Today's contents

- Variable names
  - Naming rule of variables, keywords
- Data types
  - Integral type and character type
    - radix
    - character constant, integer constant
  - Floating type
    - float constant
  - Array type
- Operator
  - Arithmetic operators, logical operators, bitwise operators, assignment operators
  - "sizeof" operator, "size_t" type, "typedef" declaration

今日の内容

- 変数名
  - 変数名のルール、予約語
- データ型
  - 整数型と文字型
    - 基数
    - 文字定数、整数定数
  - 浮動小数点型
    - 浮動小数点定数
  - 配列型
- 演算子
  - 算術演算子、論理演算子、ビット演算子、代入演算子
  - sizeof 演算子、size_t 型、typedef 宣言

Naming rule of variables

- Characters allowed as variable identifiers in C language
  - Alphabets (up-case, low-case) or underscore (`_’) as a first character
  - Alphabets (up-case, low-case), digits, or underscore (`_’) as following characters
  - Symbols cannot be used for variable names except for ‘_’.
  - Digits cannot be used as a first character of variable names.
  - It is better to think that you cannot use Japanese characters for variable names.
    - You can use it in C language specification, but not in real implementations.

変数名の付け方

- C言語の変数の識別子として許されるもの
  - 先頭文字がアルファベット（大文字、小文字）またはアンダースコア文字（'_'）
  - 2文字目以降は、アルファベット、数字、アンダースコア文字
  - アンダースコア以外の記号は変数名には使えない
  - 数字で始まる変数名は使えない
  - 日本語は変数名として使えない（と思っておいた方が良い）
    - 言語仕様上は使用できるようになっているが、使えるように実装されているものがない

Keywords

- Keywords cannot be used as variable names.

auto, break, case, char, const, continue,
default, do, double, else, enum, extern,
float, for, goto, if, inline, int, long,
register, restrict, return, short, signed,
sizeof, static, struct, switch, typedef,
union, unsigned, void, volatile, while,
_Bool, _Complex, _Imaginary

  - “inline”, “restrict”, “_Bool”, “_Complex”, “_Imaginary” are new keywords from C99.

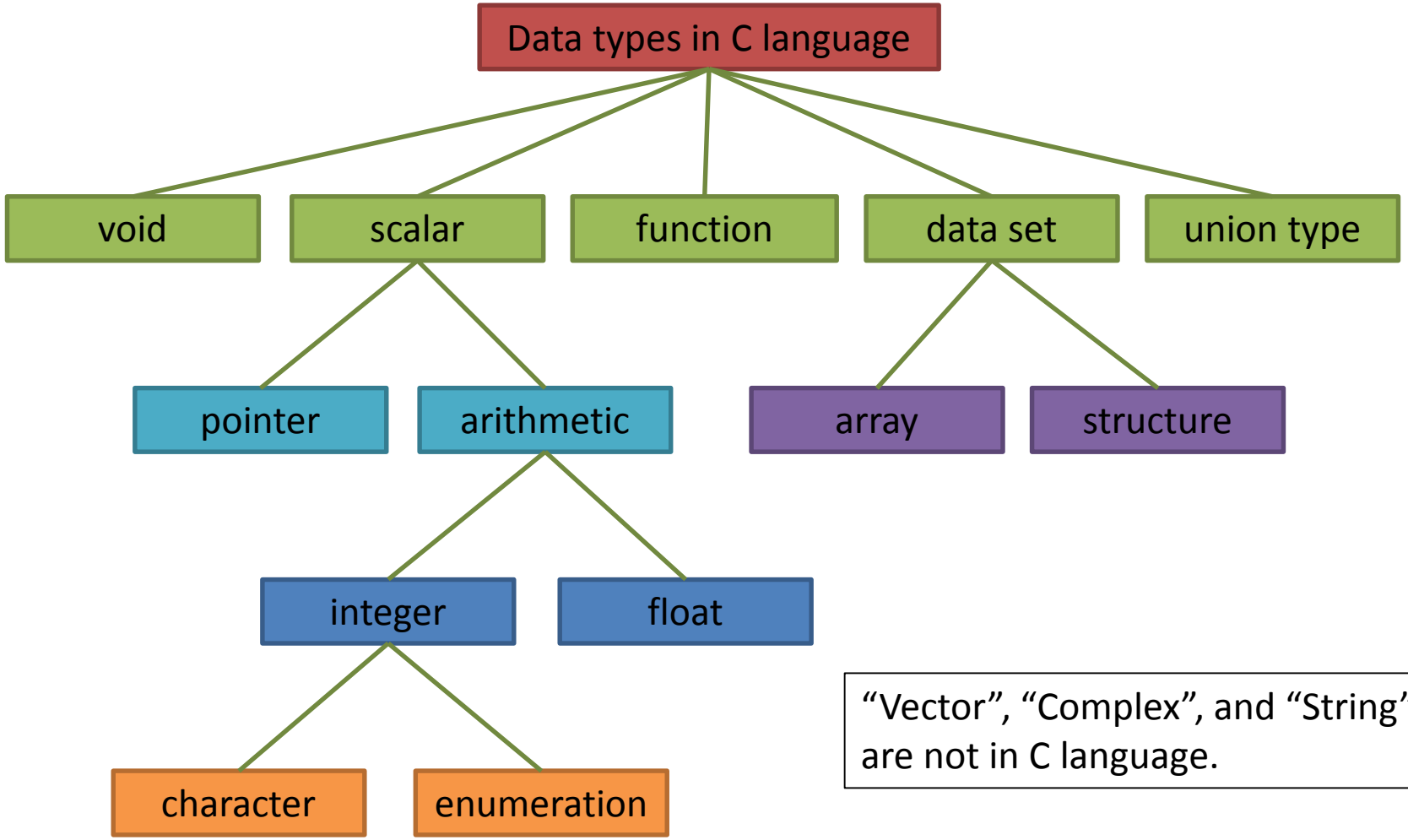
予約語（キーワード）

- 予約語は変数名として使えない

auto, break, case, char, const, continue,
default, do, double, else, enum, extern,
float, for, goto, if, inline, int, long,
register, restrict, return, short, signed,
sizeof, static, struct, switch, typedef,
union, unsigned, void, volatile, while,
_Bool, _Complex, _Imaginary

