1 Einführung

1.1 Historisches zu C und C++

1.2 Struktur von C-Programmen

1.3 Obfuscated C Code Contest

1.4 Phasen der Übersetzung

1.5 Kommentare

1.6 Einige C-Snippets

1.7 Operatoren in C

1.8 Sequence Points in C

Entwicklung von C und C++

Vorgänger von C

BCPL **"Basic Combined Programming Language", entworfen und implementiert von Martin Richards etwa zwischen 1966 und 1967.**

B **Entworfen und implementiert von Kenneth L. Thompson etwa ab 1969.**

C

Erste Versionen von C wurden zwischen 1971 und 1973 von Dennis M. Ritchie entwickelt und implementiert. C diente auch als Systemprogrammiersprache für Unix.

1978 wurde das Buch "The C Programming Language" von Brian Kernighan und Dennis Ritchie publiziert.

1983 wurde das ANSI Komitee X3J11 gegründet, um die Sprache C zu standardisieren. 1989 erschien der Standard X3.159-1989, der ANSI-Standard wurde 1990 zum ISO-Standard ISO/IEC 9899:1990.

1999 erschien der Standard ISO/IEC 9899:1999, kurz C99, und 2011 erschien der Standard ISO/IEC 9899:2011, kurz C11.

C++

Entwickelt und implementiert von Bjarne Stroustrup etwa ab 1979 als "C with Classes".

1983 wird "C with Classes" umbenannt in C++.

Eine erste nahezu vollständige Beschreibung von C++ wurde von Margaret A. Ellis und Bjarne Stroustrup 1989 publiziert.

1989 wurde das ANSI-Komitee X3J16 zur Standardisierung von C++ gegründet.

1998 wurde die Norm ISO/IEC 14882:1998 verabschiedet.

2003 erschien ein leicht überarbeiteter Standard ISO/IEC 14882:2003.

In 2006 wurde der technische Bericht ISO/IEC TR 18015 herausgegeben, sein voller Titel lautet:
Information technology — Programming languages, their environments and system software interfaces — Technical Report on C++ Performance.

Der gegenwärtige Standard, genannt C++14, trägt die Bezeichnung International Standard ISO/IEC 14882: 2014(E) Programming Language C++. Eine Beschreibung findet man auch in n3936.

Bemerkung: In einer rückschauenden Betrachtung über die Entwicklung von C++ legt Herr Stroustrup besonderen Wert auf die Feststellung: "There is a direct mapping of C++ language constructs to hardware."

Aus "Rationale for Programming Language C":

Keep the spirit of C.

The Committee kept as a major goal to preserve the traditional spirit of C. There are many facets of the spirit of C, but the essence is a community sentiment of the underlying principles upon which the C language is based. Some of the facets of the spirit of C can be summarized in phrases like

- **Trust the programmer.**
- **Don't prevent the programmer from doing what needs to be done.**
- **Keep the language small and simple.**
- **Provide only one way to do an operation.**
- **Make it fast, even if it is not guaranteed to be portable.**

The last proverb needs a little explanation. The potential for efficient code generation is one of the most important strengths of C. To help ensure that no code explosion occurs for what appears to be a very simple operation, many operations are defined to be how the target machine's hardware does it rather than by a general abstract rule.

