FernUniversität in Hagen

```
#include <stdio.h>

void main(void)
{
  int c;

  while ((c = getchar()) !=  EOF)
            putchar(c);
}
```

Programmieren in C

Kurseinheit 1:

Einführung i
C
I
E
O
An

Autor:
Dr. Andreas Bortfeldt

I. Inhaltsübersicht

Kurseinheit 1: Programmieren in C

Kurseinheit 2: Programmieren in C

Kurseinheit 3: Programmieren in C

II. Leseprobe

Auszug aus Kurseinheit 1, Kapitel 1

1 Einführung in die C-Programmierung

Zur Einführung werden zunächst einige Grundbegriffe der Programmierung und der Programmentwicklung knapp erläutert. Im zweiten Abschnitt des Kapitels werden einige einfache C-Programme besprochen, um anhand von Beispielen einen Eindruck und einen ersten Überblick über die Sprache zu gewinnen.

1.1 Algorithmen und Programme

Algorithmus

Prozeß, Prozessor

Der Begriff des **Algorithmus** ist von zentraler Bedeutung für die Programmierung. Ein Algorithmus beschreibt ein Verfahren zur Lösung einer bestimmte Aufgabe. Er enthält eine Folge von Anweisungen, deren schrittweise korrekte Ausführung zur Lösung der gestellten Aufgabe führt. Die Abarbeitung eines Algorithmus wird als **Prozeß**, die ausführende Einheit allgemein als **Prozessor** bezeichnet.

Beispiele für Algorithmen und ihre Ausführung aus dem Alltag stellen etwa die Zubereitung einer Mahlzeit nach einem bestimmten Rezept oder das Spielen eines Musikstücks nach einem Notenblatt dar.

Um einen Algorithmus abarbeiten zu können, muß der Prozessor die Bedeutung jeder einzelnen Anweisung verstehen und in eine von ihm ausführbare Operation umsetzen können. Das Verstehen oder Interpretieren einzelner Anweisungen des

Algorithmus, der in einer bestimmten Sprache formuliert ist, besitzt verschiedene Aspekte.

Syntax

Die **Syntax** einer Sprache legt fest, welche elementaren Symbole oder Worte die Sprache umfaßt und welche formalen grammatikalischen Regeln bei der "Kombination" der Symbole zu umfassenderen sprachlichen Ausdrücken wie etwa den Sätzen einer natürlichen Sprache einzuhalten sind.

Werden Operationen unter Verwendung von nicht zur Sprache gehörenden Symbolen formuliert, kann der Prozessor die Operation nicht interpretieren. Aber auch bei ausschließlicher Verwendung bekannter Symbole können Operationen syntaktisch falsch formuliert werden. Im Kontext der Arithmetik erscheinen in der Anweisung

8 + = 7

nur bekannte Symbole, jedoch ist der Ausdruck syntaktisch fehlerhaft, weil die Symbole + und = nicht unmittelbar aufeinander folgen dürfen.

Semantik

Die **Semantik** einer Sprache umfaßt die Bedeutungen der Symbole und der aus diesen gebildeten Ausdrücke. Syntaktisch korrekte Ausdrücke können inhaltslos oder inhaltlich falsch sein:

"Drei mal drei ist Donnerstag."

"Nenne den achten Tag der Woche."

Bezogen auf die Ausführung eines Algorithmus verlangt die semantische Korrektheit, daß der Prozessor jeder im Algorithmus formulierten Anweisung eine gewisse von ihm ausführbare Operation eindeutig zuordnen kann.

Insgesamt gilt, daß nur ein in der Sprache eines Prozessors syntaktisch und semantisch korrekt formulierter Algorithmus erfolgreich ausgeführt werden kann.

Ist als Prozessor ein Computer vorgesehen, muß ein Algorithmus in einer **Programmiersprache**, d.h. in einer dem Computer verständlichen Sprache dargestellt werden. Ein **Programm** ist ein in einer Programmiersprache formulierter Algorithmus.