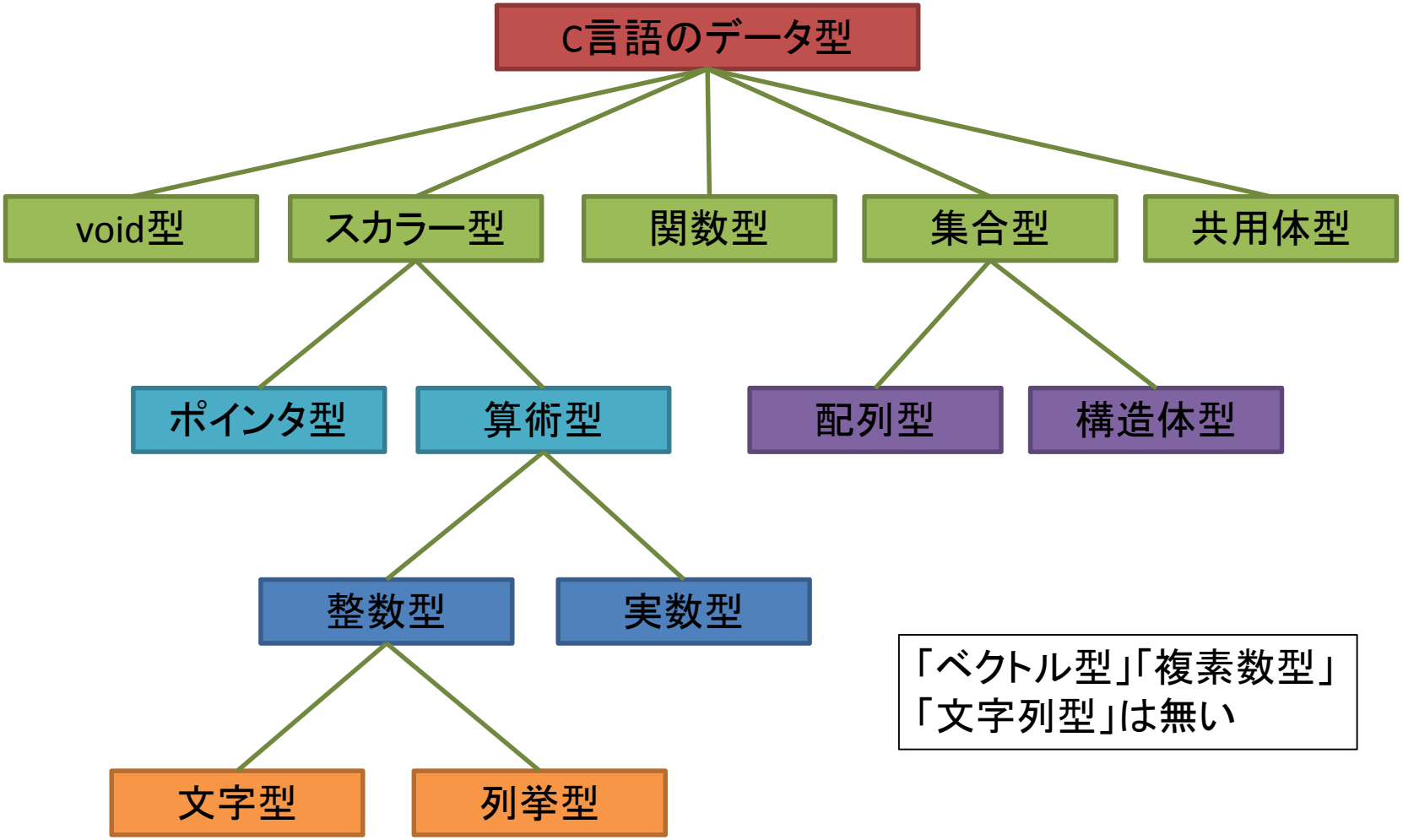
  - “inline”, “restrict”, “ _Bool”, “_Complex”, “_Imaginary” は、C99の新しい予約語

