C

> "Vergiß in keinem Falle,
> auch dann nicht, wenn vieles mißlingt:
> Die Gescheiten werden nicht alle!
> (So unwahrscheinlich das klingt.)"
> — *Erich Kästner*

1. Was ist C ?

C ist eine Programmiersprache, die Mitte der siebziger Jahre von Brian Kernighan und Dennis Ritchie (K&R) entwickelt [5] und seitdem deutlich erweitert und verbessert wurde. Vorgänger von C waren die beiden Sprachen BCPL und B. (Frage: Wie wird der Nachfolger von C heißen, D oder P?)
C ist im Gegensatz zu Pascal und LOGO nicht zu Lehrzwecken geschrieben worden, sondern als "ernsthafte" Programmiersprache, daher wurde auf Beschränkungen, die den Programmierer vor seinen eigenen Fehlern schützen sollen, abgesehen. Dies heißt jedoch nicht (entgegen einer häufig anzutreffenden Meinung, vgl. z.B. [3]), daß C deshalb unübersichtlich, schwer zu debuggen, oder gar vollkommen undurchschaubar ist. Es ist natürlich auch in C (leider?) möglich, Spaghetticode zu produzieren; hierin unterscheidet es sich jedoch nicht von Pascal, FDOS-BASIC oder INTERCAL. Entscheidend ist jedoch, daß es auch möglich ist, in C Programme zu schreiben, die sowohl *effizient* als auch *strukturiert* sind. Insofern beinhaltet C das Beste beider Welten, von Assembler und von höheren Programmiersprachen wie Pascal.
Es gibt natürlich auch Programmierer, die nichtsdestoweniger unübersichtlich und unstrukturiert programmieren (in welcher Sprache auch immer). Für sie wäre es vielleicht sinnvoller, den Fehler bei sich selbst und nicht bei der verwendeten Programmiersprache zu suchen (falls sie nicht gerade für den International Obfuscated C Code Contest (IOCCC) schreiben) ...

> "Was ist die beste Methode, Pascal zu beschleunigen?"
> — "$9.87\frac{m}{s^2}$"

2. Wie sieht C aus?

Ich will hier ein kleines Programm in Pascal und in C vorstellen, um Unterschiede der beiden Programmiersprachen aufzuzeigen.

C

```

#include <stdio.h>

int ist_primzahl(int zahl)
{
	int i;

	for (i=2;i<=(zahl/2);i++) {
		if ((zahl % i) == 0)
			return 0;
	}
	return 1;
}

void main()
{
	int max,i;

	printf("Höchste Zahl ?  ");
	scanf("%i",&max);

	printf("2 ");
	for (i=0;i<=max;i++) {
```

Pascal

```
program primzahlen;


function ist_primzahl(zahl:integer):boolean;
var i:integer;
begin;
	ist_primzahl := true;
	for i:= 2 to (zahl/2) do begin
		if (zahl mod i) = 0 then
			ist_primzahl := false;
	end;

end;

{Hauptprogramm}
var i,max:integer;
begin;

	write('Höchste Zahl ?  ');
	readln(max);

	write('2 ');
	for i:= 3 to max do begin;
```

```
24              if (ist_primzahl(i))                        if ist_primzahl(i) then
25                   printf("%i ",i);                            write(i,' ');
26         }                                           end;
27         printf("\n");                               writeln;
28}                                            end.
```

Man sieht, daß beide Programme sich stark ähneln. Dennoch gibt es einige auffällige Unterschiede:

Zeile 1: C kennt keine `program`-Anweisung, deren Nutzen ohnehin zweifelhaft ist. `:-)`

Zeile 2: Diese Zeile sorgt dafür, daß der C-Compiler die Anzahl und Art der Parameter (die sog. Prototypen) der I/O-Funktionen der Standardbibliothek einliest. Der Compiler hat so die Möglichkeit, Funktionen auf korrekten Aufruf zu überprüfen.
Pascal verwendet einen anderen Ansatz: die Aufrufart von Prozeduren wie `writeln` und `readln` sind dem Compiler fest eingebaut. Eine Möglichkeit zur Erweiterung dieses Mechanismus für eigene Funktionen besteht (zumindest in Standard-Pascal) nicht. In Turbo Pascal kann dieser Nachteil jedoch durch den Unit-Mechanismus im Wesentlichen ausgeglichen werden werden.