Maschinensprache

Computer können unmittelbar nur in einer **Maschinensprache** formulierte Anweisungen interpretieren und ausführen, die binär als Folgen von Nullen und Einsen zu formulieren, daher für den Programmierer schwer verständlich und vor allem sehr elementar sind.

höhere Programmiersprachen

Sogenannte problemorientierte oder **höhere Programmiersprachen** wie Fortran, Cobol, Pascal oder C erlauben eine wesentlich effizientere, auf das Anwendungsgebiet wie auch auf die Bedürfnisse des Programmierers zugeschnittene Programmerstellung mit vergleichsweise mächtigen Anweisungen.

Programmerstellung

Den Prozeß der Erstellung eines von einem Computer ausführbaren (lauffähigen) Programms unter Verwendung einer höheren Programmiersprache zeigt die Abb. 1.1. Diese sei wie folgt erläutert:

- Zu Beginn der Programmerstellung liege der Algorithmus in einer natürlichen Sprache oder in semiformaler Notation vor. Verbreitete Formen der Notation sind etwa Programmablaufpläne, Pseudocode-Darstellungen oder Struktogramme.

- In einem ersten Schritt wird der Algorithmus in der gewählten höheren Programmiersprache formuliert. Dieser Schritt wird als **Programmierung** oder auch Codierung bezeichnet. Das Ergebnis ist der **Quelltext** oder **Quellcode** des Programms.

Programmierung, Codierung, Quelltext, Quellcode

- Im zweiten Schritt wird das Quellprogramm durch einen sogenannten Compiler in die Maschinensprache eines Computers übersetzt. Die Übersetzung wird als Compilierung bezeichnet. Der Compiler einer höheren Programmiersprache ist selbst ein Programm, welches als Eingabe das Quellprogramm erhält und als Ausgabe den **Maschinencode** des Algorithmus, auch als **Objektcode** bezeichnet, liefert.

Compiler, Maschinencode, Objektcode

- Das nun lauffähige Programm kann danach von dem Computer ausgeführt werden.

Auf die systematische Entwicklung von Programmen, darunter auch auf Darstellungstechniken für Algorithmen, wird in dem Kap. 6.9 sowie im Kap. 8.4 der zweiten Kurseinheit noch näher eingegangen.

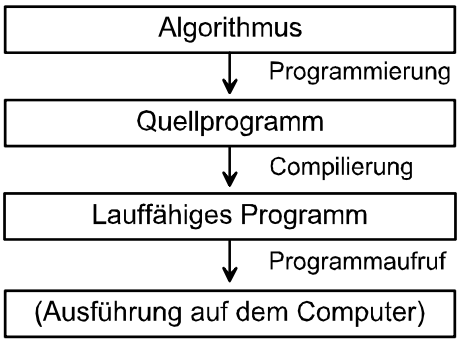

Abb. 1.1. Vom Algorithmus zur Ausführung eines lauffähigen Programms.

Fehler

Auf den einzelnen Stufen der Programmerstellung können verschiedene **Fehler** auftreten. Der Compiler entdeckt **syntaktische Fehler**, d.h. Verstöße gegen die Syntax einer höheren Programmiersprache. **Semantische Fehler** wie z.B. die Aufforderung, den Namen des achten Wochentages auszugeben oder eine Division durch Null auszuführen, werden meist erst bei der Ausführung eines Programms festgestellt. Semantische Fehler können sich auf verschiedene Weise äußern. Kommt es aufgrund eines semantischen Fehlers wie etwa einer Division durch Null zu einem Programmabbruch, wird von einem **Laufzeitfehler** gesprochen. Andere semantische Fehler haben zur Folge, daß ein Programm nicht terminiert, d.h. seine Ausführung von selbst beendet. Schließlich kann der Fall eintreten, daß ein Programm zwar normal abläuft, jedoch falsche Ergebnisse erzeugt.

Der folgende Abschnitt führt überblicksartig in die Programmiersprache C ein. Hierbei werden die oben angesprochenen Schritte (vgl. Abb. 1.1) bei der Erstellung eines ausführbaren (C-)Programms noch eingehender dargestellt.

1.2 Ein kurzer Überblick über C

C-Programmierumgebung

Editor, Quelldateien

Um einen Überblick über die Sprache C zu gewinnen, werden im folgenden vier Beispielprogramme besprochen. Dabei geht es vornehmlich darum, einen ersten Eindruck von der Sprache zu gewinnen. Alle hier vorgestellten Sprachelemente werden später noch ausführlich besprochen.