Data types in C language



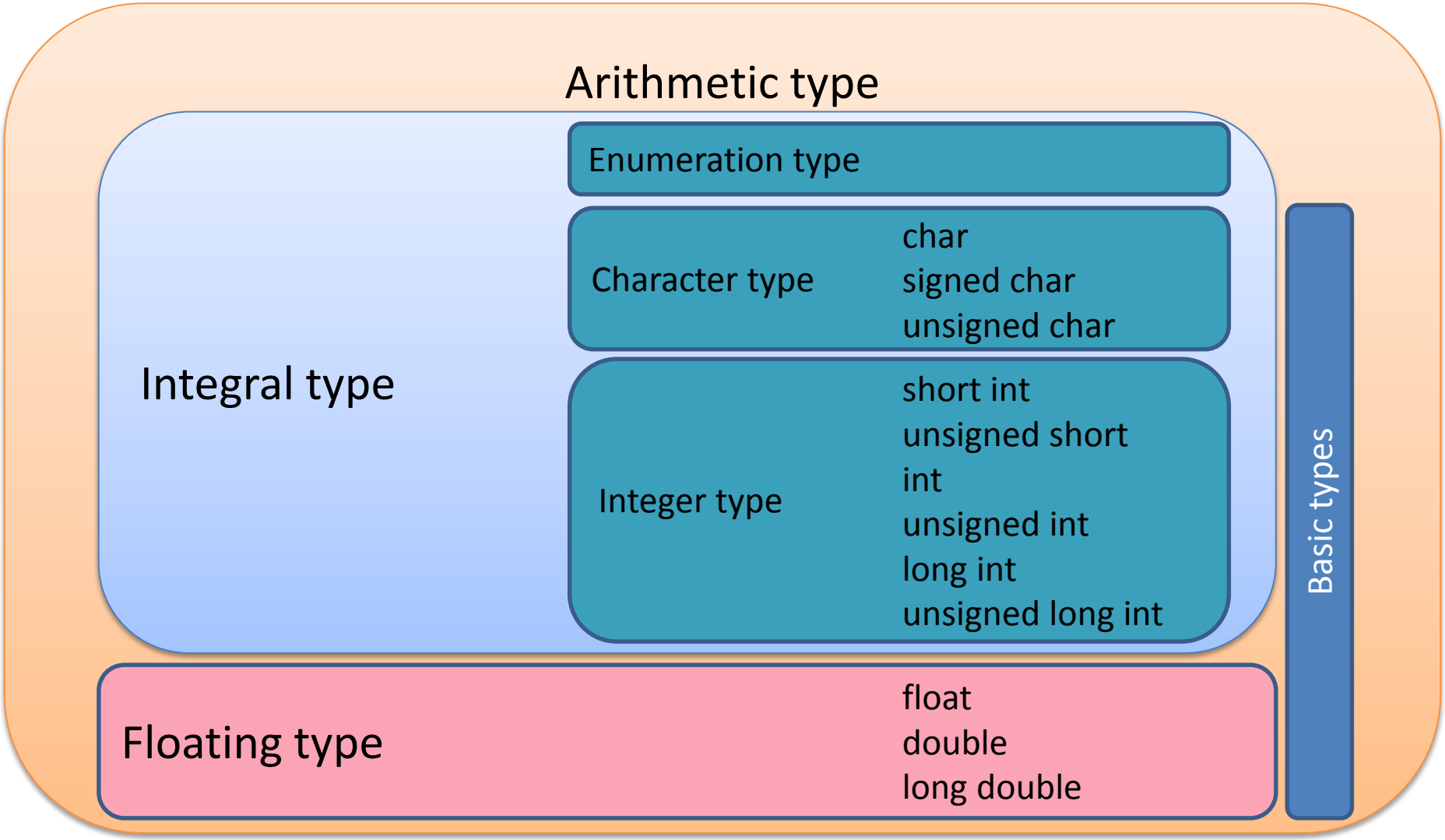
"Vector", "Complex", and "String" are not in C language.