Programm aus BCPL-Manual: Freitag, der 13-te

```
GET "libhdr"

MANIFEST { mon=0; sun=6; jan=0; feb=1; dec=11 }

LET start() = VALOF
    {LET count       = TABLE 0, 0, 0, 0, 0, 0, 0
    LET daysinmonth = TABLE 31,  ?, 31, 30, 31, 30,
                            31, 31, 30, 31, 30, 31
    LET days = 0
    FOR year = 1973 TO 1973+399 DO
        {daysinmonth!feb := febdays(year)
        FOR month = jan TO dec DO
            { LET day13 = (days+12) REM 7
            count!day13 := count!day13 + 1
            days := days + daysinmonth!month
        }
    }
    FOR day = mon TO sun DO
        writef("%i3 %sdays*n",
        count!day,
        select(day,
            "Mon", "Tues", "Wednes", "Thurs",
            "Fri", "Sat", "Sun"))
    RESULTIS 0
}

AND febdays(year) = year REM 400 = 0 -> 29,
                    year REM 100 = 0 -> 28,
                    year REM 4 = 0   -> 29,
                    28

AND select(n, a0, a1, a2, a3, a4, a5, a6) = n!@a0
```

```
/*  Struktur eines C-Programms  */

#include  <stdio.h>
#include  <math.h>

double area (double);
double volume (double);


int main (void) {
        double radius   =  0.0;
        double flaeche  =  0.0;
        double kugel    =  0.0;


        radius   =  3.1;
        flaeche  =  area (radius);
        kugel    =  volume (radius);

        printf ("Radius = %4.2f\n"  "Oberflaeche = %7.4f\n"
                "Kugelvolumen = %10.6f\n\n",
                radius,  flaeche,  kugel);

        radius  = 6.2;
        flaeche = area (radius);
        kugel   = volume (radius);

        printf ("Radius = %4.2f\n"  "Oberflaeche = %7.4f\n"
                "Kugelvolumen = %10.6f\n\n",
                radius,  flaeche,  kugel);

return 0;
}
```

```
/*  Routinentexte  */

/*#include <math.h>*/
double area (double x) {
      const double pi = 4 * atan (1.0);
return   4.0 * pi * x * x;
}
/*#include <math.h>*/
double volume (double x) {
      const double pi = 4 * atan (1.0);
return   pi * x * x * x * 4.0 / 3.0;
}


/*  Ausgabe

Radius = 3.10
Oberflaeche = 120.7628
Kugelvolumen = 124.788249

Radius = 6.20
Oberflaeche = 483.0513
Kugelvolumen = 998.305992

*/
```

Bemerkung: Die Sprache C ermöglicht die separate Übersetzung von Teilprogrammen. So kann das obige Programm ohne weiteres auf drei Dateien verteilt werden, eine für das Hauptprogramm, eine für die Routine area und eine für die Routine volume.

Bemerkung: Es existiert ein "International Obfuscated C Code Contest", er wird vom 01. 09. 2014 bis zum 19. 10. 2014 zum 23. Mal durchgeführt. Eines seiner Ziele ist: To write the most Obscure/Obfuscated C program.

```
/*
      Ein "unverständliches" C-Programm ?
*/

      int i;

      main () {

            i = 0;

            for (    ;
                  i < 14;
                  writeone   (    i++
                                + "hello, world!\n"  )
                )
            ;
      }


      writeone  (i)  {
            write  (1,  i,  1);
      }
```

Frage: Was tut dies Programm?

Ein Beispiel aus dem dritten IOCCC von 1986,
Programm von Holloway:

```
#include "stdio.h"
#define   e 3
#define   g (e/e)
#define   h ((g+e)/2)
#define   f (e-g-h)
#define   j (e*e-g)
#define   k (j-h)
#define   l(x) tab2[x]/h
#define   m(n,a) ((n&(a))==(a))

long tab1[]={ 989L,5L,26L,0L,88319L,123L,0L,9367L };
int tab2[]={ 4,6,10,14,22,26,34,38,46,58,62,74,82,86 };

main(m1,s)
char *s;
{
   int a,b,c,d,o[k],n=(int)s;
   if(m1==1) {
        char b[2*j+f-g];
        main(l(h+e)+h+e,b);
        printf(b);
   }
   else switch(m1-=h) {
      case f:
         a=(b=(c=(d=g)<<g)<<g)<<g;
         return(m(n,a|c)|m(n,b)|m(n,a|d)|m(n,c|d));
```

```
        case h:
           for(a=f;a<j;++a)
               if(tab1[a]&&!(tab1[a]%((long)l(n))))return(a);
        case g:
           if(n<h)return(g);
           if(n<j){n-=g;c='D';o[f]=h;o[g]=f;}
           else{c='\r'-'\b';n-=j-g;o[f]=o[g]=g;}
           if((b=n)>=e)
               for(b=g<<g;b<n;++b)
                   o[b]=o[b-h]+o[b-g]+c;
           return(o[b-g]%n+k-h);
        default:
           if (m1-=e)
                main(m1-g+e+h,s+g);
           else
                *(s+g)=f;
           for(*s=a=f;a<e;)
               *s=(*s<<e)|main(h+a++,(char *)m1);
        }
}


/*  Ausgabe:

hello world!

*/
```

```
/* Nach Ersetzung der defines  */
#include <stdio.h>

long tab1[]={ 989L,5L,26L,0L,88319L,123L,0L,9367L };
int tab2[]={ 4,6,10,14,22,26,34,38,46,58,62,74,82,86 };

main(m1,s) char *s; {
   int a,b,c,d,o[6],n=(int)s;

   if(m1==1){ char b[15]; main(tab2[5]/2+5,b); printf(b); }
   else switch(m1-=2){
      case 0:
         a=(b=(c=(d=1)<<1)<<1)<<1;
         return ((n&(a|c))==(a|c))|((n&b)==b)|
                  ((n&(a|d))==(a|d))|((n&(c|d))==(c|d));
      case 2:
         for(a=0;a<8;++a)
            if(tab1[a]&&!(tab1[a]%((long)tab2[n]/2)))
               return(a);
      case 1:
         if(n<2)return(1);
         if(n<8){n-=1;c='D';o[0]=2;o[1]=0;}
         else{c='\r'-'\b';n-=7;o[0]=o[1]=1;}
         if((b=n)>=3)
            for(b=1<<1;b<n;++b)o[b]=o[b-2]+o[b-1]+c;
         return(o[b-1]%n+4);
      default:
          if(m1-=3) main(m1+4,s+1); else *(s+1)=0;
          for(*s=a=0;a<3;)
           *s=(*s<<3)|main(2+a++,(char *)m1);
      }
}
```

