Datentypen in C

Universitat Hamburg
Fachbereich Informatik
Proseminar: C-Grundlagen und Konzepte

Jan Branitzki

Inhalt

1. Was sind Datentypen?

Im Kontext von Programmiersprachen meint man mit dem Begriff "Datentyp" im allgemeinen eine Zusammenfassung von Operationen und Wertebereichen bestimmter Einheiten.
Dementsprechend bildet ein Datentyp die Grundlage zum Verarbeiten von Einheiten.
Als "elementaren Datentypen" bezeichnet man die Datentypen zur Verarbeitung von Zahlen, Zeichen und ähnlichem.
Als bekannteste Beispiele sind wohl Integer, Float, Double und Char zu nennen.
Sie werden zur Verarbeitung von Ganzzahlen(Integer), Gleitkommazahlen(Floats, für doppelte Präzision Doubles) und Zeichen(Char) verwandt und bieten als solche Datentypen Zahlen-/Zeichenrepräsentationen und Rechenoperationen für diese an.

Außer den elementaren Datentypen gibt es noch weiterführende Datentypen, die wiederum elementare Datentypen nutzen, um ihre Einheiten zu repräsentieren.
Als solche wären z.B. Strings zu nennen, welche verwandt werden, um Zeichenketten darzustellen.
Datentypen können in Datenstrukturen, wie Arrays, Stacks, Listen, oder ähnlichem, verwaltet werden.
Datenstrukturen sind im Grunde auch nur Datentypen, welche als Einheiten Datentypen haben und Speicherstrukturen, sowie Operationen für die Verwaltung dieser anbieten.
Datenstrukturen können weitaus komplexer werden, als hier dargestellt und werden daher in dieser Ausarbeitung von mir nicht behandelt.

In C und vielen weiteren Sprachen werden Chars intern als Integer gehandhabt und können auch genauso verrechnet und behandelt werden.

1.1. Welche elementaren Datentypen bietet C? Und wie groß sind sie?

In C sind Integer und Floats bzw. Doubles in diversen Größen, bzw. Genauigkeiten definiert, welche und wie man das z.T. Beeinflussen kann, wird in Kapitel 2 genauer behandelt.
Im C-Standart ist lediglich definiert, dass es einen Datentypen für Chars geben muss, der der kleinstmögliche Integer auf diesem System sein muss. Er Muss einen Wertebereich von 255 Werten abdecken, wie viele Bytes er dafür benutzt, ist nich festgelegt.
Außerdem muss ein Short Integer angeboten werden, welcher mindestens so viele Bytes wie der Char-Datentyp benutzt.
Es muss einen "normalen" Integer geben, welcher mehr oder gleich viele Bytes, wie der Short Integer benutzen muss.
Dieser soll der für die Maschine natürlichste Integer sein. Er nutzt also die Standartwortbreite der Maschine.
Darüber hinaus sind Long- und Long Long Integer-Datentypen angegeben, welche entsprechend größer sein sollen und einen Wertebereich von bis zu 2^{64} = 18446744073709551616(LLong Int)[1] Werten abdecken sollen.
Für Gleitkommazahlen muss ein Float-Datentyp vorhanden sein, wobei nicht angegeben ist, wie groß er sein muss.
Darüber hinaus definiert der C-Standart Double und Long Double-Datentypen, welche Gleitkommazahlen mit doppelter Präzision darstellen sollen.
Alle diese Datentypen können signed oder unsigned, also vorzeichenbehaftet, oder nicht sein.
Unsigned sind sie immer positiv und können einen größeren Wertebereich abdecken, als signed, da das Vorzeichen ein Byte braucht und somit den Zahlenbereich fast halbiert.
Wieviele Bytes ein Datentyp benutzt ist also von Maschine und Compiler abhängig und kann nicht genau gesagt werden.
Für den GNU C-Compiler und den Microsoft Visual C-Compiler habe ich eine Übersichtstabelle der Größen verschiedener Datentypen erstellt.

Name	Anderer Name	Größe* bei MCC**	Größe* bei GCC***	Zahlenbereich
unsigned char	Unsigned _int8	1	1	0…255
signed char	Signed _int8	1	1	-128…127
wchar_t	_wchar_t	2	2	0…65.535
unsigned short int	Unsigned _int16	2	2	0…65.535
signed short int	Signed _int16	2	2	-32.768…+32.767
unsigned int	Unsigned _int32	4	4	0…4.294.967.295
signed int	Signed _int32	4	4	–2.147.83.648…2.147.483.647
unsigned long int	Unsigned _int64	8	8	0…18.446.744.073.709.551.615
signed long int	Signed _int64	8	8	–9.223.372.036.854.775.808…9.223.372.036.854.775.807
(signed) float	-	4	4	3,4E-38…3,4E38
(signed) double	-	8	8	1,7E-308…1,7E308
(signed) long double	-	8	12/16(Prozessorarchitektur)	MCC: s. Double GCC: 3,65E-4951…1,18E4932
(signed) float complex	-	8	8	Reell: s. Float; Imaginär: s. Float
(signed) double complex	-	16	16	Reell: s. Double; Imaginär: s. Double
(signed) long double complex	-	16	24/32(Prozessorarchitektur)	Reell: s. Long double; Imaginär: s. Long double

* in Byte auf 32/64Bit-Maschinen
**Microsoft Visual C-Compiler
***GNU C-Compiler

Siehe Quellen 3.