C言語で使えるデータ型

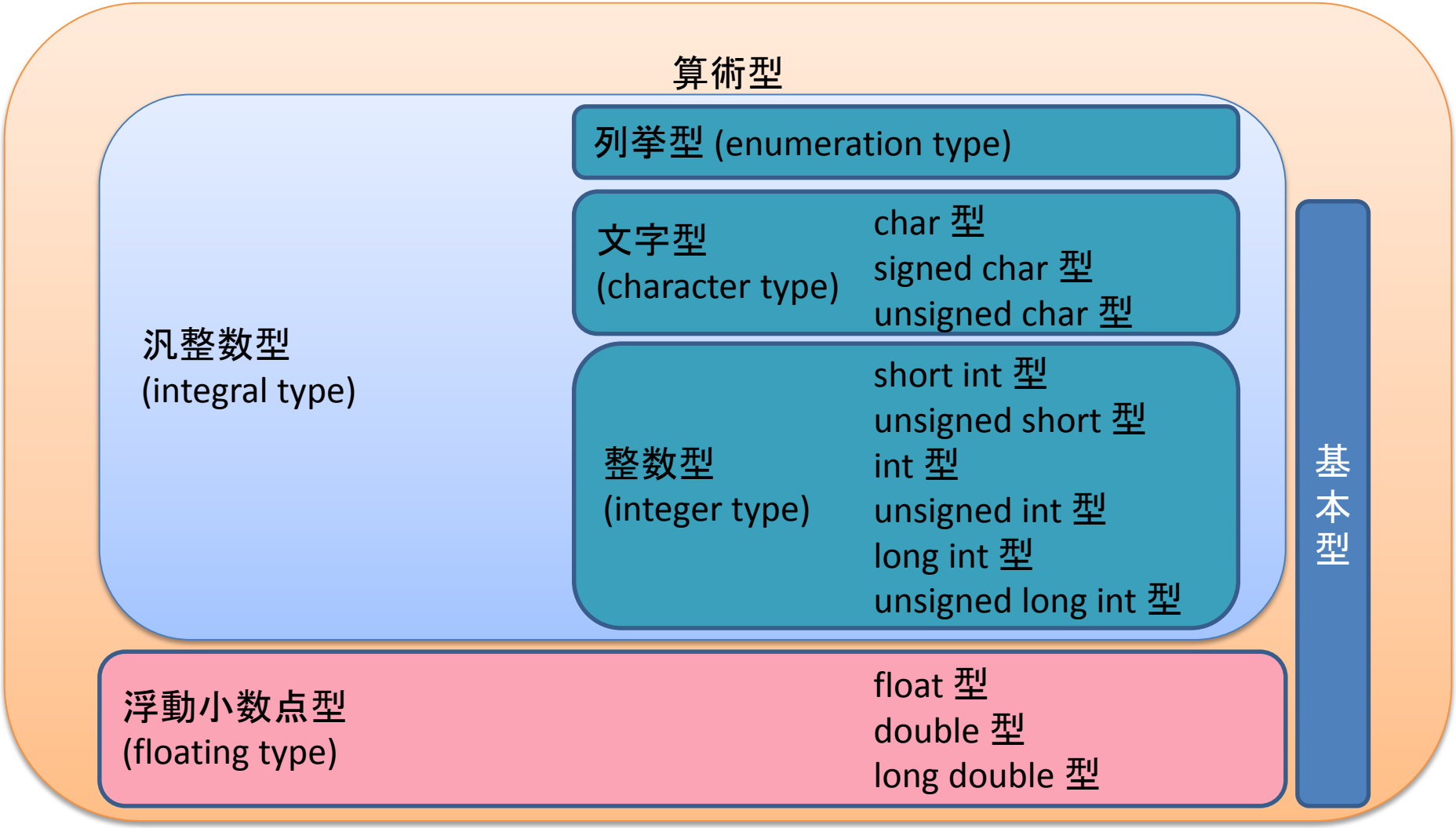


「ベクトル型」「複素数型」
「文字列型」は無い

Basic types and numbers

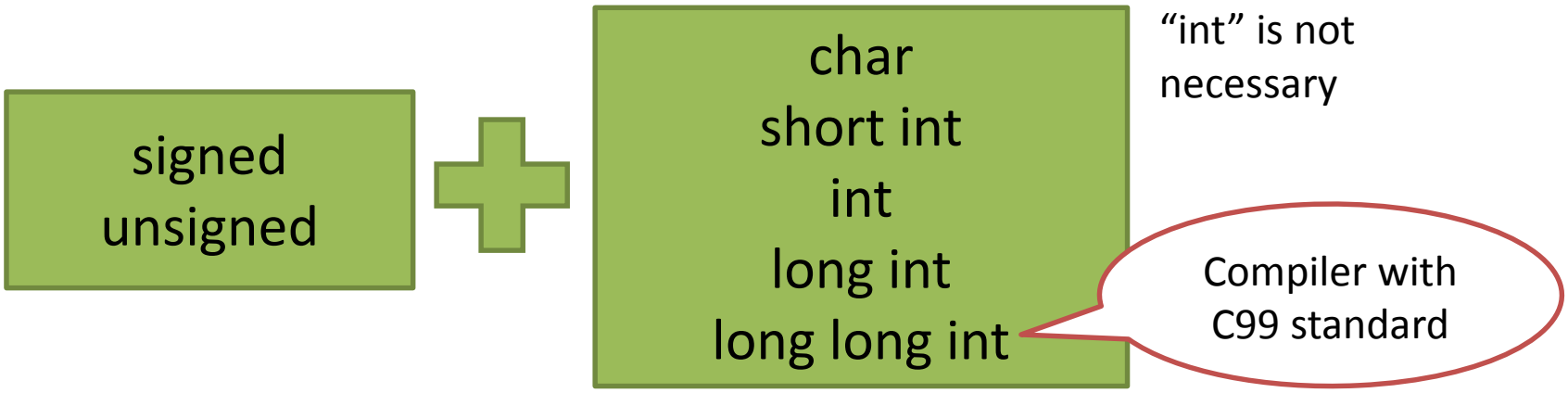


基本型と数


算術型
列挙型 (enumeration type)
文字型
(character type)
char 型
signed char 型
unsigned char 型
汎整数型
(integral type)
整数型
(integer type)
short int 型
unsigned short 型
int 型
unsigned int 型
long int 型
unsigned long int 型
浮動小数点型
(floating type)
float 型
double 型
long double 型
基本型

Integer type and character type (1)

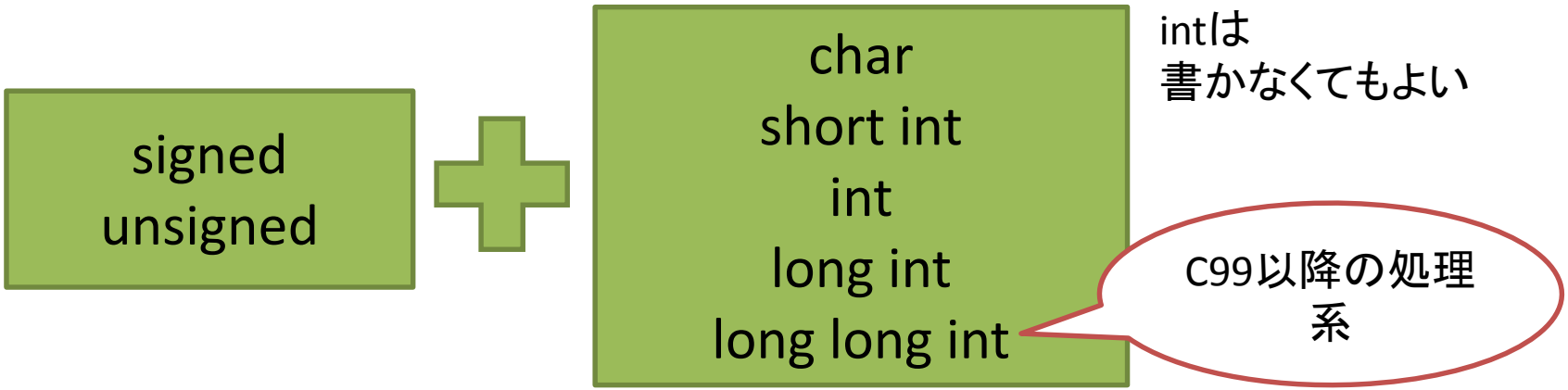
- Signed integer and unsigned integer
  - signed type specifier / unsigned type specifier
  - signed automatically if signed / unsigned is not specified.



  - The numerical range depends on the computer ⇒ <limits.h>
  - C language standards defines the MINIMUM range of it.

整数型と文字型（1）

- 符号付き整数と符号無し整数
  - signed 型指定子／unsigned 型指定子
  - signed／unsigned を指定しなければ、signedになる



  - 表わせる数の範囲は処理系に依存する ⇒ <limits.h>
  - 「最低限」は言語仕様で決められている

Integer type and character type (2)

- Character type
  - char
  - Neither signed nor unsigned
  - Actually, signed or unsigned inside of the computer
    - Depends on the computer
    - Important when comparing small and large of characters?
      - ASCII code use only 7 bits. It is not affected by sign bit.
      - If the computer uses 8 bit characters, the small and large is changed depending on the sign bit.
  - Some compiler can treat multi-byte character type (wchar_t).

