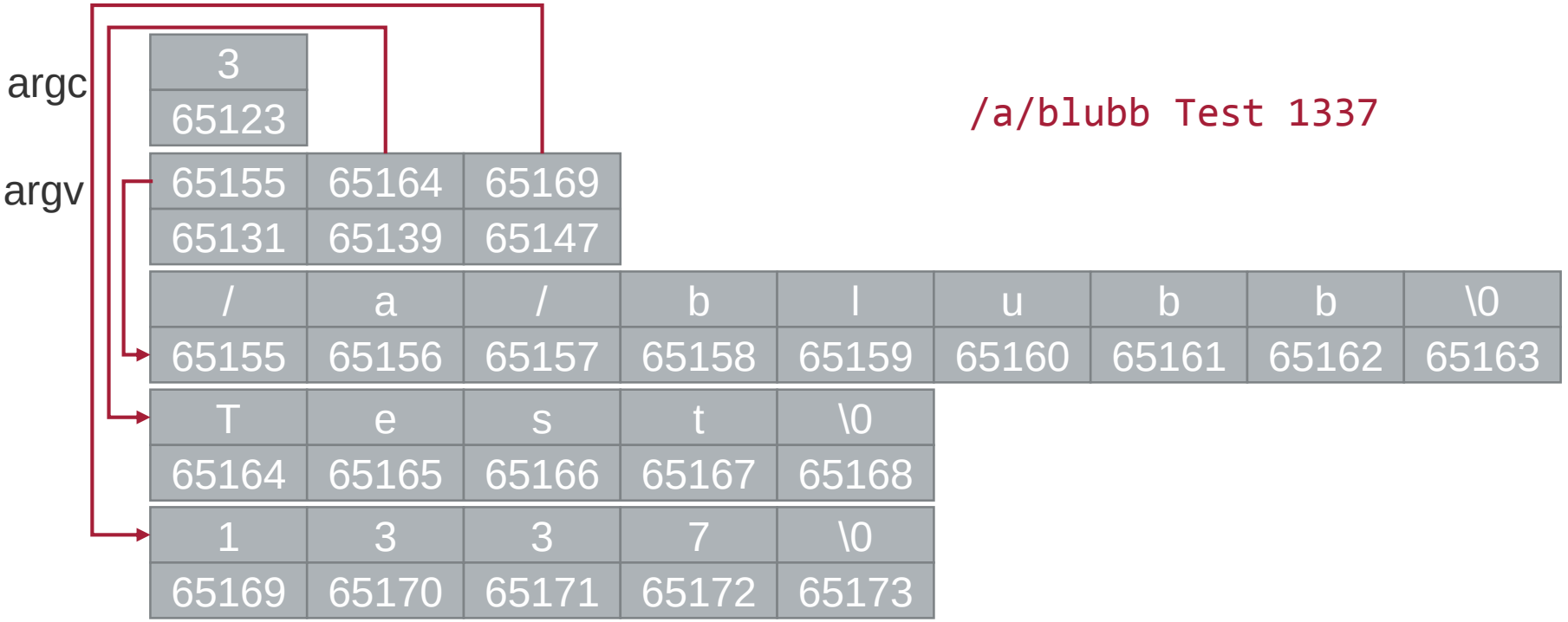
- Wie finde ich heraus, was für einen Typ Variable XY hat?
  Beispiel: char* str[10];
- Lösung: Die Clockwise spiral rule



str ist ein Array (Größe 10) aus Pointer auf char

Zeiger und Arrays

- Selbst in einem sehr simplen Programm müssen wir mit Arrays umgehen: Den Kommandozeilenparametern
  Beispiel: `main(int argc, char* argv[])`



Zeiger und Arrays

Abfragen der Kommandozeilenparameter:

```
printf(„Programmname %s\n“, argv[0]);
printf(„Erstes Argument %s\n“, argv[1]);
printf(„Zweites Argument %s\n“, argv[2]);
```

Strukturen

- C bietet die Möglichkeit, aus einfachen Datentypen wie etwa int oder char, benutzerdefinierte und komplexere Datentypen zu erstellen. Diese nennt man Strukturen (struct).
- Strukturen sind ein Hilfsmittel, um logisch zusammenhängende Daten zu gruppieren. In Java würden sie den Objekten entsprechen.

Struct
Deklaration/Definition

- Die Deklaration erfolgt mit dem Schlüsselwort struct und den einzelnen Variablen.
  Beispiel:

```
struct artikel{
int id;
char bezeichnung[100];
double preis;
};
```

- Beachten Sie das Semikolon am Ende.
- Die Definition erfolgt wie folgt:

```
struct artikel laptop;
```

Struct
Zugriff

- Für den Zugriff auf die Variablen der Struktur nutzen wir den „.“ Operator, bzw. „->“ bei Referenzen.
  Beispiel:

```
laptop.price=10.0;
printf(„price is %f\n“,laptop.preis);
```

oder bei Referenzen(beispielsweise als Funktionsparameter)

```
printThePrice(struct artikel* item){
printf(„price is %f\n“,laptop->preis);
}
```

Ein- und Ausgabe put-/getchar

- Funktionen für die Ein- und Ausgabe befinden sich in der Bibliothek <stdio.h>.
- Die einfachste Ein- und Ausgabe findet mit getchar() und putchar(char) statt. Hier werden einzelne Buchstaben auf die Kommandozeile geschrieben oder von ihr gelesen.
  Beispiel:

```
char c;
while((c=getchar()) != -1){
putchar(tolower(c));
}
```

Ein- und Ausgabe scanf/printf

- scanf und printf sind Funktionen für formatierte Ein- und Ausgaben. Printf haben wir bereits in vielen Beispielen kennengelernt. Analog zu printf formatiert und speichert scanf Eingaben der Kommandozeile.
  Beispiel:

```
int count=0;
printf(„Value of count is %d\n“, count);
printf(„Enter new value:\n“);
scanf(„%d“,&count);
printf(„Value of count is %d\n“, count);
```

Ein- und Ausgabe
Fehlerbehandlung

- Fehler sollten in jedem vernünftigen Programm abgefangen werden. In Java wären dies Exceptions. In C wird hierfür, zusätzlich zu den zwei Standardströmen (stdin, stdout), stderr sowie Exitscodes verwendet.
  Beispiel:

```
fprintf(stderr, „Opps“);
exit(2);
```

Makefiles

- Da es sehr umständlich ist, wenn man bei jedem Kompilieren und Linken alles in die Kommandozeile eingeben muss, bietet uns das Programm „make“ eine kleine Erleichterung. Make nutzt Makefiles. Beispiel:

```
CC=gcc
CFLAGS=-Wall -std=c99
DEBUG=-O0 -g
RELEASE=-O3
debug:
$(CC) $(CFLAGS) $(DEBUG) test.c -o test
stable:
$(CC) $(CFLAGS) $(RELEASE) stabletest.c -o stable
clean:
rm -vfr *~ test stable
```