2. Der Datentyp Integer

Zur Darstellung von Natürlichen Zahlen gibt es den Datentyp Integer.
In nahezu allen modernen Programmiersprachen wird er genutzt.
Der Datentyp Integer bietet eine Reihe an Operationen zur Verrechnung Natürlicher Zahlen, wie Addieren, Subtrahieren, Multiplizieren, Dividieren und ähnliches.
Wie diese Operationen implementiert werden können, wird in Kapitel 2.1 behandelt.
Castet man einen Float in einen Integer, so wird der Teil nach dem Komma "abgetrennt" und verfällt.
Wie bereits erwähnt gibt es keine garantierte Aussage darüber, wieviele Bytes ein Integer benutzt. Allerdings gibt es Möglichkeiten, die Größe des Integers zu erfragen und so, z.B. über Fallunterscheidung mit der Breite des Integers arbeiten.

Des Weiteren ist es möglich, Integer mit festen Größen zu definieren, die auf jeden Fall die Größe (in Byte) haben, die man einstellt.
Dies wird in Kapitel 2.2 näher behandelt.

2.1. Integertypen mit bestimmten Größen

Da C keine Garantie für die Größe von ints gibt, es aber mitunter notwendig ist, einen Integer einer gewissen Größen zu benutzen, bietet C die Möglichkeit an, Integers mit definierten Größen zu erstellen.
Hierfür muss die Header-Datei <stdint.h> eingebunden werden.
Diese bietet verschiedene Typedefs zum definieren von Integern mit bestimmter Größe:

int*N*_**t** *x* oder **uint***N*_**t** *x*:
Für einen unsigned Integer *x* der Breite *N,* wobei *N* eine Potenz von 2, kleiner als 64, oder 24 sein muss.

int_least*N*_**t** *x* oder **uint_least***N*_**t** *x*:
Für einen unsigned Integer *x* mit Mindestbreite *N*.

int_fast*N*_**t** *x* oder **unint_fast***N*_**t** *x*:
Für einen Integer *x* vom schnellsten Integer-Typ, den das System anbietet, mit Mindestbreite *N*.

intmax_t *x* oder **uintmax_t** *x*:
Für einen größtmöglichen Integer *x*. Dies kann mitunter der langsamste und speicheraufwendigste Integer des Systems sein.

intptr_t *x* oder **uintptr_t** *x*:
Für einen Integer *x*, der numerische Werte von Pointern aufnehmen kann.

Es ist natürlich möglich, immer intmax_t zu benutzen und sich überhaupt keine Gedanken über die Größe des Integers zu machen. Dies ist allerdings nich zu empfehlen, da je nach System intmax_t der langsamste Integer des System ist und die Operationen bedeutend länger brauchen, als bei anderen Integern.

3. Datentypen zur Repräsentation von Gleitkommazahlen

Zur Repräsentation von Gleitkommazahlen gibt es den Datentypen Float, bzw., für doppelte Genauigkeit, Double. Sie sind nicht absolut genau, da die Gleitkommazahl exponentiell angenähert werden.
Die Formel hierfür lautet:
Die Floatzahl $r=b^{e}*m$, wobei b die Basis der Zahl, meist 2, seltener 10, oder 16, bei IBM-Mainframes 16^{2}; m die Mantisse mit einer gewissen Genauigkeit(welche bestimmte Zahl an Stellen hat und so die Genauigkeit beeinflusst) und e der Exponent ist.
Die Größe des Floats hängt also fast auschließlich von der Mantisse ab. Dadurch ergibt sich der signifikante Größenunterschied von Double zu Float, bei "lediglich" verdoppelter Genauigkeit.
Wie genau, oder groß der Float oder Double auf einem System ist, ist nicht festgelegt und kann sich von Compiler zu Compiler und von System zu System unterscheiden.