整数型と文字型（2）

- 文字型
  - char 型
  - signedでもunsignedでもない
  - コンピュータ内部ではsignedかunsignedのどちらかで扱われている
    - どちらになるかは処理系依存
    - 文字の大小を比較する際に重要？
      - ASCIIコードは7ビットしか使わないので、signedでもunsignedでもどっちでも良かった
      - 8ビット目は符号ビットなので、signedとunsignedで大小が変わる
  - マルチバイト文字を扱う型（wchar_t）を持つ処理系もある

Character constant (1)

- Like ’a’
  - one character within single quotes
  - It’s “Integral type” so that it can be arithmetic operated.

```
char  c,  d;

c  =  ’A’;

if  (  c  >=  ’a’  &&  c  <=  ’z’  ) {  /*  up-case? low-case? */
  …
}

d  =  ’C’  +  1;  /*  d will be ’D’ */
```

文字定数(1)

- 「'a'」のように表す
  - シングルクオートで囲む
  - 汎整数型なので、算術演算も可能

```
char c, d;

c = 'A';

if ( c >= 'a' && c <= 'z' ) { /* アルファベットの小文字判定 */
  …
}

d = 'C' + 1; /* d は 'D' になる */
```

Character constant (2)

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

文字定数(2)

文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進	文字	10進	16進
NUL	0	00	DLE	16	10	SP	32	20	0	48	30	@	64	40	P	80	50	`	96	60	p	112	70
SOH	1	01	DC1	17	11	!	33	21	1	49	31	A	65	41	Q	81	51	a	97	61	q	113	71
STX	2	02	DC2	18	12	"	34	22	2	50	32	B	66	42	R	82	52	b	98	62	r	114	72
ETX	3	03	DC3	19	13	#	35	23	3	51	33	C	67	43	S	83	53	c	99	63	s	115	73
EOT	4	04	DC4	20	14	$	36	24	4	52	34	D	68	44	T	84	54	d	100	64	t	116	74
ENQ	5	05	NAK	21	15	%	37	25	5	53	35	E	69	45	U	85	55	e	101	65	u	117	75
ACK	6	06	SYN	22	16	&	38	26	6	54	36	F	70	46	V	86	56	f	102	66	v	118	76
BEL	7	07	ETB	23	17	'	39	27	7	55	37	G	71	47	W	87	57	g	103	67	w	119	77
BS	8	08	CAN	24	18	(	40	28	8	56	38	H	72	48	X	88	58	h	104	68	x	120	78
HT	9	09	EM	25	19	)	41	29	9	57	39	I	73	49	Y	89	59	i	105	69	y	121	79
LF*	10	0a	SUB	26	1a	*	42	2a	:	58	3a	J	74	4a	Z	90	5a	j	106	6a	z	122	7a
VT	11	0b	ESC	27	1b	+	43	2b	;	59	3b	K	75	4b	[	91	5b	k	107	6b	{	123	7b
FF*	12	0c	FS	28	1c	,	44	2c	<	60	3c	L	76	4c	\¥	92	5c	l	108	6c	|	124	7c
CR	13	0d	GS	29	1d	-	45	2d	=	61	3d	M	77	4d	]	93	5d	m	109	6d	}	125	7d
SO	14	0e	RS	30	1e	.	46	2e	>	62	3e	N	78	4e	^	94	5e	n	110	6e	~	126	7e
SI	15	0f	US	31	1f	/	47	2f	?	63	3f	O	79	4f	_	95	5f	o	111	6f	DEL	127	7f

* LFはNL、FFはNPと呼ばれることもある。
* 赤字は制御文字、SPは空白文字(スペース)、黒字と緑字は図形文字。
* 緑字はISO 646で割り当ての変更が認められており、例えば日本ではバックスラッシュが円記号になっている。

Character constant (3)

- Special characters
  - ’¥0ZZZ’: character definition in octal (Z denotes digits in octal)
  - ’¥xZZ’: character definition in hexadecimal (Z denotes digits in hex.)
  - ’¥0’: character string terminal (NULL character)
  - ’¥a’: bell
  - ’¥b’: back space
  - ’¥n’: line feed
  - ’¥r’: carriage return
  - ’¥t’: horizontal tab
  - ’¥v’: vertical tab
  - ’¥f’: new page
  - ’¥?’: ? symbol (question mark)
  - ’¥’’: ’ symbol (single quote)
  - ’¥”’: ” symbol (double quote)
  - ’¥¥’: back slash

文字定数(3)

- 特殊文字
  - '¥0ZZZ' : 8進数での文字定義(Zは0～7の数字)
  - '¥xZZ' : 16進数での文字定義(Zは0～9の数字またはa～f)
  - '¥0' : 文字列終端を表す(ヌル文字)
  - '¥a' : ベル
  - '¥b' : バックスペース
  - '¥n' : 改行
  - '¥r' : 復帰(キャリッジ・リターン)
  - '¥t' : 水平タブ
  - '¥v' : 垂直タブ
  - '¥f' : 改ページ
  - '¥?' : クエスチョンマーク
  - '¥'' : シングルクオート
  - '¥"' : ダブルクオート
  - '¥¥' : バックスラッシュ

Functions for characters

- Definitions in <ctype.h>
  - for example,
    - int isalpha(int) : the character is an alphabet?
    - int iscntrl(int) : the character is a control code?
    - int isdigit(int) : the character is a digit?
    - int isprint(int) : the character code is printable?
    - int isspace(int) : the character is white space?

    - int tolower(int) : convert to lower-case character
    - int toupper(int) : convert to upper-case character

文字に関する関数

- <ctype.h>の中に定義がある
  - 例えば
    - int isalpha(int)：アルファベットかどうか
    - int iscntrl(int)：制御文字かどうか
    - int isdigit(int)：数字かどうか
    - int isprint(int)：印字可能かどうか
    - int isspace(int)：空白文字かどうか

    - int tolower(int)：小文字に変換
    - int toupper(int)：大文字に変換

Radix

- We usually use “Decimal”
  - Digits are 0, 1, ..., 9. Carry at 10.
- “Binary” is easily understood by computers.
  - only 0 / 1 (corresponding to low or high of voltage)
- “Octal”
  - packed 3 digits in binary into 1 digit
  - Digits are 0, 1, ..., 7. Carry at 8 (in decimal).
- “Hexadecimal”
  - packed 4 digits in binary into 1 digit
  - Digits are 0, 1, ..., 9+A, B, ..., F. Carry at 16 (in decimal).