Phasen der Übersetzung

1 Die Außendarstellung von Programmtexten wird in den "basic source character set" überführt, insbesondere werden Trigraph-Codierungen ersetzt.

2 Zeichenpaare "\"-Zeichen new-line-Zeichen, die am Ende einer Zeile stehen, werden gelöscht. Hierdurch werden Zeilen verschmolzen.

3 Kommentare werden durch ein Leerzeichen ersetzt. Der Quelltext wird in eine Folge von "preprocessing tokens" und "white-space Zeichen" überführt.

4 Preprocessing Direktiven werden verarbeitet und Makro-Expansion findet statt. Durch #include-Direktiven eingebundene Texte werden rekursiv den Phasen 1 bis 4 unterzogen. Alle Preprocessing Direktiven werden nun gelöscht.

5 Die Codierung wird in den "execution character set" überführt.

6 Nebeneinanderstehende Literale werden konkateniert.

7 Die "preprocessing tokens" werden in Lexeme gewandelt. Die eigentliche Übersetzung findet statt.

8 Die Behandlung von Templates wird vollzogen.

9 Aus den einzelnen Übersetzungseinheiten wird ein lauffähiges Programm mittels eines Binders erstellt.

Trigraph-Codierungen:

??= #	??([	??< {
??/ \	??)]	??> }
??‘ ^	??! |	??- ~

Beispiel:

```
??=include <stdio.h>

int main (void)??<              /* { */
   char c??(5??);               /* [ und ] */

   c??(4??)  =  '0' - (??-0 ??' 1 ??! 2);          /* ~, ^ and | */

   printf("Zeichen = %c??/n",  c??(4??));
   printf ("Was !?!?!??!");
return 0;
??> /* } */
```

```
/*  Ausgabe:
Zeichen = 2
Was !?!?!|
*/
```

Bemerkung: In Visual Studio sollte für obiges Programm der Schalter /Zc:trigraphs benutzt werden.