Unter einer **C-Programmierumgebung** (oder C-Entwicklungsumgebung) wird eine Gesamtheit von Programmen verstanden, die zur Erstellung eines ausführbaren C-Programms genutzt werden. Neben dem Compiler, dessen Funktion bereits erläutert wurde, umfaßt eine C-Programmierumgebung einen **Editor**. Mit diesem wird der Quelltext eines C-Programms erzeugt und in einer oder mehreren Quelldateien abgespeichert. Weitere Komponenten einer C-Programmierumgebung werden bei der Behandlung einzelner Beispielprogramme eingeführt. Am Abschnittsende werden die Bestandteile einer C-Programmierumgebung sowie die Schritte der Erstellung eines ausführbaren C-Programms zusammenfassend dargestellt.

Kommentare

Beispielprogramm 1

Das Programm beginnt mit einem **Kommentar**, der mit den Zeichen /* eingeleitet und mit */ beendet wird. Kommentare enthalten lediglich Informationen für den menschlichen Programmleser und haben keine Auswirkungen auf die Programmausführung. Sie können sich auch über mehrere Zeilen erstrecken.

```
/* Beispiel B01-01, das erste C-Programm */

#include <stdio.h>            /* EA-Funktionen bereitstellen */

void main(void)               /* main-Funktion beginnt */
{
    int i;                    /* Variablendefinition */
                              /* ab hier Anweisungen */
    i = 1;                    /* Variable i erhält Wert 1 */

                              /* Ausgabe mit Funktion printf */
    printf("\nDies ist C-Programm Nr. %d.", i);

    i = i + 1;                /* Addition und erneute Zuweisung */

    printf("\nDas %d-te C-Programm folgt.", i);
}                             /* Ende main */
```

Präprozessor

Auf den ersten Kommentar folgt eine Programmzeile, die mit #include beginnt. Dabei handelt es sich um eine Präprozessoranweisung. Der **Präprozessor** ist ein weiteres zu einer C-Programmierumgebung gehörendes Programm, der einen Quelltext für den Compiler vorbereitet. Die Wirkung der Anweisung #include wird später erläutert.

Funktionen, main

Im Programm folgt nun eine sogenannte **Funktion** mit dem Namen **main**. Die Funktionen eines C-Programms stellen Unterprogramme dar, mit denen die Aufgabe eines Programms arbeitsteilig erledigt wird. Hierzu können sich Funktionen gegenseitig aufrufen. Wird eine Funktion aufgerufen, verrichtet sie eine bestimmte Tätigkeit und gibt danach die Kontrolle wieder an die aufrufende Funktion ab.

Jedes C-Programm muß genau eine main-Funktion enthalten, die bei Ausführungsbeginn des Programms vom Betriebssystem aufgerufen wird.

Eine Funktion beginnt mit einem Funktionskopf, der ihren Namen – hier main – enthält. Diesem folgt ein Funktionsrumpf, der mit einer öffnenden geschweiften Klammer { beginnt und mit einer schließenden geschweiften Klammer } endet.

Im Funktionsrumpf von main wird durch

```
int i;
```

zunächst die **Variable** i definiert. Hierdurch wird dem Namen i ein bestimmter Speicherplatz zugeordnet. Auf diesen wird stets zugegriffen, wenn der Name i in einer Anweisung erscheint. int ist eine feste Bezeichnung (Schlüsselwort) für einen von der Sprache bereitgestellten **Datentyp** und besagt, daß die Variable i bzw. der zugeordnete Speicherplatz beliebige (positive und negative) ganze Zahlen eines gewissen Intervalls enthalten kann, aber z.B. nicht die reelle Zahl 1.5.

Variablen, Datentypen

Die Definition der Variablen i stellt eine passive **Vereinbarung** dar, der aktive Anweisungen folgen, die während der Programmabarbeitung ausgeführt werden.

Vereinbarung

Die erste **Anweisung**

Anweisung

```
i = 1;
```

bewirkt, daß die Variable i den Wert 1 erhält, d.h. auf dem Speicherplatz von i wird die Zahl 1 eingetragen. Man beachte, daß das Zeichen = eine Zuweisung eines Wertes bewirkt und nicht etwa eine wertmäßige Gleichheit beider Seiten behauptet oder testet. Anweisungen wie auch Vereinbarungen werden in C durch ein Semikolon abgeschlossen. Das Datenobjekt 1 ist eine **Konstante** mit dem Wert 1.