基数

- 人間が普段使っているのは、10進数
  - 数字は0～9、10で桁上がり
- コンピュータが理解しやすいのは、2進数
  - 0/1のみ。電圧の高低に対応する。
- 8進数
  - 2進数を3桁毎にまとめたもの
  - 数字は0～7、8で桁上がり
- 16進数
  - 2進数を4桁毎にまとめたもの
  - 数字は0～9+A～F、(10進数の) 16で桁上がり

Integral constant

- Decimal constant
  - not with special notation, as it used.
- Octal constant
  - add ”0” (zero) as prefix
    - 012 … 12 in octal (in decimal, corresponding to 10).
- Hexadecimal constant
  - add ”0x”(zero + ”x”) as prefix
    - 0x12 … 12 in hex. (in decimal, corresponding to 18).

整数定数

- 10進定数
  - 特別な書き方をしない今まで使ってきたもの
- 8進定数
  - 先頭に”0”（数字のゼロ）を付ける
    - 012 … 8進数の12。10進数では10に相当。
- 16進定数
  - 先頭に”0x”（数字のゼロと”x”）を付ける
    - 0x12 … 16進数の12。10進数では18に相当。

Type of integral constants

- Integral postfix “u/U/l/L”
  - The number will be unsigned if adding ”u” or ”U” as a postfix.
  - The number will be long if adding ”l” or ”L” as a postfix.
  - Use CAPITAL LETTER because “l” and “1” are easily confused.

- Rule for constant type
  - decimal constant w/o postfix int → long → unsigned long
  - octal/hex. constant w/o postfix int → unsigned → long → unsigned long
  - with u/U unsigned → unsigned long
  - with l/L long → unsigned long
  - with both l/L and u/U unsigned long

整数定数の型

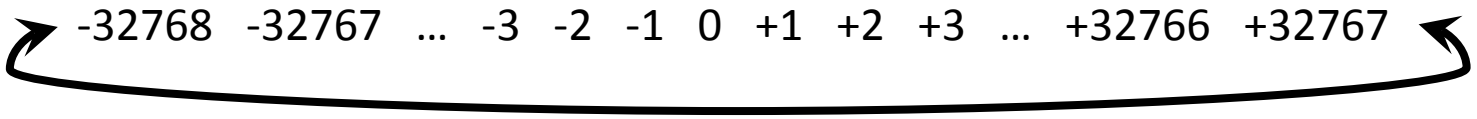
- 整数接尾語 u/U/l/L
  - 数値の後ろに”u”か”U”を付けると符号無しになる
  - 数値の後ろに”l”か”L”を付けると long になる
  - 小文字だと分かりにくい（特に”l”。数字”1”と見間違う）ので、大文字を使いましょう！

- 定数が最終的に何型になるかのルール
  - 接尾語無し10進定数　int → long → unsigned long
  - 接尾語無し8/16進定数　int → unsigned → long → unsigned long
  - u/U接尾語付き　unsigned → unsigned long
  - l/L接尾語付き　long → unsigned long
  - l/Lとu/U接尾語付き　unsigned long

Internal expression

- All information is represented by “bits” (0/1) inside of computers.

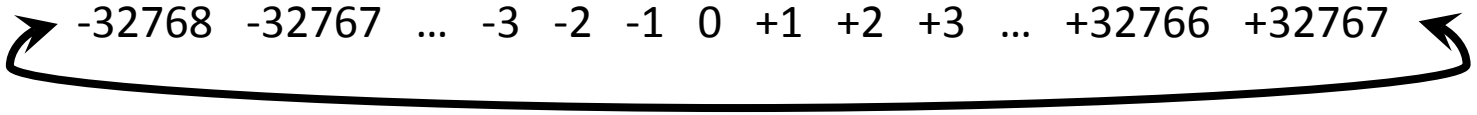
- Internal expression of signed integer
  - “Complement on two” expression
  - Most significant bit is “sign bit”.
  - For example, in the case of 16 bits:



内部表現

- コンピュータの内部では全てがビット（0/1）の並びで表現されている

- 符号付き整数の内部表現
  - だいたいが「2の補数」表現
  - 最上位ビットが符号ビット
  - 16ビットの場合



Floating type

- float / double / long double
  - “float” is “single precision floating point number”, “double” is “double precision floating point number”.
  - “long double” is “quadruple precision floating point number” or “double extended precision floating point number”.
  - Precision is different by internal expression
    - Generally “IEEE 754 format” is used
    - How internal expression ?
- Float constant
  - floating point postfix f/F/l/L
    - w/o postfix, the number will be “double”
    - with f/F, the number will be “float”
    - with l/L, the number will be “long double”

浮動小数点型

- float 型 / double 型 / long double 型
  - float は単精度実数、double は倍精度実数
  - long double は倍々精度実数 or 拡張倍精度実数
  - 内部表現により精度が異なる
    - 一般的には “IEEE 754 フォーマット”
    - どのような内部表現だろうか？

- 浮動小数点定数
  - 浮動小数点接尾語 f/F/l/L
    - 何も付けないと、double型になる
    - f/Fを付けると、float型になる
    - l/Lを付けると、long double型になる

Float constant (cont'd)

- Exponent of 10 represented by "e/E"
  - "80.0E-5" ⇒ "80.0×10^{-5}"
  - "6.02E+23" ⇒ "6.02×10^{23}"
- Abbreviation
  - ".0" ⇒ the same as "0.0"
  - "10." ⇒ the same as "10.0"
    - Notice: "10" is different from "10.0" ("10" is a integral number, "10.0" is a floating point number).

浮動小数点定数の続き

- “e/E”で10のべき乗を表現する
  - “80.0E-5” ⇒ “80.0×10^{-5}”
  - “6.02E+23” ⇒ “6.02×10^{23}”
- 省略形
  - “.0” ⇒ “0.0”の意味
  - “10.” ⇒ “10.0”と同じ意味
    - ”10”と”10.0”は別の型（前者は整数型、後者は浮動小数点型）になり、別の物なので注意

Trap of “==”

- Floating point number does not become “exact”.
  - This programs won’t finish correctly.

```
int main(void)
{
  float x;

  for (x = 0.0; x != 10.0; x += 0.01) {
    printf( “x = %f¥n”, x);
  }

  return 0;
}
```

    - x will never be “10.0” exactly !
  - Significand has limited bits. ⇒ Significant figures are limited. ⇒ It generates “rounding error”.