Zeile 3: C kennt keinen Datentyp `boolean`. Stattdessen wird der Typ `int` verwendet, wobei `1` dem Wahrheitswert `true` und `0` dem Wert `false` entsprechen soll.

Zeile 8: Die `for`-Anweisung ist in C sehr viel mächtiger und allgemeiner als in Pascal gefaßt. Allgemein hat sie die folgende Form:

```
for (expr1;expr2;expr3) {...}
```

Hierbei wird `expr1` einmalig beim Eintritt in die Schleife ausgeführt, `expr3` nach jedem Schleifendurchgang. `expr2` ist die Bedingung für den Nichtabbruch der Schleife. Man sieht, daß `for` in C eigentlich eine Mischung aller Schleifentypen darstellt.
Man kann das C-`for` in Pascal auf die folgende Art nachbilden:

```
expr1;
while (expr2) do begin
        ...
        expr3;
end;
```

Zeile 15: Das Hauptprogramm ist in C nichts anderes als eine gewöhnliche Funktion, die den Namen `main` trägt.

Zeile 24: Wie schon oben erwähnt, kennt C keinen Datentyp `boolean`. Die logischen Operatoren (wie `<` und `==`) liefern `int`s zurück, wobei `1` für `true` und `0` für `false` steht.
Auch Anweisungen wie `if` basieren auf diesen Konventionen, sie interpretieren `0` als falsch und jeden anderen Wert als wahr. Dementsprechend ist z.B. `if (0) {...}` immer falsch, `if (42) {...}` immer wahr, und bei `if (zahl % 3) {...}` wird der Teil in `{...}` exakt dann ausgeführt, wenn `zahl` nicht durch 3 teilbar ist (`%` ist der Modulo-Operator).
Ja, man kann sogar das folgende machen: `int a,b;` … `b = 3*(a == 2);`. Hier wird `b` der Wert 3 zugewiesen, wenn `a` gleich 2 ist, andernfalls der Wert 0.
Ich weiß, daß dies ein Graus für jeden rechtschaffenen Informatiker ist.

Dieses sind nur einige wenige Unterschiede. Dennoch wird klar, daß C in beinahe jeder Hinsicht flexibler ist als Pascal.

"Jede hinreichend weit fortgeschrittene Technologie ist nicht von Zauberei zu unterscheiden."

— *Arthur C. Clarke*

3. Der totale Horror?

```
/* von Byron Rakitzis und Sean Dorward, IOCCC 1990  */
#define D ,close(
char             *c,q             [512             ],m[             256
],*v[           99], **u,        *i[3];int        f[2],p;main       (){for
  (m[m        [60]=   m[62      ]=32   ]=m[*      m=124   [m]=       9]=6;
```

```
  e(-8)     ,gets      (1+(    c=q)     )||      exit      (0);     r(0,0)
   )for(    ;*++        c;);  }r(t,      o){    *i=i        [2]=    0;for
     (u=v  +98           ;m[*--c]         ^9;m [*c]          &32  ?i[*c
       &2]=                *u,u-            v^98               &&++u:
3 )if(!m[*c]){for(*++c=0;!m[*--c];);
* --u= ++c;}u-v^98?strcmp(*u,"cd")?*c?pipe(f),o=f[
1 ]:
4 ,(p=fork())?e(p),o?r(o,0)D o)D*f):
1 ,wait(0):(o?dup2(*f,0)D*f)D o):*i?
5 D 0),e(open(*i,0)):
9 ,t?dup2(t,1)D t):i[
2 ]?
6 D 1),e(creat(i[2],438)):
5 ,e(execvp(*u,u))):e(chdir(u[1])*2):
3 ;}e(x){x<0?write(2,"?\n$ "-x/4,2),x+1||exit(1):
5 ;}
```

Herr A.: Was um Himmels willen ist denn das?!?

Herr C.: Ein C-Programm.

Herr A.: Waaaaas? Und so soll ich programmieren?

Herr C.: Ja, wenn es nach gewissen Leuten ginge. Daran sieht man, daß 'C' für 'Cryptic' steht.