4. Der Aufzählungstyp enum

C bietet für Aufzählungen einen eigenen Datentypen an, den Datentypen enum. Bei der Erstellung eines enums werden Werte angegeben, die den Aufzählungsschritten entsprechen sollen.
Ein enum wird erstellt mit

enum *name* {*Wert0*, *Wert1, ... , Wertn*},

wobei *Wert0* bis *Wertn* die Werte sind, die das enum bei der jeweiligen Zahl ausgeben soll.
So kann man z.B. ein enum erstellen, welches von 1 bis 4 geht und die die Zahlen als String ausgibt.
Man würde also der 1 die "Eins" zuweisen, der 2 die "Zwei", bis hin zur 4 die "Vier".
Intern wird das enum als Integer behandelt und bietet daher alle Operationen einen normalen Integers an. Allerdings ist die Ausgabe die für den Wert festgelegte Entsprechung des enum-Wertes.
So wäre es z.B. bei dem erwähnten Integer-zu-String-enum möglich, wenn es momentan bei "Drei" ist, 1 zu addieren und man bekäme "Vier".
Es ist möglich, ein enum bei einem festegelegten Wert starten zu lassen.
Die Syntax wäre hier beispielsweise:

enum *name* {*Wert0 = 42, Wert1, ... Wertn*}.

Dieses enum würde bei 42 starten, das erste Element wäre also bei 42, das zweite bei 43, etc.

Gibt man keinen Anfangswert an, startet ein enum immer bei 0.
Ebenfalls ist es möglich, ein enum nicht am ersten Element zu indizieren, sondern an einem späteren Element, die Syntax hierfür wäre, wieder mit Beispielwert 42:

enum *name* {*Wert0, Wert1, Wert2=42, ... , Wertn*}.

Das enum beginnt nun wieder bei 0 für *Wert0* und *Wert1* ist der 1 zugeteilt, allerdings wird *Wert2* erst bei 42 erreicht, *Wert3* bei 43, etc.

5. Repräsentation von Komplexen Zahlen

Zur Repräsentation von Komplexen Zahlen wurde der Datentyp Complex entwickelt, er besteht technisch gesehen aus zwei Gleitkommazahlwerten, zur Repräsentation des reellen und imaginären Teils der Zahl, und hat für Komplexe Zahlen angepasste Operationen.
Um mit Complexes arbeiten zu können, muss die Headerdatei <complex.h> eingebunden werden.
Dann kann ein Complex mit folgender Syntex erstellt werden:

float _Complex *c* = 2.0 + 3.0***I**;

oder

double _Complex *c* = 2.0 + 3.0***I**;

Statt **_Complex** kann auch das Typedef **complex** verwandt werden.
Der imaginäre und der reelle Teil des Complex müssen vom gleichen Datentypen sein.

Demnach ist die Größe eines Complex von der Größe der Floats bzw. Doubles auf dem System und dem Compiler abhängig.
Es gibt verscheidene Operatoren zum verrechnen der Complexes und Operationen für den Zugriff auf den imaginären bzw des reellen Teil des Complexes.
Mit

creal(*c*)

und

cimag(*c*),

wobei *c* ein Wert im Complex-Datentyp ist kann so auf die jeweiligen Teile der Komplexen Zahl zugegriffen werden.

6. Wann welchen Datentypen nutzen?

Diese Frage ist pauschal eigentlich nicht zu beantworten. Es kommt immer stark darauf an, was man vorhat zu tun.
Zum Beispiel ist, wie bereits in Kapitel 2 erwähnt, durchaus möglich sich nicht groß Gedanken darum zu machen, welchen Integer-Typ man verwendet, aber birgt dies Risiken und bringt mitunter enorme Performanzverluste mit sich.
Iteriert man beispielsweise über ein Array der Größe 10 ist es nicht notwendig, einen Long Integer zum speichern der Laufvariable zu verwenden, ein Short Integer reicht hier vollkommen aus.
Man könnte jetzt auf die Idee kommen, statt eines Short Integers einen Char-Datentypen zu verwenden, schließlich sind Chars ja auch nur Integer-Typen. Dies ist zwar möglich, sollte aber vermieden werden, da der Code unleserlich wird und es unprofessionell wirkt, Datentypen zu "missbrauchen".
Bei wissenschaftlichen Berechnungen wird man nicht um Gleitkommazahlen herumkommen und meist ist es hier von enomer Wichtigkeit eine möcglichst hohe Genauigkeit zu haben, allerdings kann dies zu großen Einbußen bei der Berechnungszeit und zu hohem Speicheraufwand führen. Hier muss man sich immer Gedanken machen, wo die Prioritäten liegen.
Man sollte sich immer Gedanken darüber machen, ob das Programm auch auf anderen System läuft bzw laufen soll und dementsprechend an einigen Stellen sicherstellen, dass auch die richtigen Integer-Typen verwandt werden und im Zwiefelsfall lieber einen mit exakter Breite aus <stdint.h> benutzen, als sich auf sein Glück zu verlassen.

Quellen

1. C-Standart : N1548 Committee Draft — December 2, 2010 **ISO/IEC 9899:201x** §6.2.6
http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf – abgerufen 13.05.13
2. Datentyp Float http://www2.informatik.uni-halle.de/lehre/c/c623.html – abgerufen 15.05.13
3. Datentyp-Bereiche http://msdn.microsoft.com/de-de/library/vstudio/s3f49ktz.aspx – abgerufen 22.05.13