Konstanten

Mit der nächsten Anweisung wird die Funktion **printf** aufgerufen, die folgende Ausgabe am Bildschirm bewirkt:

printf

```
Dies ist C-Programm Nr. 1.
```

Der Funktionsaufruf von printf enthält neben dem Funktionsnamen eine Zeichenkette in Anführungszeichen ("...") und nach einem Komma den Namen der auszugebenden Variablen i.

Der Text in der Zeichenkette wird unverändert ausgegeben, bis auf die Zeichen \n und die Formatspezifikation %d. \n bewirkt einen Zeilenvorschub, d.h. die nächste Ausgabe beginnt am Anfang der nächsten (Bildschirm-)Zeile. %d veranlaßt die Ausgabe des aktuellen Wertes der Variablen i vom Typ int als ganze Zahl in dezimaler Schreibweise.

Die Funktion printf konnte aufgerufen werden, ohne daß diese Funktion im Programm mit ihrem Quelltext erscheint.

Viele Fähigkeiten von C, insbesondere die Mittel für die Ein- und Ausgabe von Daten, sind nicht in der Sprache selbst, sondern wie printf als Funktionen in der sogenannten **C-Standardbibliothek** implementiert. Die Standardbibliothek ist in einzelne Teilbibliotheken gegliedert, denen jeweils eine sogenannte **Definitionsdatei** zugeordnet ist. Die Standardbibliothek und die zugehörigen Definitionsdateien stellen weitere Komponenten einer C-Programmierumgebung dar. Während die Standardbibliothek meist als Objektcode, d.h. in übersetzter Form vorliegt,

C-Standardbibliothek

Definitionsdatei

werden die zugehörigen Definitionsdateien als Quelldateien, also in lesbarer Form bereitgestellt.

Funktionsdatei

Definitionsdateien sind erkennbar an der Dateinamenserweiterung .h und enthalten keine Funktionen, sondern lediglich Vereinbarungen, d.h. passive Bestandteile von Programmen. Quelldateien mit Funktionen, hier als **Funktionsdateien** bezeichnet, besitzen hingegen die Erweiterung .c. Der Teilbibliothek für die Ein- und Ausgabe ist die Definitionsdatei stdio.h zugeordnet. Mit der Anweisung

```
#include <stdio.h>
```

am Programmanfang wird die Datei stdio.h in das Programm vor der Compilierung eingefügt. Anhand der in der Definitionsdatei stdio.h enthaltenen und mittels #include eingefügten Vereinbarungen kann der Compiler bei der Übersetzung prüfen, ob printf oder andere Funktionen für die Ein- und Ausgabe korrekt aufgerufen wurden.

Mit der nächsten Anweisung in main

```
i = i + 1;
```

lesen und schreiben

wird zunächst der aktuelle Wert von i, nämlich 1, sowie der Wert der Konstanten 1 gelesen. Dann werden beide Werte gemäß dem Operator + addiert. Das Resultat 2 wird dann der Variablen wieder zugewiesen, d.h. auf den Speicherplatz von i geschrieben.

Auf Variablen kann also lesend und schreibend zugegriffen werden. Dagegen wäre eine Anweisung

```
1 = i + 1;          /*Achtung: falsch!*/
```

Konstanten

syntaktisch falsch und würde vom Compiler beanstandet. **Konstanten** können nur gelesen, aber nicht verändert werden.

Datenobjekte rechts von einem Zuweisungsoperator = werden gelesen, Objekte links werden mit dem für die rechte Seite ermittelten Wert überschrieben. Rechts dürfen also Variablen und Konstanten, links nur Variablen stehen.

