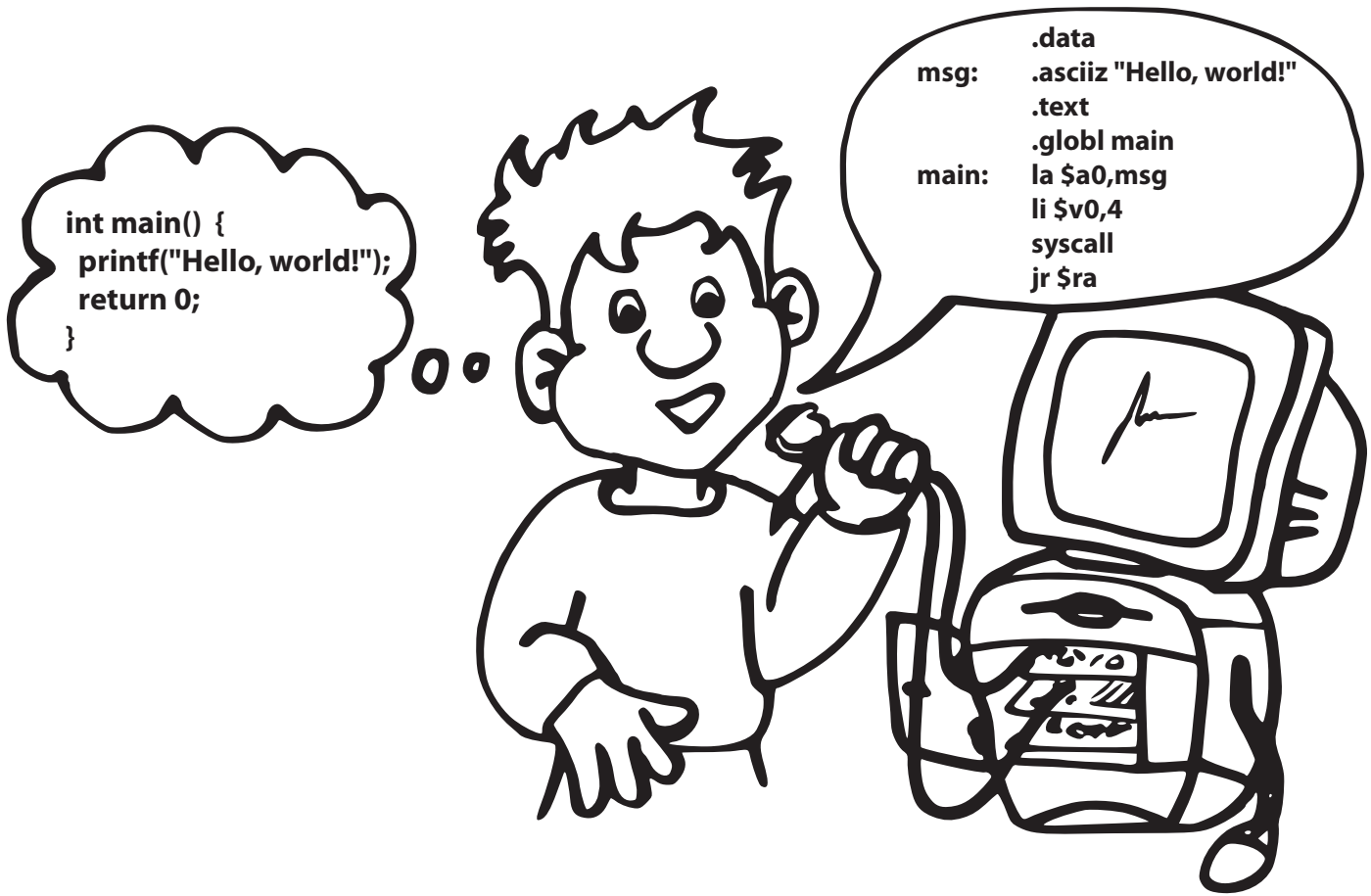
Grundlagen der Informatik III
Wintersemester 2010/2011 – 10. Vorlesung
B.Sc. Patrik Schmittat

