Betriebssysteme - Übungen
Der Ansi C-Standard

Franz Wiesinger

Hardware-Software-Design
FH Hagenberg

Der Ansi C-Standard

- C wurde 1972 von **Dennis Richie & Ken Thompson** erfunden.
- Für UNIX entwickelt, da vorher UNIX vollständig in Assembler implementiert war.
- Aufgrund vieler unterschiedlicher C-Dialekte -> folgte 1983 die Standardisierung (**American National Standard Institute – ANSI**).
- 1989 folgte der erste offizielle **C89-Standard** mit zugehörigen Software Bibliotheken (**ANSI C Standard Library**).
- Der Hauptvorteil des ANSI C-Standards ist die **Portabilität von C-Programmen**.
- Hierbei ist die **Neuübersetzung des Quellcodes** gemeint und nicht das ausführbare Programm.
- ANSI C-Compiler gibt es für eine **Vielzahl an Systemen**, vom kleinsten 8-Bit-Computer bis hin zum Superrechner.
- Nichtportabel sind hardwarenahe bzw. betriebssystemspezifische Operationen (z.B ein Linux-Programm das die Grafikkarte anspricht).

Was gibt es in C++, nicht aber in C?

- kein OOP (keine Klassen, keine Polymorphie, kein stat./dynam. Datentyp, etc)
- keine Templates
- keine STL -> keine Container, Algorithmen, Iteratoren, Funktoren, ...
- kein generisches Programmieren
- keine C++ Standard Library (keine IO-Streams, keine Manipulatoren, ...)
- kein std::string -> C-String ist Character-Array
- kein Function- und Operator-Overloading
- keine dynamic/static/reinterpret-casts -> sondern C-typecasts
- keine Referenzen
- keine Defaultparameter
- nur Blockkommentare in K&R-C (C89); Zeilenkommentare in ANSI-C (ab C99) erlaubt.
- keine namespaces

Hello World

```c
#include <stdio.h> /*eine neue Bibliothek*/

int main() {

   printf("Hello World!\n");

   /* oder mit lokaler Variable */
   char Message[] = "Hello World!"; /*kein Datentyp string, sondern char-Array */

   /* Ausgabe mit der printf(...) Funktion => kein Ausgabestrom cout */
   /* newline mit \n => keine Manipulatoren in C */
   printf("%s\n", Message);  /* %s -> Platzhalter für String */

   return 0;
}
```

C ist ebenso case-sensitive und hat gleiche Kontrollstrukturen wie C++.

Ausgabe auf der Konsole mit printf()

- print formatet => **printf("Formatangaben", Liste von Argumenten)**

```
int    i      = 10;
char   ch     = 'A';
char   Text[] = "Hallo";
double dbl    = 2.718;
int   *pi     = &i;        // pointer

// Ausgabe mit printf()
// --------------------

printf("i = %d",    i   );  printf("\n");  // i = 10
printf("ch = %c",   ch  );  printf("\n");  // ch = A
printf("Text = %s",Text);  printf("\n");  // Text = Hallo
printf("dbl = %f", dbl );  printf("\n");  // dbl = 2.718000
printf("pi = %p",   pi  );  printf("\n");  // pi = 0012FED4

/* mehrere Argumente */
printf("%d + %d = %d", i, i, i+i);  printf("\n");
                    // 10 + 10 = 20
// Ausrichtung
printf("i = %5d\n", i);    // rechtsbuendig bei 5 Zeichen:
                    // i =    10
```

Einlesen mit scanf()

- scan formatet => **scanf("Formatangaben", Variablenliste)**

```
int nr  = 0;
int val = 0;

printf("Zahl eingeben: ");

nr = scanf("%d", &val);     // Adresse der Variable val!

if (nr != 1) {
    printf("Eingabefehler");
}

//mehrere Werte eingeben
printf("Tag, Monat, Jahr eingeben:\n");

nr = scanf("%d %s %d", &day, month, &year);
            // kein & bei month (da char-array!)

if (nr == 3) {
   printf("%d. %s %d\n", day, month, year);
}

//Aus der Eingabe:  15 Nov 2021  wird:  15. Nov 2021
```

Formatiert in einen String schreiben und lesen

- string print formatet => **sprintf(String, "Formatangaben", Argumentliste)**
- string scan formatet => **sscanf(String, "Formatangaben", Variablenliste)**

```
//Konvertierung von Zahl in Zeichenkette (String)

val = 123;
sprintf(str, "%d", val);  /* str = "123" */
printf("%s", str);

//siehe auch:  atoi(), itoa() und Varianten davon (stdlib.h)


//Aus einem String lesen / konvertieren
char Line[] = {"16 Nov 2021"};
nr = sscanf(Line, "%d %s %d", &day, month, &year);

if (nr == 3) {
   printf("%d. %s %d\n", day, month, year);
}

//liest von Line formatiert die 3 Elemente aus und legt sie in den Variablen ab.
```

Dateioperationen

- Es gibt keine Dateiströme wie ofstream, ifstream, ...
- Der Zugriff erfolg mit Funktionen wie fopen, fread, fwrite, fclose, ...

```
//eine Datei read-only oeffnen

FILE *fp = 0;
fp = fopen("File.txt", "r");

if (p != 0) {
   //lesen von der Datei
}
```

Wir beschäftigen uns später noch genauer mit Dateioperationen!