Der letzte printf-Aufruf erzeugt entsprechend dem neuen Inhalt von i die Ausgabe:

```
Das 2-te C-Programm folgt.
```

Beispielprogramm 2

```
/* Beispiel B01-02, Berechnung des Minimums zweier Zahlen */

#include <stdio.h>      /* EA-Funktionen bereitstellen */

void main(void)
{
  float a, b, min;

  /* Eingabe */
  printf("\nProgramm bestimmt das Minimum zweier reeller Zahlen");
  printf("\nBitte jede Eingabe mit RETURN beenden.");
  printf("\nBitte reelle Zahlen mit Dezimalpunkt (z.B. 5.9)"
```

```
        " eingeben. \n");
  printf("\nErste Zahl: ");
  scanf("%f",&a);
  printf("Zweite Zahl: ");
  scanf("%f",&b);

  /* Berechnung */
  if (a < b)            /* Test der Bedingung a < b */
      min = a;          /* wenn erfuellt: Anweisung-1 */
  else                  /* andernfalls: */
      min = b;          /* Anweisung-2 */

  /* Ausgabe */
  printf("Minimum von%f und %f lautet %f", a, b, min);
}
```

Das zweite Programm bestimmt das Minimum zweier reeller Zahlen. Erneut ist nur eine main-Funktion vorhanden, deren Anweisungen sich in drei Teile gliedern. Nach einer Eingabe zweier Zahlen folgen die Minimumberechnung und die Ausgabe des Resultats.

Alle printf-Aufrufe im Programmabschnitt Eingabe dienen der Benutzerführung und geben nur die in der Zeichenkette enthaltenen Informationen aus. Bei dem dritten Aufruf treten zwei Zeichenketten nacheinander auf. Beide Zeichenketten werden zu einer einzigen verkettet, die ausgegeben wird.

scanf

Neben printf wird die Bibliotheksfunktion **scanf** für die formatgesteuerte Eingabe von der Tastatur benutzt. Die Aufrufe von scanf sehen ähnlich wie die von printf aus. In einer Zeichenkette werden Formatspezifikationen zur Steuerung der Eingabe angegeben. Hier wird bei dem scanf-Aufruf:

```
scanf("%f", &a);
```

mit der Formatspezifikation %f festgelegt, daß eine reelle Zahl mit Dezimalpunkt von der Tastatur erwartet wird. Dies entspricht der Definition der Variablen a mit dem Datentyp float, der eine Teilmenge der reellen Zahlen umfaßt.

Das Zeichen & vor einem Variablennamen bezeichnet in C den Adressoperator, der zu einer Variablen ihre Speicheradresse liefert. Variablen müssen in scanf-Aufrufen meist mit vorangestelltem Adressoperator angegeben werden.

Nach einem scanf-Aufruf hält die Verarbeitung an und wartet auf eine Benutzereingabe. Nach einer korrekten Eingabe, die mit der RETURN-Taste abzuschließen ist, enthält die Variable a nun eine reelle Zahl.

if-else-Anweisung

Im zweiten Teil von main wird das Minimum der Werte der Variablen a und b berechnet. Hierzu wird eine **if-else-Anweisung** genutzt. Der Vergleichsoperator < besitzt die aus der Arithmetik bekannte Bedeutung. Die if-else-Anweisung prüft also zuerst, ob a kleiner als b ist. Falls diese Bedingung erfüllt ist, wird die Zuweisung

```
min = a;
```

andernfalls die Zuweisung

```
min = b;
```

ausgeführt. Es wird also abhängig von einer Bedingung nur eine von zwei Anweisungen ausgewählt und damit gewährleistet, daß die Variable min den kleineren (bzw. nicht größeren) der beiden Werte von a und b erhält.

Bei der abschließenden Ausgabe mit printf werden mit einem Aufruf die Werte aller drei Variablen a, b und min ausgegeben. Hierbei werden die drei Formatspezifikationen %f, die eine Ausgabe als reelle Zahl veranlassen, nacheinander den drei Variablenwerten a, b und min zugeordnet.

Beispielprogramm 3

Wenden wir uns dem **dritten Beispielprogramm** zu. Auch dieses enthält nur eine main-Funktion, deren Anweisungen sich in die Teile Eingabe und Berechnung sowie Ausgabe gliedern lassen. Berechnet werden soll diesmal das Maximum von 5 positiven reellen Zahlen.

while-Anweisung

Sollen mehrere Zahlen in analoger Weise verarbeitet werden, so eignen sich hierfür Schleifenanweisungen. In diesem Programm wird eine **while-Anweisung**, die auch als while-Schleife bezeichnet wird, verwendet.