Kommentare in C99

Beispiele aus dem Standard

```
#include <stdio.h>

//  Ein Einzeilenkommentar

/*     Ein Kommentar
       über mehrere Zeilen
*/

#define glue(x, y)   x##y

int  f = 1,  g = 2,  h = 3;


int main (void) {

       "a//b";    // Literal

        //  */       // normaler Kommentar

       f = g /**//h;
```

```
        //\
        i + 2                       // ein Zweizeilenkommentar

        /\
/       j(x);           // ein weiterer Zweizeilenkommentar

        glue (f , =)   7;

        printf ("1f = %d\n", f);

        glue (/,/)  ? Kommentar in VS2010
        //  glue (/,/)  stellt nach Standard Syntaxfehler dar,
        //  "//" ist kein "preprocessing token"

        f = g//**/o
           + h;

        printf ("2f = %d\n", f);

        f = (g//*h*/h++
             , h);

        printf ("3f = %d\n", f);

return 0;
}


/*  Ausgabe:
1f  =  7
2f  =  5
3f  =  3
*/
```

```
#include <stdio.h>

#define glue(x, y)  x##y

int  f = 1,  g = 2,  h = 3,  o = -1;

int main (void) {
       "a//b";                  /* Literal*/
       f = g /**// h;
       glue (f , =)   3;
       printf ("1f = %d\n", f);

       f = g//**/o
           + h;
       printf ("2f = %d\n", f);

       f = (g//*h*/h++
            , h);
       printf ("3f = %d\n", f);
return 0;
}

/*  Ausgabe nach
    cc -xc99=none comm01.c  unter Solaris
1f = 3
2f = 1
3f = 4

   Ausgabe von VS2010
1f = 3
2f = 5
3f = 3
*/
```

Die folgenden C-Snippets stammen aus Alan R. Feuer "The C Puzzle Book".

```
//  Programm 1
#include <stdio.h>

int main () {

    int x;
    x = - 3 + 4 * 5 - 6;
    printf ("x0 = %d\n", x);
    x = 3 + 4 % 5 - 6;
    printf ("x1 = %d\n", x);
    x = - 3 * 4 % - 6  /  5;
    printf ("x2 = %d\n", x);
    x = (7 + 6) % 5 / 2;
    printf ("x3 = %d\n", x);

}


/*
Ausgabe:

x0 = 11
x1 = 1
x2 = 0
x3 = 1

*/
```

```
//  Programm 2
#include <stdio.h>

#define PRINTX   printf ("%d\n", x)

int main () {

   int  x  =  2;
   int  y,  z;

   x *= 3 + 2;  PRINTX;

   x *= y = z = 4; PRINTX;

   x = y == z; PRINTX;

   x == (y = z); PRINTX;  /* in VS2010 : warning C4553:
                             '==': Operator hat keine
                             Auswirkungen; ist '='
                             beabsichtigt?  */

}


/*
Ausgabe:

10
40
1
1

*/
```

```
//  Programm 3
#include <stdio.h>

#define PRINT(int)   printf ("%d\n", int)

int main () {

    int x,  y,  z;

    x = 2;  y = 1;  z = 0;
    x = x && y || z;
    PRINT(x);
    PRINT(x || !y && z);

   x = y = 1;
   z =  x ++ - 1; PRINT (x);  PRINT (z);
   z +=  - x  ++  +  ++  y;  PRINT(x); PRINT (z);
   z =  x / ++ x;  PRINT (z);   //  Fragwuerdig

}

/*
Ausgabe:

1
1
2
0
3
0
1     undefiniert    //  Die Reihenfolge der Auswertung von
                     //  Teilausdrücken ist nicht spezifiziert.
*/
```

```
//  Programm 4
#include <stdio.h>

#define PRINT(int)   printf (#int " = %d\n", int)


int main () {

    int x,  y,  z;

    x = 03;  y = 02;  z = 01;

    PRINT (x | y & z);
    PRINT (x | y & ~ z);
    PRINT (x ^ y & ~ z);
    PRINT (x & y && z);

    x = 1;  y = - 1;
    PRINT (! x | x);
    PRINT (~ x | x);
    PRINT (x ^ x);

    x <<= 3;  PRINT (x);
    y <<= 3; PRINT (y);
    y >>= 3; PRINT (y);

}
```

/*
Ausgabe:

x | y & z = 3
x | y & ~ z = 3
x ^ y & ~ z = 1
x & y && z = 1
! x | x = 1
~ x | x = -1
x ^ x = 0
x = 8
y = -8
y = -1 // Ist der rechte Operand von a >> b
// negativ, dann ist das Ergebnis undefiniert,
// ist der nur linke Operand negativ, dann wird
// das Ergebnis durch die Implementation
// definiert.