C-Strings

- Kein Datentyp std::string, sondern Character-Array.
- Ist eine nullterminierte Folge von Zeichen in einem Array.
- In **string.h** sind eine Menge String-Funktionen verfügbar.

```
char Text[] = "Hallo";
   // -> Feld mit 6 Elementen:  H  a  l  l  o  \0

int len = strlen(Text);     // 5
strcpy(Text, "ABC");

if (strcmp(Text, "Hallo") == 0) {  //liefert <0, 0 und >0
   printf(Text);
}

strcat(Text, "DE");

printf("text: %s\n", Text);              // text: ABCDE
printf("len: %d\n", strlen(Text));       // len: 5
printf("size: %d\n", sizeof(Text));      // size: 6

char* sub = strstr(Text, "CD");
if (sub != 0) {
   printf("substr: %s\n", sub);          // substr: CDE
}
```

Pointerarithmetik und Funktionspointer (1)

```
// schreiben in eine Log-Datei
void Log(char* str) {
    // fopen, fprintf, fclose
}

// schreiben auf stdout
void Print(char* str) {
    printf("print: %s\n", str);
}
```

```
char Text[] = "Hallo";
char *ptr = Text;
ptr++;
*ptr ='e';
printf("%s", Text);      // -> Hello

// Funktionspointer deklarieren
void (*Printptr)(char* str);

// Adresse der Funktion an den Funktionspointer zuweisen
Printptr = Print;
Printptr(Text);  //ruft via Zeiger die Funktion Print() auf -> schreibt Hello auf stdout.
Printptr = Log;
Printptr(Text);  //ruft via Zeiger die Funktion Log() auf -> schreibt Hello in die Log-Datei.
```

Pointerarithmetik und Funktionspointer (2)

```
//besser: Funktionspointer via typedef deklarieren -> Datentyp
typedef void (*TFktPtr)(char * str); // TFktPtr ist ein Datentyp

//Beispiel: Funktion mit Funktionspointer als Parameter
void Output(TFktPtr func, char const* text)
{
   func(text);
}
```

```
TFktPtr Fptr;      // Fptr ist eine Variable

Fptr = Print;      // Zuweisung der Funktionsadresse
Fptr(Text);        // Aufruf der Funktion

Fptr = Log;
Fptr(Text);

//oder
Output(Log, Text);    //Funktion wird via Parameter übergeben
Output(Print, Text);
```

Strukturen und Enumerationen

```
struct Point {
   int x;
   int y;
} var1, var2;          // -> Variablen!

struct Point var3;     // Keyword struct nötig!

// besser mit typedef
typedef struct {
   int x;
   int y;
} TPoint;           // ein Datentyp!

TPoint var4;        // wie in C++


enum State {sOn=1, sOff=0} st1;  // -> Variable
st1 = sOn;

enum State st2;
st2 = st1;

//besser mit typedef
typedef enum {On=0, Off=1} TState;
TState st3 = On;
st2 = st3;
```

Referenzen

- C kennt keine Referenzen.
- Call by Reference wird mit einem Zeiger realisiert.

```
void Foo(int* val) {
    *val += 4711;
}

void Hoo(char** text) {
    *text = "I'm Hoo";
}
```

```
int x = 1;

Foo(&x);                    //Übergabe der Adresse

printf("x = %d\n", x);      // -> x = 4712

//Referenzvariable => ist in C nicht möglich
int& y = x;

char* myText = "I'm Foo";
Hoo(&myText);               //Übergabe der Adresse eines Zeigers
puts(myText);               //put string to stdout -> I'm Hoo
```

Dynamische Speicherverwaltung

- statt new und delete gibt es die Funktionen **malloc** und **free** aus der **stdlib.h**.
- Die Speicheroperationen memcpy, memmove, memset, memcmp, ... sind schnell und effizient!

```
char *pch = 0;
pch = (char*) malloc(4);        // 4 Byte anfordern
if (pch != 0) {
  strcpy(pch, "ABC\n");
  puts(pch);
  free(pch); pch = 0;           // Speicher freigeben
}

typedef struct Point {
    int x,y;
} TPoint;

TPoint p;
memset(&p, 0, sizeof(TPoint));
/* befüllt den angegebenen Speicher mit dem konstanten Byte 0
-> es wird byteweise initialisiert!!! */

char str1[7] = "aabbcc";
printf( "The string: %s\n", str1 );
memcpy( str1, str1+2, 4 );
printf( "New string: %s\n", str1 );    // -> New string: bbcccc
```

Weiterführendes

- Interessante Libraries sind: stdlib.h, ctype.h, limits.h, string.h, stdio.h.
- Literatur: The C Programming Language; Kernighan/Ritchie; Prentice Hall; 0-131-10362-8
- `http://www.cplusplus.com/reference/clibrary/`