Nach einigen Bildschirmausgaben werden die Zählvariable i und die Variable max für das zu berechnende Maximum mit Nullwerten initialisiert. Die folgende while-Schleife wird fünfmal durchlaufen. Die Anzahl der Durchläufe wird anhand der Zählvariablen i kontrolliert.

```
/* Beispiel B01-03, Maximum von 5 positiven reellen Zahlen */

#include <stdio.h>           /* EA-Funktionen bereitstellen */

void main(void)
{
  int i;                     /* Zaehler fuer reelle Zahlen*/
  float max;                 /* Maximum reeller Zahlen*/
  float zahl;                /* eine reelle Zahl */

  /* Eingabe und Verarbeitung */
  printf("\nDas Programm bestimmt das Maximum von 5 positiven "
         " reellen Zahlen.\n\n");
  printf("\nBitte jede Eingabe mit Return beenden und");
  printf("\npositive reelle Zahl mit Dezimalpunkt eingeben.\n\n");

  i = 0;                     /* Zaehler initialisieren */
  max = 0.0;                 /* Maximum initialisieren */

  /* es folgt Schleife: */
  while (i < 5)              /* Test: i noch kleiner als 5? */
  {                          /* wenn ja, dann erneut: */
    printf("%d-te Zahl: ", i+1);
    scanf("%f", &zahl);      /* einlesen naechste Zahl */
    if (zahl > max)          /* Vergleich und eventuell */
        max = zahl;          /* Aktualisierung Maximum */
```

```
    i = i + 1;                /* Zaehler aktualisieren */
  }                           /* Ende der Schleife */

  /* Ausgabe */
  printf("\nMaximale Zahl lautet: %f", max);
}
```

Ein erneuter Schleifendurchlauf findet nur statt, wenn die Bedingung am Schleifenanfang

$i < 5$

noch erfüllt ist. Vor dem ersten Durchlauf besitzt i den Wert 0. Am Ende jedes Durchlaufs wird die Zählvariable jeweils um 1 erhöht:

$i = i + 1;$

Nach dem fünften Durchlauf besitzt i daher den Wert 5, folglich wird die Schleife kein weiteres Mal durchlaufen. Man beachte, daß die Verarbeitung einer while-Schleife vom Schleifenende automatisch zum Schleifenanfang zurückspringt.

Im Inneren der Schleife wird jeweils eine weitere Zahl in die Variable zahl mit der scanf-Funktion eingelesen. Diese wird mit dem bisherigen Wert des Maximums in der Variablen max verglichen. Gegebenenfalls wird der Wert von max aktualisiert. Das Vorgehen entspricht dem aus dem Beispielprogramm 2 bekannten Vergleich zweier Zahlen und wird wieder mit einer if-else-Anweisung realisiert. Es zeigt sich, daß eine while-Schleife und eine if-else-Anweisung geschachtelt werden können. Der alternative else-Zweig ist bei einer if-else-Anweisung optional und fehlt hier. Ist im vorliegenden Fall die Bedingung

$zahl > max$

nicht erfüllt, wird die Verarbeitung einfach bei der nächsten Anweisung fortgesetzt.

Neben der if-else-Auswahlanweisung und der while-Schleife stellt C weitere Auswahl- und Wiederholungsanweisungen zur Verfügung.

Felder

Im Programm werden die zu vergleichenden Zahlen alle in der Variablen zahl gehalten. Sie existieren daher nur nacheinander, nicht gleichzeitig. Daher mußten im Beispielprogramm auch die Eingabe und die Verarbeitung der Zahlen verzahnt durchgeführt werden. Sollen alle Zahlenwerte zugleich vorgehalten werden, bietet sich hierfür ein sogenanntes **Feld** an.

zusammengesetzte Datenobjekte

Felder stellen **zusammengesetzte Datenobjekte** dar, die mehrere elementare Datenobjekte (Variablen) umfassen. Charakteristisch für Felder ist, daß die elementaren Objekte alle denselben Datentyp besitzen. So hätte für die Bildung des Maximums ein Feld aus 5 Variablen des Typs float benutzt werden können.