Herr A.: Erst wenn das Kultusministerium einen Erlaß herausbringt, der C zur allgemeinen Programmiersprache für den Informatikunterricht in der Oberstufe erklärt, werde ich eine Fortbildung zu dieser Sprache nehmen — *aber keine Sekunde früher!*

Herr C.: Weiter so! Dijkstra hatte Recht, als er sagte: "Der Gebrauch von COBOL verkrüppelt den Geist, diese Programmiersprache zu lehren, sollte deshalb als ein Gewaltverbrechen aufgefasst werden." — nur, daß es auf C genauso gut passt.

Herr A.: Wenn meine Schüler (vor allem dieser schreckliche Typ, der geradezu danach süchtig zu sein scheint) das jemals im Unterricht verwenden, dann ... dann ist die Scheisse am dampfen!

Herr C.: Im Editorial der c't August 1990 schrieb Detlef Grell: "Traut nur den Programmierern, die C im Herzen gram. [...] Er wird auch nicht 'Bei meiner Ehre' brüllen und zum Duelle schreiten, will ANSI oder C++ man ihm zur Seite stellen."[3] Ich finde, daß dies auch heute, nach über vier Jahren, noch vollkommen zutreffend ist.

Herr A.: *C Causes Catastrophally Constrained + Confusing Computers!* Auf daß wir niemals mit dem Fluch "C" geschlagen werden!
(Herr B. tritt hinzu.)

Herr B.: Was höre ich, Herr A.? Tut Ihnen Ihr Zeh weh?

Herr A.: Was ... oh, nein! Wir sprachen über die Programmiersprache C. Scheußlich! Schauen Sie sich das nur an:
(Herr A. zeigt Herrn B. das Blatt mit dem besten Utility des IOCCC 1990)
(Herr B. rückt seine Brille zurecht.)

Herr B.: Hmm ... das sieht wirklich nicht sehr klar aus. Aber andererseits, ist dies die einzige Möglichkeit, in C zu programmieren?

Herr C.: Nein, natürlich nicht. Zugegeben, es geht auch anders. Sehen Sie?
(Herr C. zeigt Herrn B. eine Gegenüberstellung eines Programms in C zur Berechnung von Primzahlen mit einem äquivalenten Programm in Pascal.)

Herr B.: *(Liest eine halbe Minute lang interessiert, dann:)* So besehen ist C gar nicht mal so schlimm. Dieses allgemeine `for`-Statement gefällt mir sogar, es zeugt von einem ausgesprochen klaren Blick des Entwicklers der Sprache für das Wesentliche. Was sagen Sie dazu?

Herr A.: *(Räuspert sich.)* Hmmmm ... ich weiß nicht, aber ich glaube, Sie haben recht. Ja, so gesehen ist es wirklich nicht so schlimm.

Herr B.: Was meinen Sie, Herr C.?

Herr C.: So schnell können Sie mich nicht überzeugen! Das `for`-Statement mag elegant sein, aber sehen Sie den Tatsachen doch ins Auge! Eine Programmiersprache, in der solche Programme möglich sind *(weist auf das IOCCC-Programm)*, ist absolut indiskutabel!

Herr B.: Solch grauenvolle Programme sind in C sicherlich möglich. Aber, wie das Primzahlen-Beispiel zeigt, ist kann man in C doch auch völlig übersichtliche und klare Programme schreiben (und solche Programme werden vornehmlich geschrieben), oder wollen Sie das abstreiten?

Herr C.: Nein, nein, aber ...

Herr B.: Ja, was haben Sie dann gegen C? Nun gut, es hat ein `goto`, aber das hat Pascal schließlich auch. Und, was den IOCCC-Gewinner anbetrifft, keiner zwingt Sie oder sonst irgendjemanden, so zu programmieren.

Herr C.: Nun, ja. Ich glaube, auch ich komme langsam auf den Geschmack. Die Macht von C ist wirklich verlockend ...

Herr A.: Ja, aber seien Sie unbesorgt! Es ist ja nicht die dunkle Seite der Macht! Und – wenn Sie immer noch nicht überzeugt sind, habe ich hier zwei Texte, die Sie sicher umstimmen werden. Der eine heißt "Why Pascal Is Not My Favorite Programming Language" [1], der andere ist ein Referat von einem meiner Schüler, das er kürzlich in meinem Unterricht vortrug [4]. Hier, lesen Sie!
(Herr C. nimmt "Why Pascal Is Not My Favorite Programming Language" in die Hand.)