***/**

```
//  Programm 5
#include <stdio.h>

#define PRINT(int)   printf (#int " = %d\n", int)


int main () {

   int  x  =  1,  y = 1,  z = 1;

   x  +=  y  +=  z;
   PRINT (x < y ? y : x);

   PRINT (x < y ? x ++ : y ++);
   PRINT (x);
   PRINT (y);

   PRINT (z += x < y ? x ++ : y ++);
   PRINT (y);
   PRINT (z);

   x = 3;  y = z = 4;
   PRINT ((z >= y >= x) ? 1 : 0);
   PRINT (z >= y && y >= x);

}
```

/*

Ausgabe:

x < y ? y : x = 3
x < y ? x ++ : y ++ = 2
x = 3
y = 3
z += x < y ? x ++ : y ++ = 4
y = 4
z = 4
(z >= y >= x) ? 1 : 0 = 0 **// Assoziativität des Operators**
// >= ist links nach rechts.
z >= y && y >= x = 1

***/**

```
//  Programm 6
#include <stdio.h>

#define PRINT3(x, y, z)  \
      printf (#x "=%d\t" #y "=%d\t" #z "=%d\n", x, y, z)


int main () {
   int x, y, z;

   x = y = z = 1;
   ++x || ++y && ++z;  PRINT3 (x, y, z);

   x = y = z = 1;
   ++x && ++y || ++z;  PRINT3 (x, y, z);

   x = y = z = 1;
   ++x && ++y && ++z;  PRINT3 (x, y, z);

   x = y = z = -1;
   ++x && ++y || ++z;  PRINT3 (x, y, z);

   x = y = z = -1;
   ++x || ++y && ++z;  PRINT3 (x, y, z);

   x = y = z = -1;
   ++x && ++y && ++z;  PRINT3 (x, y, z);

}
```

/*
Ausgabe:

x=2 y=1 z=1
x=2 y=2 z=1
x=2 y=2 z=2
x=0 y=-1 z=0
x=0 y=0 z=-1
x=0 y=-1 z=-1

***/**

```
//  Programm 7
#include <stdio.h>
#define PRINT(x, y)  \
        printf (#y " = %" #x "\n", y)

int integer = 5;
char character = '5';
char* string = "5";

int main () {
    PRINT (d, string);
    PRINT (d, character);
    PRINT (d, integer);
    PRINT (s, string);
    PRINT (c, character);
    PRINT (c, integer = 53);
    PRINT (d, ('5' > 5));

    {  int x = -2;
       unsigned ux = -2; /* warning C4245: "Initialisierung":
                            Konvertierung von "int" in
                            "unsigned int", signed/unsigned-
                            Konflikt. */

       PRINT (o, x);
       PRINT (o, ux);
       PRINT (d, x/2);
       PRINT (d, ux/2);
       PRINT (o, x >> 1);
       PRINT (o, ux >> 1);
       PRINT (d, x >> 1);
       PRINT (d, ux >> 1);
     }
}
```

/*
Ausgabe:

string = 4198960
character = 53
integer = 5
string = 5
character = 5
integer = 53 = 5
('5' > 5) = 1
x = 37777777776
ux = 37777777776
x/2 = -1
ux/2 = 2147483647
x >> 1 = 37777777777 // implementationsabhängig
ux >> 1 = 17777777777
x >> 1 = -1 // implementationsabhängig
ux >> 1 = 2147483647

***/**

```
//  Programm 8
#include <stdio.h>

#define PR(x)  printf (#x " = %.8g\t", (double) x)
#define NL putchar ('\n')
#define PR4(a, b, c, d)  PR(a); PR(b); PR(c); PR(d); NL

int main () {
   double d;
   float f;
   long l;
   int i;

   i = l = f = d = 100/3; PR4 (i, l, f, d);
   d = f = l = i = 100/3; PR4 (i, l, f, d);
   i = l = f = d = 100/3.; PR4 (i, l, f, d);
   d = f = l = d = (float)100/3; PR4 (i, l, f, d);
   i = l = f = d = (double) (100000/3); PR4 (i, l, f, d);
   d = f = l = i = 100000/3; PR4 (i, l, f, d);

/*  Viele Warnungen über möglichen Datenverlust.
*/
}

/*
Ausgabe:
i = 33  l = 33  f = 33  d = 33
i = 33  l = 33  f = 33  d = 33
i = 33  l = 33  f = 33.333332   d = 33.333333
i = 33  l = 33  f = 33  d = 33
i = 33333       l = 33333       f = 33333       d = 33333
i = 33333       l = 33333       f = 33333       d = 33333
*/
```

```
//  Programm 9
#include <stdio.h>

#define PR(x)  printf (#x " = %g\t", (double) (x))
#define NL putchar ('\n')
#define PR1(a)  PR(a); NL
#define PR2(a,b)  PR(a); PR1(b)

int main () {

   double d =3.2,  x;
   int i = 2, y;

   x = (y=d/i) * 2;  PR2 (x, y);
   y = (x = d/i) * 2; PR2 (x, y);

   y = d * (x = 2.5/d); PR1 (y);
   x = d * (y = ((int)2.9 + 1.1) / d);  PR2 (x, y);

/*  Mehrere Warnungen über möglichen Datenverlust.
*/
}



/*
Ausgabe:

x = 2       y = 1
x = 1.6     y = 3
y = 2
x = 0        y = 0
*/
```