Strukturen

Daneben gibt es sogenannte **Strukturen**, deren elementare Komponenten nicht denselben Typ besitzen müssen. So können etwa verschiedenartige Personendaten wie Name (Zeichenkette) und Alter (ganze Zahl) in einer Struktur vereinigt werden.

Beispielprogramm 4

mehrere Funktionen...

Abschließend sei ein **viertes Beispielprogramm** vorgestellt. Es bestimmt wie das zweite Programm das Minimum zweier reeller Zahlen. Im Unterschied zu Beispiel 2 enthält das Programm neben main eine weitere Funktion. Diese wird von main nach dem Einlesen der reellen Werte aufgerufen und berechnet deren Minimum.

...und ihre Kommunikation über Parameter

Die Arbeitsteilung beider Funktionen setzt ihre Kommunikation voraus. Diese erfolgt hier über Parameter. Der Kopf der Funktion berechne_minimum enthält neben dem Funktionsnamen eine **Parameterschnittstelle**, die die Parameter x und y des Typs float umfaßt. Bei dem Aufruf der Funktion übergibt main die vorher eingelesenen Zahlen über diese Parameter. Sie stehen danach der Funktion berechne_minimum für die Berechnung zur Verfügung.

Rückgabewert

Umgekehrt liefert die Funktion berechne_minimum abschließend das ermittelte Minimum über den Funktionsnamen mit der return-Anweisung an main zurück. Dort wird das Minimum der Variablen min zugewiesen und ausgegeben. Der Typ float des **Rückgabewertes** steht im Funktionskopf vor dem Funktionsnamen berechne_minimum.

void

Die Funktion main des vierten Beispielprogramms erhält weder Parameter noch gibt sie einen Wert (mit return) zurück. Dies wird durch eine leere Parameterliste (void) sowie das Wort void vor dem Namen main angegeben.

Prototyp

Im Programm steht vor beiden Funktionen ein **Prototyp** der Funktion berechne_minimum, der dem Kopf dieser Funktion entspricht. Die Angabe des Prototyps vor der main-Funktion sorgt dafür, daß die Funktion berechne_minimum innerhalb von main aufgerufen werden kann.

```
/* Beispiel B01-04, Berechnung des Minimums zweier Zahlen */

/* -------------------- Includes ----------------------------- */
#include <stdio.h>      /* EA-Funktionen bereitstellen */

/* -------------------- Prototypen --------------------------- */
/* Funktionen deklarieren */
float berechne_minimum(float x, float y);

/* -------------------- Funktion main ------------------------ */
void main(void)
{
  float a, b, min;

  /* Eingabe – wie in Beispiel 2 */
  printf("\nProgramm bestimmt das Minimum zweier reeller Zahlen");
  printf("\nBitte jede Eingabe mit RETURN beenden.");
  printf("\nBitte reelle Zahlen mit Dezimalpunkt (z.B. 5.9)"
         " eingeben. \n");
  printf("\nErste Zahl: ");
  scanf("%f",&a);
  printf("Zweite Zahl: ");
  scanf("%f",&b);
```

```
  /* Berechnung, hier durch Aufruf einer Funktion!*/
  min = berechne_minimum(a,b);

  /* Ausgabe – wie in Beispiel 2 */
  printf("Minimum von%f und %f lautet %f", a, b, min);
}                /* Ende main */

/* -------------------- Funktion berechne_minimum ------------- */
/* Funktion ermittelt Minimum zweier uebergebener reller Zahlen */
float berechne_minimum(float x, float y)
{
  float minxy;
  if (x < y)      /* Minimumberechnung wie in Beispiel 2, aber: */
      minxy = x;  /* - x und y werden als Parameter uebergeben */
  else
      minxy = y;
  return minxy;   /* - Rueckgabe des berechneten Wertes */
}                 /* Ende Funktion */
```

globale Variablen

Neben der hier realisierten Kommunikation über Parameter können mehrere Funktionen auch über **globale Variablen** Werte austauschen. Außerhalb von Funktionen definierte Variablen werden globale Variablen genannt, während die innerhalb einer Funktion definierten Variablen als lokal bezeichnet werden. Alle Funktionen, die im Programmtext auf globale Variablen folgen, können auf diese lesend und schreibend zugreifen. Ein Zugriff auf lokale Variablen ist hingegen nur in der Funktion möglich, wo die Variablen definiert wurden. Im gegebenen Fall hätten die Variablen a und b für die zu vergleichenden Zahlen global definiert werden können, um die Kommunikation zwischen den beiden Programmfunktionen abzuwickeln.