“==”の罠

- 浮動小数点数が“ピッタリ”にならない
  - このプログラムは終了しない

```
int main(void)
{
  float x;

  for (x = 0.0; x != 10.0; x += 0.01) {
    printf( “x = %f¥n”, x);
  }

  return 0;
}
```

    - xがちょうど10.0になることがない！
  - 仮数部が有限ビット ⇒ 有効数字が有限 ⇒ 丸め誤差が発生する

Over the usable numeric range

- Overflow
  - Over the upper limit of the usable numeric range
  - happens to both integer and float type

- Underflow
  - Under the lower limit of the usable absolute numeric range
    ⇒ Under the lower limit of the exponent part in a floating number

扱える値の範囲を超える

- オーバーフロー
  - 扱える値の範囲の上限を超える
  - 整数型でも浮動小数点型でも起きる
- アンダーフロー
  - 扱える絶対値の下限を越える
    ＝ 浮動小数点型で、指数部の最小値を下回る

Array type

- A data type which can manage a set of objects which have the same type by index numbers
- Used often for representing a “vector” for scientific operations

- defined by “variable_type variable_name[number_of_elements]”

```
/*  int type, array name is vc, number_of_elements is 5  */
int vc[5];
```

  - Notice: usable index is from 0 to number_of_elements - 1
    - The first index is 0 or 1 depending on the programming language.

配列型

- 同じ型のオブジェクトの集まりを、番号により管理できるようにしたデータ型
- 科学技術演算では、ベクトルを表現するためによく用いられる

- 配列は「型名　変数名[要素数]」で宣言

```
/* int型で、配列名が vc で、要素数が 5 だったら */
int vc[5];
```

  - 注意：使える要素は0～要素数-1まで
    - 開始要素のインデックスが0か1かは言語によって異なる

Arithmetic operators (1)

- Four arithmetic operators
  - “+”, “-”, “*”, “/”
- modulus operator
  - “%” : operator for remainder obtained by division

算術演算子(1)

- 四則演算
  - “+”, “-”, “*”, “/”
- 剰余算演算子
  - “%”:剰余(余り)を求める演算子

Arithmetic operators (2)

- Increment / Decrement operator
  - Increment operator “++”: add 1 to the variable
  - Decrement operator “--”: subtract 1 from the variable
  - Prefix type (“++a”) and Postfix type (“a++”)
    - Prefix type “adds/subtracts 1 before the expression evaluation”
    - Postfix type “adds/subtracts 1 after the expression evaluation”

```
int  a,  b,  c;
a  =  1;
b  =  ++a;  /*  after the evaluation, a = 2, b = 2  */
a  =  1;
c  =  a++;  /*  after the evaluation, a = 2, c = 1  */
```

算術演算子(2)

- インクリメント・デクリメント演算子
  - インクリメント演算子“++”:1を加える
  - デクリメント演算子“--”:1を引く
  - 前置型(“++a”)と後置型(“a++”)
    - 前置型は「式を**評価する前**に1を加える／引く」
    - 後置型は「式を**評価した後**に1を加える／引く」

```
int a, b, c;
a = 1;
b = ++a; /* a = 2, b = 2 となる */
a = 1;
c = a++; /* a = 2, c = 1 となる */
```

Relational and logical operators, Conditional expression

- Relational and logical operators
  - ! *expr* : negation of *expr*
  - *expr1* && *expr2* : logical AND of *expr1* and *expr2*
  - *expr1* || *expr2* : logical OR of *expr1* and *expr2*
  - *expr1* < *expr2* : *expr1* is less than *expr2* ?
  - *expr1* <= *expr2* : *expr1* is less than or equal to *expr2* ?
  - *expr1* > *expr2* : *expr1* is greater than *expr2* ?
  - *expr1* >= *expr2* : *expr1* is greater than or equal to *expr2* ?
  - *expr1* == *expr2* : *expr1* is equal to *expr2* ?
  - *expr1* != *expr2* : *expr1* is not equal to *expr2* ?

- Conditional expression (Ternary operator)
  - *expr1* ? *expr2* : *expr3* : if *expr1* is true, do *expr2*, or do *expr3*

論理演算子・条件演算子

- 論理演算子
  - ! 式：式の否定
  - 式１ && 式２：式１と式２の論理積
  - 式１ || 式２：式１と式２の論理和
  - 式１ < 式２：式１が式２より小さい
  - 式１ <= 式２：式１が式２以下
  - 式１ > 式２：式１が式２より大きい
  - 式１ >= 式２：式１が式２以上
  - 式１ == 式２：式１と式２が等しい
  - 式１ != 式２：式１と式２が等しくない

- 条件演算子（三項演算子）
  - 式１？式２：式３：式１が真なら式2を、偽なら式3を実行

Bitwise operator (1)

- Logical operator for each bit
  - Logical AND “&”
  - Logical OR “|”
  - Logical exclusive or (XOR) “^”
  - Logical negation (NOT) “~” (bit reversal)

- Notice: Bitwise operators are unrelated to “logical operators”
  - Especially, mistakes to write “&&”, “||” as “&”, “|”.
    - It’s difficult to be aware, because it can be compiled.

ビット演算（1）

- ビット単位の論理演算子
  - 論理積(AND)　“&”
  - 論理和(OR)　“|”
  - 排他的論理和(XOR)　“^”
  - 否定(NOT)　“~”（ビット反転）

- 論理演算子に似ているが、無関係なので注意！
  - 特に”&&”, ”||”との書き間違いに注意！
    - コンパイル出来てしまうので

Bitwise operator (2)

- Shift operator
  - Shift (or move) the whole bit field, then fill the empty bits by 0 or 1.
    - whether “logical shift (the empty bits will be filled by 0)” or “arithmetic shift (the empty bits filled by 0 if left-shift, by the signed bit if right-shift)” depending on the computer
  - Left-shift “<<”
  - Right-shift “>>”
    - easy to understand visually
  - Used for faster multiplying computation ..... traditionally
    - However, I cannot say that this is effective for current computers.
    - This does not seem HIGH-LEVEL PROGRAMMING LANGUAGE.