Operatoren in C

Präzedenz	Name	Assoziativität
1	**Postfix-Operatoren:** **[] () . -> ++ --** **(type name) {list}**	**links nach rechts**
2	**Unäre Operatoren:** **++ --** **! ~ + - * & sizeof**	**rechts nach links**
3	**cast: (type name)**	**rechts nach links**
4	*** / %**	**links nach rechts**
5	**+ -**	**links nach rechts**
6	**<< >>**	**links nach rechts**
7	**< <= > >=**	**links nach rechts**
8	**== !=**	**links nach rechts**
9	**&**	**links nach rechts**
10	**^**	**links nach rechts**
11	**|**	**links nach rechts**
12	**&&**	**links nach rechts**
13	**||**	**links nach rechts**
14	**? :**	**rechts nach links**
15	**= += -= *= /=** **%= &= ^= |=** **<<= >>=**	**rechts nach links**
16	**Komma-Operator: ,**	**links nach rechts**

Bemerkung: C++ kennt weitere Operatoren.

Sequence Points in C

Bemerkung: **C ist eine Sprache mit Sequentialsemantik. Eine Anweisung wird mit einem Semikolon beendet. Vor dem Übergang zur nächsten Anweisung müssen alle Seiteneffekte der Anweisung ausgeführt sein. Daher haben Semikola die Bedeutung von "sequence points".**

Beispiel:

```
. . . ;  x = 5 + y;   z = f (a, b);   a = i++;  . . .
```

Bemerkung: **Die Auswertungsreihenfolge für Ausdrücke wird im wesentlichen durch die syntaktische Struktur festgelegt. Da Ausdrücke Teilausdrücke mit Seiteneffekten enthalten können, werden einige zusätzliche Regeln eingeführt.**

Beispiel:

```
int   a, b;
/*          */
a  =  a + 32760 + b + 5;        //  (1)
```

Der Ausdruck (1) ist zu berechnen, als ob der Klammerausdruck

a = (((a + 32760) + b) +5);

geschrieben wurde, es sei denn, man kann nachweisen, daß eine Umformung des Ausdrucks das gleiche Resultat erzeugt.

Wegen möglicher Rundungsfehler sind die folgenden Ausdrücke verschieden.

```
double   x, y, z;

x  =  (x * y) * z;     verschieden von    x  *=  y * z;
z  =  (x – y) + y;     verschieden von    z  =  x;
z  =  x + x * y;       verschieden von    y  =  x * (1.0 + y)
```

C schreibt nicht die Auswertungsreihenfolge für Teilausdrücke vor.

Beispiel:

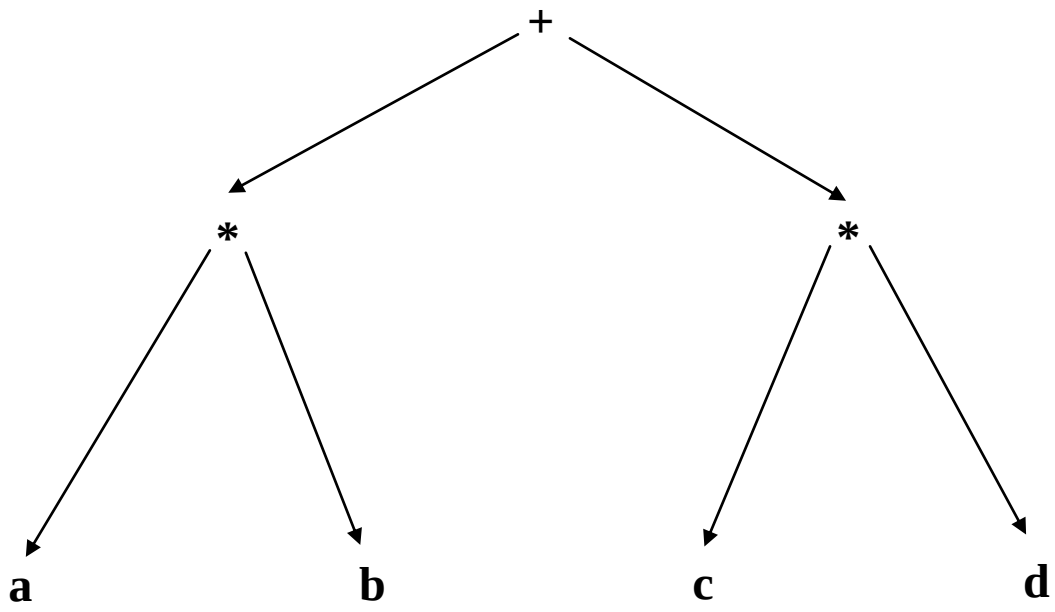
```
sum  =  sum * 10 – '0' + (*p++ = getchar ());
```

In diesem Fall muß nur gewährleistet sein, daß die Erhöhung von p vor der "Verarbeitung" des Semikolons und der Aufruf von getchar () vor der Nutzung seines Ergebnisses erfolgt sind.

Zur Auswertung von Ausdrücken:

Beispiel:

Der Ausdruck a * b + c * d besteht aus zwei Teilausdrücken a * b und c * d; die Reihenfolge, in der diese beiden Teilausdrücke ausgewertet werden, ist nicht festgelegt. Ebenso ist nicht festgelegt, in welcher Reihenfolge die Werte von a, b, c und d zu bestimmen sind. Die Reihenfolgebeschränkungen lassen sich am zugehörigen Operatorbaum ablesen.



Für manche Operatoren ist die Reihenfolge der Auswertung der Operanden festgelegt; es sind dies: die boolschen Operatoren || und &&, der Auswahloperator ? : und der Komma-Operator ,.

Diese Operatoren verfügen auch über explizite Sequence Points, jeweils nach der Auswertung des linken Operanden ist ein Sequence Point gegeben. Die Wohldefiniertheit eines Ausdrucks wird auch bestimmt durch die folgende Regel des Standards.

Between the previous and next sequence point an object shall have its stored value modified at most once by the evaluation of an expression. Furthermore, the prior value shall be read only to determine the value to be stored.

Beispiele:

```
int  x;
x  =  ++x;
```
Dieser Ausdruck ist undefiniert.

```
int  x;
x  +=  x  *  x;
```
Dieser Ausdruck ist wohldefiniert.

```
int  x,  y;
x++  *  y++  ?  x--  :  y--;
```
Dieser Ausdruck ist wohldefiniert.

```
a  =  0;
b  =  (a = 1,  2*a)  +  3*a;
```
Dieser Ausdruck ist undefiniert. Die Erstauswertung des Teilausdrucks 3*a führt zu Konflikten.

Ein Sequence Point gilt als gesetzt

nach dem ersten Operanden eines logischen and (&&),

nach dem ersten Operanden eines logischen or (||),

nach dem ersten Operanden des Auswahl-operators (? :),

unmittelbar nach einem Komma-Operator (,),

nach Auswertung des vollständigen Kontrollausdrucks in einer if-, switch-, while-, do-Anweisung,

nach jedem der drei Ausdrücke, sofern vorhanden, in einer for-Anweisung,

nach einer normalen Ausdrucksanweisung, abgeschlossen mittels Semikolon,

nach dem Ende einer vollständigen Deklaration,

zwischen der Bestimmung eines Funktionsdesignators und der Bestimmung der Funktionsargumente,

unmittelbar nach Aufruf einer Funktion,

nach Bestimmung eines Ausdrucks einer return-Anweisung.