Herr C.: Ich habe jetzt nicht die Konzentration, Englisch zu lesen, aber es klingt interessant. Lassen Sie mal den anderen Text sehen.
(Herr A. gibt ihm einen Text von ungefähr sieben zusammengehefteten DIN A4-Seiten, ausgedruckt auf einem Tintenstrahldrucker. Auf der ersten Seite prangt oben ein großes 'C'.)
(Alle drei lesen interessiert das Referat.)

Herr C.: Das ist alles sehr interessant. Ich weiß wirklich nicht, was ich sagen soll.

Herr B.: Komisch! In diesem Referat kommt sogar ein Dialog zwischen drei fiktiven Personen vor, die über C diskutieren und gegen Ende alle ziemlich plötzlich an C Gefallen finden.

Herr A.: Jetzt, wo ich es noch einmal lese ... ich frage mich, ob dieser Schüler wohl auf jemand angespielt hat, als er diese Personen erfand?

Herr C.: Nein, das glaube ich nicht. Das würde er nicht wagen. Nun ja, wie auch immer, ich glaube, Sie haben mich überzeugt. C ist wirklich eine elegante und schöne Programmiersprache. "C ist mächtig, leicht portierbar und maschinennah; in C geschriebene Programme sind schnell und kurz"[3] — ich glaube, das ist ein besser zutreffendes Zitat aus dem Editorial der c't 8/90.
(Herr A., Herr B. und Herr C. rufen laut im Chor:)

Alle: *C Controlliert Computer: Consequent; Couragiert; Charismatisch!*

"Es gibt zwei Methoden, fehlerfreie Programme zu schreiben. Nur die dritte funktioniert."

4. Warum C?

Die wesentlichen Vorteile von C sind:

a. Maschinennähe
C beinhaltet alle wesentlichen Sprachmerkmale zur Implementation von sehr systemsnaher Software (wie z.B. das Betriebssystem LINUX, das mit weniger als 1% Assembler-Code auskommt und ansonsten nur in C programmiert ist). Das heißt, daß Features wie Bitmanipulationen (z.B. logisches AND, OR, XOR, shift) zur Verfügung stehen. Pascal fehlen diese Möglichkeiten (obwohl Borland einige an Turbo-Pascal auf eine völlig inkompatible Weise angehängt hat).

b. Logisch schlüssige Konzeption
C ist eine kompakte Sprache, die mit wenigen Konstrukten und Schlüsselworten auskommt. Der Unterschied wird durch die folgenden drei Konstrukte in Pascal deutlich:

```
if ...  then begin
        ...
end;

for i:=a to b do begin
        ...
end;

repeat
        ...
```

```
until ...;
```

Wenn man es genau betrachtet, haben die drei Schlüsselworte `then`, `do` und `repeat` genau dieselbe Funktion, nämlich anzuzeigen, daß jetzt die von der Kontrollstruktur abhängige(n) Instruktion(en) folgen. Hinzu kommt die weitere Unregelmäßigkeit, daß nach `repeat` kein `begin` folgt, obgleich es einen ganzen Block von Anweisungen einschließt. In C sieht die Sache anders aus:

```
if (...)  {
        ...
}
for (a;b;c) {
        ...
}
do {
        ...
} while (...);
```

Abgesehen von der schon in Punkt 2 erwähnten Allgemeinheit der `for`-Anweisung verwendet C immer die geschweiften Klammern zur Abgrenzung der Blöcke. Ebenso werden keine überflüssige Füllwörter wie `then` und `do` benötigt; ihre Information ist in den geschweiften Klammern mit enthalten.

c. Eleganz
Aus der logischen Konsequenz folgt eine Art von Eleganz — die Regeln von C werden prinzipell und bis ins (scheinbar) Lächerliche durchgehalten. Hierzu ein Beispiel:
Arrays werden in C repräsentiert als Pointer auf das erste Element. Der Array-Zugriffsoperator `a[i]` ist definiert als `*(a + i)`. Da die Addition kommutativ ist, kann man also äquivalent zu `a[3]` auch schreiben: `3[a]`. Und dies funktioniert.
Dieses Beispiel mag eher trivial erscheinen. Es ist jedoch nur ein Beispiel und es existieren viele derartige Features in C, die die Sprache sehr viel übersichtlicher und interessanter machen.