ビット演算（2）

- シフト演算
  - ビットフィールドを全体にシフト（移動）して、空いたビットを 0 か 1 で埋める
    - 「論理シフト（空いたビットを0で埋める）」になるか「算術シフト（左シフトなら0で、右シフトなら符号ビットで埋める）」になるかは処理系依存
  - 左シフト “<<”
  - 右シフト “>>”
    - 視覚的に分かりやすい演算子
  - 整数の掛け算の高速化……に昔は使っていた
    - しかし、今もこれで速くなるかどうかは何とも言えない
    - こういうことで高速化しようという発想が高級言語っぽくない

Assignment operators

- Assignment operator
  - a = b; : substituting b for a

- Complex assignment operators
  - a += b; : substituting adding result (a + b) for a
  - a -= b; : substituting subtracting result (a – b) for a
  - a *= b; : substituting multiplying result (a * b) for a
  - a /= b; : substituting dividing result (a / b) for a
  - a %= b; : substituting remainder (a % b) for a
  - a >>= b; : substituting right-shift result (a >> b) for a
  - a <<= b; : substituting left-shift result (a << b) for a
  - a &= b; : substituting logical AND for each bit (a & b) for a
  - a |= b; : substituting logical OR for each bit (a | b) for a
  - a ^= b; : substituting logical XOR for each bit (a ^ b) for a

代入演算子

- 代入演算子
  - a = b; : a に b を代入
- 複合代入演算子
  - a += b; : a に a と b を加えた値(a+b) を代入
  - a -= b; : a に a から b を引いた値(a-b) を代入
  - a *= b; : a に a と b の積(a*b) を代入
  - a /= b; : a に a を b で割った値(a/b) を代入
  - a %= b; : a に a を b で割った余り(a%b) を代入
  - a >>= b; : a に a を b ビット右シフトした値(a>>b) を代入
  - a <<= b; : a に a を b ビット左シフトした値(a<<b) を代入
  - a &= b; : a に a と b のビット論理積(a&b) を代入
  - a |= b; : a に a と b のビット論理和(a|b) を代入
  - a ^= b; : a に a と b のビット排他的論理和(a^b) を代入

Priority of the operators

Priority	Kinds of operators		Operators
↑ High	Expression (from left if the same priority)		( ) [ ] ->(structure) .(structure)
	Unary operator (from right if the same)		! ~ ++ -- (cast) + - *(pointer) &(address) sizeof
	Binary operator (from left)	multiplication, division, modulus	* / %
		addition, subtraction	+ -
		shift	<< >>
		comparison	< <= > >=
		equality, non-equality	== !=
		logical AND (bitwise operator)	&
		logical XOR (bitwise operator)	^
		logical OR (bitwise operator)	|
		logical AND	&&
		logical OR	||
	Ternary operator (from left)		? :
Low ↓	(Complex) Assignment operator (from right)		= += -= *= /= %= <<= >>= &= ^= |=
	Sequential operator (from left)		,(comma operator)

演算子の優先順位

優先順位	演算子の種類		演算子
↑ 高	式（同順位は左から評価）		( ) [ ] ->(構造体) .(構造体)
	単項演算子（同順位は右から評価）		! ~ ++ -- (キャスト) + - *(ポインタ) &(アドレス) sizeof
	二項演算子 （左から）	剰余	* / %
		加減	+ -
		シフト	<< >>
		比較	< <= > >=
		等値、非等値	== !=
		論理積（ビット演算子）	&
		排他的論理和（ビット演算子）	^
		論理和（ビット演算子）	|
		論理積	&&
		論理和	||
	三項演算子（左から）		? :
↓ 低	代入演算子、複合代入演算子（右から）		= += -= *= /= %= <<= >>= &= ^= |=
	順次演算子（左から）		,(カンマ演算子)

Cast (Type conversion)

- Type conversion
  - happens when doing operation between different type numbers or assignment for different type variables.
    - An arithmetic operation will be done by using the type with larger value range in the types of operands.
  - Basically, “The value is the same as it was when the converted type can keep the original value” and “The value is a little bit changed when the converted type cannot keep the original value”.
    - Digits after the decimal point are cut off when converting a floating point number into an integer number.
    - “Rounding” happens when converting an integer number into a floating point number if the number of bits of the integer number is bigger than the mantissa length (bits) of the floating point number.
    - “Rounding” happens when converting “double” into “float” because the mantissa length (bits) is shorten.
    - The value is the same as it was when converting “float” into “double”.

型変換

- 型変換
  - 型の違うもの同士の演算、型の違う変数への代入を行うとき、型変換が起こる
    - 算術演算は、より表現できる範囲の広い型に揃えて行われる
  - 基本的には「ある値をそのまま表現できる型に変換するとき、値は保存される」、「そのまま表現できない型に変換するとき、値は変化する」
    - 浮動小数点型の値を整数型に変換すると、小数点以下が切り捨てられる
    - 整数型を浮動小数点型に変換するとき、仮数部のビット数より整数型のビット数の方が多い場合は、値の丸めが起きる
    - double型をfloat型に変換するとき、仮数部のビット数が減少するため、値の丸めが起きる
    - float型をdouble型に変換する場合は、値は保存される

“sizeof” operator, “typedef” declaration

- “sizeof” returns the number of bytes which the variable or the type uses.
  - usage: “sizeof(variable name or type name)”

- “typedef” declaration
  - gives another name to a type

```
typedef   unsigned   size_t;
```

    - Many compiler defines like above.

sizeof 演算子・typedef 宣言

- 変数が使っているバイト数を調べることができる
  - “sizeof(型名)”として使う

- typedef 宣言
  - 型に別の名前を与える

```
typedef   unsigned   size_t;
```

    - 多くの処理系ではこうなっている