1 Einleitung

1.1 Programmiersprachen

Programmiersprachen künstliche Sprachen, mit denen Algorithmen formuliert werden können.

Quellcode die Beschreibung eines Algorithmus in einer Programmiersprache; auch (Quell-)Programm genannt.

Programm (binäre, absolutierte) Datei, die ausführbar ist; auch (ausführbares) Programm genannt.

Maschinencode Anweisungen für die CPU. Wenn die dabei benutzten Adressen absolutiert sind, sind sie ausführbar.

Compiler Programm, das einen Quellcode lexikalisch, syntaktisch und semantisch analysiert und in Maschinencode übersetzt, wenn keine Fehler entdeckt worden sind.

Linker und Lader Programme, die Bibliotheksaufrufe des Maschinencodes realisieren und den Maschinencode absolutieren, so daß ein ausführbares Programm entsteht.

Interpreter Programm, das einen Quellcode Zeile für Zeile abarbeitet. Jede Zeile wird analysiert, – falls kein Fehler aufgetreten ist – in Maschinencode übersetzt, dann ausgeführt.

	Compiler		Linker/Lader	
Quellcode	$\longrightarrow$	Maschinencode	$\longrightarrow$	Programm

Symbolische Programmiersprachen gibt es seit 1954 (Fortran), es folgen (u. a.) Algol, Cobol, Apl, Lisp, Simula, Prolog, Pascal, Smalltalk, C (1973 von Ken Thompson aus den Vorläufern von BCPL und B entwickelt, mit der Entwicklung des Betriebssystems Unix von D. Ritchie ausgebaut, die Definition der Sprache stammt von Ritchie und B. Kernighan), Ada, C++ (1980), Java (1990).

Auf allen Rechnern wird mit dem GNU C-Compiler der ANSI-Standard eingehalten. Programme laufen daher (mit wenigen Ausnahmen) auf allen Rechnern identisch ab, die diesen Compiler implementiert haben.

1.2 Ein erstes Programm

Listing 1: hello.c (hello.c)

```
#include <stdio.h>                    /* Header-file stdio */

int main()                            /* Hauptprogramm */
{
                int otto;
                otto = otto+1;
        printf("Hello, world\n");
        return 0;
}
```

Um einfach arbeiten zu können, sollte man sich eine Prozedur **c** anlegen, die in einem Directory liegt, das stets durchsucht wird, i. a. ist das `$HOME/bin`. Man kann sie dorthin mit

```
/usr/home/public/vim/c+vimcopy
```

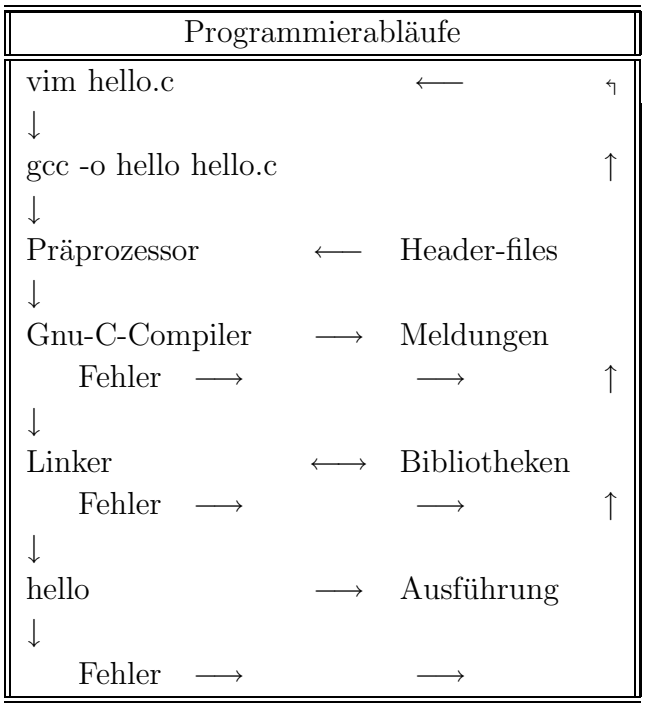
kopieren, gleichzeitig werden Dateien eingelesen, damit der Editor **vim** die Struktur von C-Programmen farblich unterstützt.