d. Allgemeinheit und Einkapselung von Algorithmen
Die meisten Mechanismen in C sind sehr allgemein gehalten. Als Beispiel soll die Funktion `qsort()` aus der Standard-Library dienen, die Daten sortiert. Nun, welche Daten? Integer oder Strings oder was? Die Antwort ist, was immer der Programmierer will. Man übergibt (vereinfacht gesagt) einen Pointer auf ein Array des jeweiligen Typs und einen Pointer auf eine Funktion, die in der Lage ist, zwei Variablen dieses Typs zu vergleichen.
In Pascal müßte man stattdessen (u.a. wegen des Fehlens von Pointer auf Funktionen) für jeden Datentyp eine eigene Sortierfunktion (analog zu `qsort()`) schreiben, die, bis auf wenige Zeilen, alle identisch werden. C ermöglicht es, den gemeinsamen Teil dieser Funktionen (d.h. den eigentlichen Sortieralgorithmus) in einer allgemeinen Funktion einzukapseln.

e. Trennung von Sprache und Library
Pascal kennt einige Befehle zur Ein- und Ausgabe (wie z.B. `writeln`). Sie sind in der Sprache fest eingebaut und nicht durch eigene Routinen (z.B. zur Ausgabe in einem Fenster o.ä.) ersetzbar. Der Programmierer ist an sie gefesselt.
Selbstverständlich kennt auch C Möglichkeiten der I/O (`printf()` u.a.). Der entscheidende Unterschied ist, daß es sich hierbei um Funktionen der Standardbibliothek handelt. Sie sind nicht Befehle der Programmiersprache C und daher im Bedarfsfalle durch selbstgeschriebene Funktionen austauschbar.
Dieses Problem ist vielleicht am besten von Brian Kernighan in seinem Text *Why Pascal is Not My Favorite Programming Language* [1] formuliert worden, aus dem ich hier zitiere (Übersetzung von mir):

> *9. Es gibt keinen Ausweg*
> *Dieser letzte Punkt ist vielleicht der wichtigste. Die Programmiersprache [Pascal] ist für viele Zwecke nicht adäquat, aber begrenzt, weil es keine Möglichkeit gibt, die vorliegenden Mängel zu beheben. Es gibt beispielsweise keine casts, um die Typprüfung nötigenfalls aufzuheben. Es*

gibt keinen Weg, die unbrauchbare Bibliothek durch eine bessere zu ersetzen, es sei denn man verändert die "Standard-Prozeduren" und damit den Compiler. Die Sprache ist zugeschlossen.
Wer Pascal zur ernsthaften Programmierung verwendet, stürzt in eine tödliche Falle. Weil die Sprache impotent ist, muß sie erweitert werden. Aber jede Gruppe von Programmierern erweitert Pascal auf ihre eigene Art, um es wie die Sprache aussehen zu lassen, die sie eigentlich gerne verwendet hätte. Erweiterungen wie dateienweise Compilierung, FORTRAN-artiges `COMMON`*, String-Datentypen, interne statische Variablen, Initialisierung von Variablen, oktale Zahlen, Bitoperatoren usw. erhöhen den Nutzen der Sprache für eine Gruppe, zerstören jedoch die Portabilität für alle anderen.*
Ich bin der Meinung, daß es ein Fehler ist, Pascal für etwas anderes als dessen ursprünglichen Zweck einzusetzen. In seiner reinen Form ist Pascal eine Spielzeug-Programmiersprache, geeignet für den Unterricht, aber nicht zur ernsthaften Programmierung.

"Das Schöne an Standards ist, daß es so viele gibt, von denen man sich einen aussuchen kann."

— *Andrew S. Tanenbaum*

5. Quo vadis, C?

C ist nicht bei dem stehengeblieben, was Brian Kernighan und Dennis Ritchie Mitte der Siebziger Jahre formulierten [5]. 1985 wurde die Sprache vom ANSI standarisiert, wobei viele Erweiterungen und Ergänzungen aufgenommen wurden. In dieser Form wird C heute fast ausschließlich verwendet.
Es gibt mittlerweile auch eine objektorientierte Version von C, die 1988 von Bjarne Stroustrup entworfen wurde. Sie heißt C++ (was "Eines mehr als C" bedeuten soll). In ihr sind alle Prinzipien des objektorientierten Programmierens verankert; es handelt sich also nicht (wie bei den neueren Versionen von Turbo Pascal) um eine "halbe Sache".
C++ hat sich jedoch zum gegenwärtigen Zeitpunkt noch nicht durchgesetzt; es wird manchmal als zu umständlich kritisiert. Eine ANSI-Standarisierung ist geplant, aber bis jetzt noch nicht realisiert.