mehrere Quelldateien, Modularisierung

Bei allen gezeigten Beispielen wurde das gesamte Quellprogramm in einer Datei verwaltet. C-Programme lassen sich jedoch auch auf **mehrere Quelldateien** (*.c) verteilen, die jeweils eine oder mehrere Funktionen enthalten und getrennt compiliert werden. Die sinnvolle Gliederung von C-Programmen in mehrere Quelldateien wird als **Modularisierung** bezeichnet.

Objektdateien *.obj,

Linker,

ausführbare Dateien *.exe

Das lauffähige Programm muß hierbei aus den compilierten Dateien zusammengefügt werden, die an der Dateinamenserweiterung .obj erkennbar sind und als Objektdateien bezeichnet werden. Dies übernimmt ein weiteres Programm der C-Programmierumgebung, ein sogenannter **Linker** (deutsch: Binder). Aber auch bei Programmen, die nur eine Quelldatei umfassen, wird der Linker benötigt. Bei jeder Erstellung eines ausführbaren Programms erweitert der Linker den Objektcode des Programms um benutzte Teile der als Objektcode vorliegenden Standardbibliothek. Vom Linker erzeugte lauffähige Programme werden schließlich in ausführbaren Dateien (Namenserweiterung .exe) abgelegt.

Abschließend seien die eingeführten Komponenten einer C-Programmierumgebung nochmals aufgezählt. Diese umfaßt (mindestens) die folgenden Programme bzw. Werkzeuge:

Komponenten einer C-Programmierumgebung

(1) einen Editor,
(2) einen Präprozessor,
(3) einen C-Compiler,
(4) die C-Standardbibliothek einschließlich zugehöriger Definitionsdateien,
(5) einen Linker.

Erstellung eines ausführbaren Programms

Die Abb. 1.2 veranschaulicht zusammenfassend die Schritte bei der Erstellung eines ausführbaren C-Programms, wobei hier von einem Programm mit nur einer Quelldatei ausgegangen wird. Angemerkt sei, daß die Bearbeitung des Quellcodes eines C-Programms durch den Präprozessor und den Compiler meist zusammenhängend erfolgt und daher hier in einem einzigen Schritt dargestellt wird.

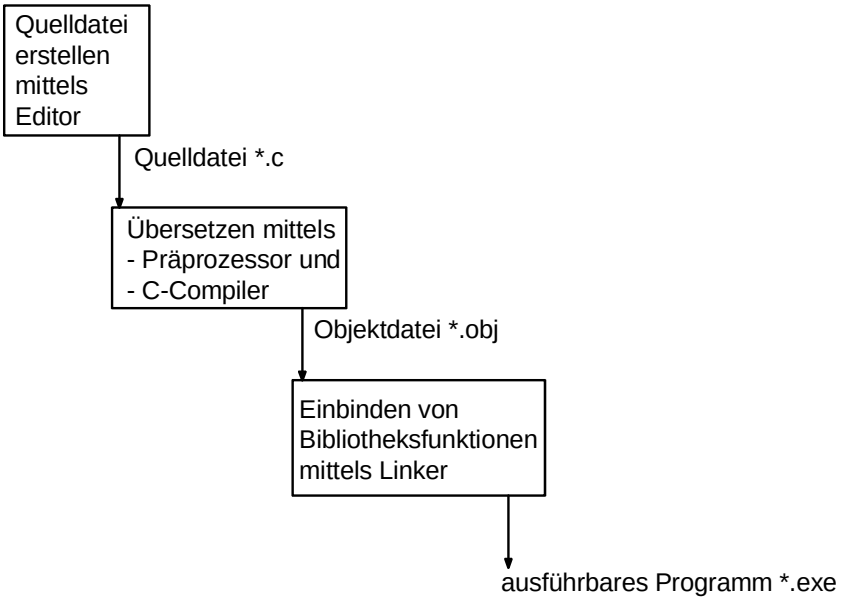

Abb. 1.2. Erstellung eines ausführbaren C-Programms.