Listing 2: Shell Prozedur c (c)

```
#!/bin/sh
if [ $# = 0 ]
        then
        echo "Usage: c -options file"
        echo "    file.c wird in den vim geladen"
        echo "    und mit dem ANSI-Standard bersetzt"
        echo "    -noansi (nicht ANSI-Standard)"
        echo "    file ist dann ausfhrbar"
        exit 1
fi
GCC=/usr/local/gcc/bin/gcc
# weitere Optionen:
# -H fuer headerfile info
# -g debugging information
# -lbib mit libbib
OPTIONS=""
ANSI="-ansi -pedantic"
while [ $# -gt 0 ]
do
    case "$1" in
        -noansi) ANSI="";;
        -*) OPTIONS="$OPTIONS $1";;
        *) FILE=$1;
    esac
    shift
done
vim $FILE.c
if [ -f $FILE.c ]
        then
        $GCC -Wall $ANSI $OPTIONS -o $FILE $FILE.c
        echo $GCC -Wall $ANSI $OPTIONS -o $FILE $FILE.c
else
        echo "no file $FILE.c"
fi
```

Vorgehensweise

c hello ruft **vim hello.c** auf, jetzt kann das Programm geschrieben bzw. verändert werden. Nach Verlassen des Editors wird das Programm übersetzt, wenn keine Fehler aufgetreten sind, liegt eine ausführbare Datei *hello* vor, die durch Eingabe von *hello* ausgeführt wird.



Tools

indent -kr -i8 hello.c bringt *hello.c* in eine standardisierte Form, die angegebene Optionen orientieren sich an dem Stil von Kernighan und Ritchie, die ursprüngliche Programmquelle bleibt als *program.c* erhalten. Weitere Optionen findet man mit `man indent`.

splint -strict -modfilesys -ifblock -modunconnomods hello.c überprüft das Programm hello.c auf Fehler. Mit weiteren Parametern kann man den Übereifer des Programms in Grenzen halten.

1.3 Der Wortschatz

C Programme werden mit den 26 Klein- und Großbuchstaben der englischen Sprache gebildet, hinzu kommen 29 Sonderzeichen

```
! " # : . { } ^ % & ' ; [ ] ~ < = > ? | \ _ ( ) * + , - /
```

sowie Leer- und Tabulatorzeichen. Innerhalb von Zeichenfolgen, können auch andere Zeichen erscheinen, sofern sie vorhanden sind.
Kommentare werden in der Form

```
/* 1. Kommentarzeile
   2. Kommentarzeile */
```

geschrieben. Kommentare dürfen nicht geschachtelt werden.

Wortschatz (key-words)	
Datentypen	`char, double, float, int, void`
Datentyp oder Typ-Modifikation	`long, short, signed, unsigned`
Attribut für Datentyp	`volatile, const`
Speicherklassen	`auto, extern, register, static`
neue Datentypen	`typedef, struct`
Datenstruktur mit Alternativen	`union`
Aufzählungstyp Bezeichner	`enum`
Speichergröße von Variablen	`sizeof`
Teile von Schleifen	`for, while, do, continue, break`
Teile einer Auswahlanweisung	`switch, case, default, break`
Teile einer bedingten Anweisung	`if, else`
Sprunganweisung	`goto`
Rücksprung	`return`

Manchmal sind auch `fortran, asm` keywords. Früher war `entry` ein Schlüsselwort. Mit dem ANSI-Standard wurden `const, signed, volatile, enum, void` eingeführt.
Die restlichen Elemente der Sprache bestehen aus Eigennamen, die sich Programmierer für Variablen und Funktionen ausgedacht haben, verbunden mit Sonderzeichen zum Gruppieren, speziell mit dem ; zum Beenden einer Befehlszeile, und aus Operatoren und Zahlen.

Die Sprache sieht also recht arm aus, wir können keine Anweisung entdecken, die etwa einem `read` oder `write` entsprechen. Dies wird über Funktionen abgehandelt. Man muß deshalb nicht nur deren Namen und Aktion lernen, auch, in welchem Header-file sie stehen. Der muß dann dem Programm bekannt gemacht werden.
Anhand eines englischen Satzes sollen die unterschiedlichen Fehlerklassen einer Sprache dargestellt werden.

Falsche Orthographie Never change a runing programm,

Falsche Syntax Never running a program change,

Falsche Semantik Never program a running change,

Falsche Logik Never change a running program.

Fehler in der Schreibweise und Syntax werden i.a. vom Compiler entdeckt und es erfolgt eine Fehlermeldung. Semantische und logische Fehler können das Programm zum Absturz bringen. Solche Fehler nennt man Laufzeitfehler (z.B. Division durch 0). Häufig bleiben aber semantische oder logische Fehler unentdeckt! Um das zu vermeiden, sollte man unbedingt mit einem Syntax-Checker arbeiten, z.B. mit **splint**, und das Programm austesten.

B. Kernighan, D.M. Ritchie The C Programming Language, 2. ed, Addison Wesley, in Deutsch bei Hanser (nur die 2. Auflage beschreibt den ANSI-Standard).