"Die Verbindung zwischen der Programmiersprache, die wir verwenden, und den Problemen und Lösungen, die wir uns vorstellen können, ist sehr eng. Sprachelemente zu entfernen mit der Absicht, Programmierfehler einzuschränken, ist daher bestenfalls gefährlich."

— *Bjarne Stroustrup*

"Whereas Europeans generally pronounce my name the right way ('Ni-klows Wirt'), Americans invariably mangle it into 'Nick-les Worth'. This is to say that Europeans call me by name, but Americans call me by value."

— *Nicklaus Wirth, Inventor of Pascal*

6. Warum C im Informatikunterricht?

Pascal ist speziell für den Informatikunterricht entwickelt worden. Insofern ist es naheliegend, diese Programmiersprache für diesen Zweck auch einzusetzen. Allerdings hat Pascal, wie oben geschildert wurde, einige gravierende Mängel, die den Einsatz für ernsthafte Programmierung unmöglich macht, obgleich (oder weil?) der Schüler vor den "harten" Problemen wie z.B. Pointerreferenzfehler geschützt ist.
Als Konsequenz muß jeder, der wirkliche Programme schreiben will, eine echte Programmiersprache (wie z.B. C, INTERCAL oder COBOL) lernen und wird dann doch mit diesen Tücken konfrontiert, denen er wegen des "Schutzes" durch Pascal bisher nicht begegnet ist. Da COBOL veraltet und INTERCAL ein wenig unpraktisch ist, bietet sich C an.
Daher — und wegen den anderen oben aufgeführten Mängel von Pascal — erachte ich es als sinnvoll, auch im Informatikunterricht eine Programmiersprache einzusetzen, die für die Verwendung in der Realität entworfen wurde.
Sollte dies in die Tat umgesetzt werden, so tritt natürlich (u.a.) das Problem der Beschaffung auf: die Pascal-Compiler waren sehr teuer; sollen wir die jetzt alle wegwerfen und nochmals Geld für C-Compiler ausgeben?

Dazu besteht kein Anlaß, es gibt einen frei verfügbaren C-Compiler, `gcc`, vom GNU-Projekt, der besser ist als die meisten kommerziellen Compiler. Zusammen mit dem Editor (Programmieroberfläche?) `emacs` würde sich ein ideales C-System ergeben — und das völlig kostenlos.

7. Zusammenfassung

C ist für die allermeisten Programmiervorhaben hervorragend geeignet. Pascal hingegen ist zwar für den Unterricht in Informatik an sich geeignet (und auch für diesen Zweck konzipiert), in der Praxis scheitert es jedoch kläglich (vgl. [1], [4]). Wie ich in Punkt 6 darlegte, hat m.E. eine solche Sprache *ipso facto* auch und vor allem im Informatikunterricht keinen Platz.
Aufgrund dieser Tatsachen ist die Programmiersprache C der Sprache Pascal (und vielen anderen) deutlich überlegen. Ihre Verwendung halte ich, auch in der Schule, für angebracht.

"Wenn ein Buch mit einem Kopf zusammenstößt und es klingt hohl, dann muß das nicht am Buch liegen."

— *Georg C. Lichtenberg*

8. Verwendete Literatur und Materialien

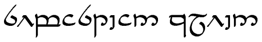
[1] Brian W. Kernighan, Why Pascal Is Not My Favorite Programming Language, Computer Science Technical Report No. 100, New Jersey 1981

[2] Das TEX Satzsystem von Donald E. Knuth, das in C entwickelt wurde und mit dem ich diesen Text gestaltet habe.

[3] Detlef Grell, c't magazin für computertechnik 8/90, S. 3

[4] Sebastian Klein, "C", Köln 1994

[5] Brian W. Kernighan und Dennis M. Ritchie, "The C Programming Language", New Jersey 1978



Sebastian Klein