TECHNISCHE UNIVERSITÄT DARMSTADT



Inhalt

TECHNISCHE UNIVERSITÄT DARMSTADT

1. Einführung in C

2. Zusammenfassung und Ausblick

Einführung in C
Zeiger

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zeiger können genutzt werden um Adressen von Objekten zu speichern
- Zeigerarithmetik sorgt für korrekte Verschiebung bei Grundrechenarten
- & und ∗ sind Referenzierungs- und Dereferenzierungoperator
- Zeiger selbst sind immer 4 Byte groß (auf einem 32 Bit-System)

```
int *p; // Integer-Zeiger
float *f1, f2, *f3; // f2 ist kein Zeiger! nur f1 und f3
main(int argc, char **argv) { .. }
main(int argc, char *argv[]) { .. } // Beide main sind identisch
// int a[] == int *a
p++; p--; p[3]; *p = 3; // Verwendung des Integer-Zeigers
int i;
p = &i;
p = 0; // Setzt Adresse nicht den Wert von i!
```

Einführung in C
call-by-value und call-by-reference

TECHNISCHE UNIVERSITÄT DARMSTADT

- Normalerweise werden Parameter via Kopie übergeben
- Danach existieren zwei verschiedene Stellen im Speicher

```
// call-by-value
void func(int i) { .. } // Parameter als Kopie
```

- Falls mehr als ein Rückgabewert ermöglicht werden soll, können Zeiger eingesetzt werden (andere Möglichkeit wäre `struct`)

```
// call-by-reference
void func(int *i) { .. } // Parameter als Zeiger
```

- Andere Beispiele haben wir bereits gesehen: `strcpy`, `strcat`, ...

Einführung in C
malloc, free und der Heap

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bisher lediglich Variablen auf dem Stack oder Konstanten in der Executable
- Es ist auch möglich Speicher auf dem Heap anzufordern
- Wichtig ist diesen auch wieder per Hand freizugeben! (falls nicht: Memoryleaks)

```
int n = 10; // Anzahl von Elementen
char *buffer; // Buffer fuer den allokiert wird

buffer = (char*)malloc(n);
if (buffer == NULL) // Falls kein Speicher mehr verfuegbar war
  exit(1);

free(buffer);
```

Einführung in C
Elementgröße, calloc und realloc

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bei einem `int*` muss bei der Allokation mit `sizeof(int)` multipliziert werden

```
int *ibuff;
ibuff = (int*)malloc(n * sizeof(int));

double *dbuff;
dbuff = (double*)malloc(n * sizeof(double));
```

- Weitere Funktionen sind `calloc` und `realloc`
- `calloc` allokiert `n` mal eine bestimmte Größe von Bytes und nullt den Speicher
- `realloc` verkleinert / -größert einen schon allokierten Speicher und behält dabei die alten Werte bei

Einführung in C
lokale Variablen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Bisher nur lokale Variablen

```
void func(int num) // lokale Variable als Parameter
{
  int i = 99; // lokale Variable
}
```

- Diese Art von Variable existiert nur während ihr Scope aktiv ist
- Danach ist die Variable sowie deren Speicher unbekannt / ungültig
- Zwei weitere Typen existieren die das Verlassen des Scopes “überleben”

Einführung in C
globale Variablen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Globale Variablen sind während der kompletten Programmausführung verfügbar

```
int i = 1; // Global
```

```
void func1()
{
   int i; // Lokal
   i = 3; // Lokal
}
```

```
void func2()
{
   i = 3; // Global
}
```

- Daher mit dem Scope und somit auch der Sichtbarkeit der Variablen aufpassen!

Einführung in C
statische Variablen

TECHNISCHE UNIVERSITÄT DARMSTADT

- Innerhalb einer Funktion angewendet “überlebt” die Variable den Funktionsaufruf
- Ist jedoch trotzdem nur innerhalb der Funktion sichtbar

```
void func()
{
  static int anz = 0; // Initialisierungswert ist 0
                      //   aber nicht bei jedem Aufruf der Funktio
  anz++; // Anzahl der Aufrufe von func zaehlen
}
```

- Statische Variablen haben unterschiedliche Bedeutungen
- Eine Version ist hier erklärt, die anderen sind dem Leser als Recherche überlassen

Einführung in C
lokale statische Variablen?!