J. Goll, U. Bröckl und M. Dausmann C als erste Programmiersprache. 4. Aufl., Teubner 2002,

D. Herrmann Effektiv Programmieren in C und C++, 5. Auflage, Vieweg, 2001.

1.4 Meta-Bezeichnungen

var bezeichnet Variablen. Namen, die man als Variablen, verwendet, dürfen keine key-words sein. Sie müssen mit einem Buchstaben oder dem _ beginnen und können sonst aus Groß- und Kleinbuchstaben sowie Ziffern bestehen, als Sonderzeichen ist allein _ erlaubt. Ihre Länge ist (i.a.) auf `31` Zeichen beschränkt, wird diese Länge überschritten, werden die restlichen Zeichen ignoriert. Variablen müssen initialisiert werden, andernfalls ist ihr Wert unbestimmt. Wir sprechen häufig (etwas ungenau) von Variablen, meinen aber ihren Wert.

Listing 3: Nicht initialisierte Variablen (nonint.c)

```
#include <stdio.h>

/* Ausgabe nicht initialisierter Variablen */

int main(void)
{
        static int lec;
        char bois, foret;
        int forest, wood;
        float holz, wald;
        double xylos;
        printf("int lec: %d, ", lec);
        printf("char: %c, %c, ", bois, foret);
        printf("int: %i, %i\n", forest, wood);
        printf("float: %g, %g, ", holz, wald);
        printf("double: %e\n", xylos);
        return 0;
}
```

Das Programm liefert als Output:

```
int lec: 0, char: Ü, í, int: 4, -4198956
float: 4.2039e-45, 9.35787e-41, double: 0.000000e+00
```

Nicht initialisierte Variablen erkennt der Syntax-Checker `splint`, man erhält dann Fehlermeldungen der Art: `Variable foret used before definition`.

typ bezeichnet einen Datentyp.

expression bezeichnet durch Operatoren verknüpfte Variablen, Funktionen oder Zahlen.

condition bezeichnet einen logischen Ausdruck (vom Typ `int`), der den Wert 0 hat, wenn er falsch ist, sonst einen Wert $\neq 0$.

Logische Operatoren					
`||`	vel	`&&`	et	!	non
Relationen					
`>=`	$\geq$	`<=`	$\leq$	`==`	$=$
`>`	$>$	`<`	$<$	`!=`	$\neq$

Beispiele:

`(result != 0)` Wert von `0` verschieden, wenn die Variable *result* von `0` verschieden ist,

`( x < 0 || 1 < x )` Wert von `0` verschieden, wenn die Variable x kleiner `0` oder größer `1`,

`( 0 < x && x < 1 )` Wert von `0` verschieden, wenn die Variable x in `(0,1)` liegt,

`( c % 4 = = 1)` Wert von `0` verschieden, wenn der Rest von c geteilt durch `4` den Wert `1` hat.

statement(s) bezeichnet einen bzw. mehrere C-Anweisungen.

func bezeichnet eine Funktion, die aus ihren Parametern einen Rückgabewert erzeugt. Parameter und Rückgabewert müssen wie folgt mit ihrem Datentyp bezeichnet werden.

typ func(typ1 var1, typ2 var2, . . . , typn, varn)
{
statements
`return` *var*
}

func gibt mit der letzten Anweisung der Inhalt von *var* vom Typ *typ* zurück.

Euklidischer Algorithmus

Sei $n_2 < n_1 \in \mathbb{N}$
und $\lambda = \text{ggt}(n_1, n_2)$.
$\lambda | n_1,\ \lambda | n_2 \Longrightarrow \lambda | (n_1 - n_2)$,
$0 < n_3 = n_1 - n_2 < n_1$
Ordne n_2, n_3 und fahre fort
...
$n_k = n_{k-1} \Longrightarrow \text{ggt}(n_1, n_2) = n_k$.

(12, 7)	(10, 16)	(12, 4)
(5, 7)	(10, 6)	(8, 4)
(5, 2)	(4, 6)	(4, 4)
(3, 2)	(4,2)	
(1, 2)	(2,2)	
(1, 1)		

Listing 4: Euklidischer Algorithmus (euklid1.c)

```
/* Euklidischer Algorithmus zur Bestimmung
   des ggT von n und m */
#include <stdio.h>

int main(void)
{
        int n, m;

        printf("\aEingabe n: ");          /* Ausgabe */
        (void) scanf("%d", &n); /* Einlesen von n */
        printf("Eingabe m: ");
        (void) scanf("%d", &m);
        printf("ggt(%d,%d) = ", n, m);

        while (n != m) {          /* While Schleife */
                if (n < m)
                        m = m - n;
                else
                        n = n - m;
        }
        printf("%d\n", n);        /* Ausgabe des ggT */
        return 0;
}
```

Listing 5: Euklidischer Algorithmus (euklid4.c)

```
#include <stdio.h>

int main(void)
{
        int m, n, r;
        printf("ggT(n,m) berechnen.\n");
        printf("Eingabe n, m: ");
        (void) scanf("%d %d", &n, &m);
        printf("ggt(%d,%d) = ", n, m);
        while (n > 0) {
                r = m%n;
                m = n;
                n = r;
        }
        printf("%d \n", m);
        return 0;
}
```