TECHNISCHE UNIVERSITÄT DARMSTADT

- Lokale Variablen können manchmal auch fälschlicherweise als statische Variablen dienen
- Wichtig ist es Variablen immer nach der Deklaration zu definieren!

```
void func(int i)
{
  int n;
  printf("n: %d", n);
  n = i;
  printf(" - %d\n", n);
}
main()
{             // Ausgabe:
  func(1); // n: 32409823 - 1
  func(2); // n: 1 - 2
}
```

Einführung in C
Datei Ein- / Ausgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

- Datei Ein- / Ausgabe in C ist ein wenig aufwändig
- Jede Funktion erwartet Zeiger auf eine Datei (da kein OOP verfügbar)

```
#include <stdio.h>

main()
{
  FILE *file;
  char sentence[256];

  printf("String␣der␣angefuegt␣werden␣soll:␣");
  fgets(sentence, 255, stdin);
  file = fopen("sentence.txt", "a");
  fputs(sentence, file);
  fclose(file);
}
```

Einführung in C
Ein- / Ausgabe-Funktionen

TECHNISCHE UNIVERSITÄT DARMSTADT

```
// Oeffnen, Schliessen, Ausgabe
FILE *fopen(const char *filename, const char *mode);
int fclose(FILE *stream);
int fflush(FILE *stream);
// Status
int feof(FILE *stream);
int ferror(FILE *stream);
// Position(-swechsel)
int fgetpos(FILE *stream, fpos_t *position);
int fsetpos(FILE *stream, const fpos_t *pos);
long int ftell(FILE *stream);
int fseek(FILE *stream, long int offset, int origin);
// Text-Ein-~/~Ausgabe
int fputs(const char *str, FILE *stream);
int fprintf(FILE *stream, const char *format, ...);
char *fgets(char *str, int num, FILE *stream);
int fscanf(FILE *stream, const char *format, ...);
// Byte-Ein-~/~Ausgabe
size_t fread(void *ptr, size_t size, size_t count, FILE *stream);
size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);
```

Einführung in C
Argumente beim Aufruf

TECHNISCHE UNIVERSITÄT DARMSTADT

- Erstes Argument ist immer der Name des Programms
- Somit fangen die gegebenen Werte konkret bei 1 an
- `main(..)` besitzt nun eine andere Signatur

```c
#include <stdio.h>
#include <stdlib.h> // Fuer exit(..)

main(int argc, char **argv)
{
  while(argc--)
    printf("%s\n", *argv++);
  exit(EXIT_SUCCESS);
}
```

Einführung in C
Programme fehlerfrei bekommen

TECHNISCHE UNIVERSITÄT DARMSTADT

- `gdb` ist an dieser Stelle sehr hilfreich
- Dazu muss beim Aufruf von `gcc` ein `-g` eingefügt werden

```
gcc -g -o hello hello.c
gdb ./hello
```

- Danach befindet man sich in `gdb`
- Eine beispielhafte Nutzung von `gdb` ist auf der nächsten Folie aufgeführt
- `(gdb)` sowie alles ab `//` ist nicht einzugeben

Einführung in C
gdb Beispiel

TECHNISCHE UNIVERSITÄT DARMSTADT

```c
#include <stdio.h>

main()
{
  int array[] = {1, 2, 4, 8, 16};
  int i, n = 5;
  for (i = 0; i < n; i++)
  {
    printf("%d ", array[i]);
  }
  printf("\n");
}
```

```
(gdb) start // Laed das Programm ein
Breakpoint 1 at 0x80483e5: file for.c, line 5.
Starting program: /pfad/for
main () at for.c:5
5         int array[] = {1, 2, 4, 8, 16};
(gdb) step
6         int i, n = 5;
(gdb) s
7         for (i = 0; i < n; i++)
(gdb) info locals
array = {1, 2, 4, 8, 16}, i = 134513801, n = 5
(gdb) s
9           printf("%d ", array[i]);
...
(gdb) s
7         for (i = 0; i < n; i++)
(gdb) info locals
array = {1, 2, 4, 8, 16}, i = 1, n = 5
(gdb) continue
Continuing.
1 2 4 8 16

Program exited with code 012.
(gdb) quit
```

Zusammenfassung und Ausblick

TECHNISCHE UNIVERSITÄT DARMSTADT

- Zeiger, call-by-value / -reference
- malloc, free, calloc, realloc
- Lokale, globale, statische Variablen
- Datei Ein- / Ausgabe
- Argumente, gdb, Dokumentation

Diese / nächste Vorlesung behandelt

- Threads
- OpenMP