● 講義資料

★ ASCII コード表

	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x7
0	NULL	DLE	SP	0	@	P	‘	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	’	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	CS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	_
F	SI	US	/	?	O	_	o	DEL

★ Cの演算子

詳しくは K&R の「付録A」を参照のこと.

優先順位１ （左結合）

“()” 優先順位を変更する括弧

“[]” 配列参照演算子 †

“->” アドレス演算子 †

“.” 間接演算子 †

優先順位２ （右結合）

“!” 論理否定演算子

“~” １の補数演算子

“++”, “--” インクリメント演算子

“+”, “-” 単項プラス・マイナス演算子

“&” ポインタ参照演算子 †

“(type)” キャスト（型変換）演算子

“sizeof” sizeof 演算子

優先順位３ （左結合）

“*”, “/”, “%” 乗法演算子

優先順位４ （左結合）

“+”, “-” 加法演算子

優先順位５ （左結合）

“>>”, “<<” シフト演算子

優先順位６ （左結合）

“>”, “<”, “>=”, “<=” 比較演算子

優先順位７ （左結合）

“==”, “!=” 等置演算子

優先順位８ （左結合）

“&” ビット AND 演算子

優先順位９ （左結合）

“^” ビット XOR 演算子

優先順位１０ （左結合）

“|” ビット OR 演算子

優先順位１１ （左結合）

“&&” 論理 AND 演算子

優先順位１１ （左結合）

“~~” 論理 OR 演算子

優先順位１２ （左結合）

“?:+” ３項演算子 ‡

優先順位１３ （右結合）

“=”

“+=”, “-=”, “*=”, “/=”, “%=”

“&=”, “~=, “^=”

“<<=”, “>>=” 代入演算子

優先順位１４ （左結合）

“,” コンマ演算子 ‡

【注意】

- この中で † がついているものは講義の後半で解説する.
- この中で ‡ がついているものは講義では特に解説しないので, 各自で調べておくこと.
- それ以外のものは今回及び次回で解説する.

演算子の優先順位 (cf. K&R p. 65)

演算子	結合規則
() [] -> .	⇒
! ~ ++ -- +（単項） -（単項） & (type) sizeof	⇐
* / %	⇒
+ -	⇒
>> <<	⇒
> < <= >=	⇒
== !=	⇒
&	⇒
^	⇒
|	⇒
&&	⇒
||	⇒
?:	⇐
= += -= *= /= %= &= |= ^= <<= >>=	⇒
,	⇒

● 実習内容

【C言語】

1. 以下の ex04-1.c を入力してコンパイルおよび実行を行ってみよう.
   - 正の整数の除算は予想通りの結果を得ることができる.
   - 除算において, 少なくとも一方の被演算子が負の場合には, 除算の結果および余りの結果の符号は**処理系依存**となる. したがって, **負の数を被演算数に持つ除算（または余り）を行ってはいけない**.
   - 負の数に関係する除算で保証されることは,
     - (a/b)*b+a%b=a が成り立つ.
     - 余りの絶対値は除数の絶対値より小さい.

     の２点のみである.
2. 以下の ex04-2.c を入力してコンパイルおよび実行を行ってみよう.
   - 各種のビット演算の意味を正確に理解しよう.
   - i&1 の値は何を意味しているのだろうか？k また, それは負の数に対しても適用できるだろうか？
   - なぜ「やってはいけないこと」となるのかを考えてみよう.
3. 以下の ex04-3.c を入力してコンパイルおよび実行を行ってみよう.
   - それぞれの計算結果がどうなるかをきちんと考えてみよう.
   - なぜ「やってはいけないこと」となるのかを考えてみよう.

電子メールで「今日の講義の感想や意見」を送ってください.

★ ex04-1.c の内容

```
/* ex04-1.c           */
/* $Id: ex04-1.c,v 1.4 2004-04-23 10:06:25+09 naito Exp $ */
/* 負の数に関する除算 */

#include <stdio.h>

int     i, j, k, l ;

int main(int argc, char **argv)
{
  i = 3 ; j = 2 ;
  k = i/j ;  l = i%j ;
  printf("%d/%d = %d, modulo %d\n", i,j,k,l) ;

  /* ここからあとは実際には「やってはいけないこと」の例 */
  i = 3 ; j = -2 ;
  k = i/j ; l = i%j ;
  printf("%d/%d = %d, modulo %d\n", i,j,k,l) ;

  i = -3 ; j = 2 ;
  k = i/j ; l = i%j ;
  printf("%d/%d = %d, modulo %d\n", i,j,k,l) ;

  i = -3 ; j = -2 ;
  k = i/j ; l = i%j ;
  printf("%d/%d = %d, modulo %d\n", i,j,k,l) ;

  return 0 ;
}
```

★ ex04-2.c の内容

```
/* ex04-2.c           */
/* $Id: ex04-2.c,v 1.4 2004-04-23 10:24:04+09 naito Exp $ */
/* ビット演算 */

#include <stdio.h>

int     i, j ;

int main(int argc, char **argv)
{
  i = 2 ; j = 3 ;
  printf("i&1 = %d, j&1 = %d\n", i&1, j&1) ;

  i = 1 ;
  printf("%d, %d, %d\n", i<<1, i<<2, i<<3) ;
  j = i<<2 ;
  printf("%d, %d, %d\n", j>>1, j>>2, j>>3) ;

  i = 1 ; j = 2 ;
  printf("i|j = %d\n", i|j) ;

  i = 1 ; j = 1 ;
  printf("i^j = %d\n", i^j) ;

  i = -1 ;
  printf("%d, %d, %d\n", i<<1, i<<2, i<<3) ;

  /* ここからあとは実際には「やってはいけないこと」の例 */
  i = 1 ;
  printf("%d, %d, %d\n", i<<-1, i<<-2, i<<-3) ;
  j = -4 ;
  printf("%d, %d, %d\n", j>>1, j>>2, j>>3) ;

  return 0 ;
}
```


★ ex04-3.c の内容

```
/* ex04-3.c           */
/* $Id: ex04-3.c,v 1.5 2004-04-24 17:36:06+09 naito Exp $ */
/* 代入とインクリメント */

#include <stdio.h>

int     i, j, k, l ;

int main(int argc, char **argv)
{
  i = 0 ;
  printf("%d, ", i++) ;
  printf("%d, ", ++i) ;
  printf("%d, ", --i) ;
  printf("%d\n", i--) ;

  k = j = i = 0 ;
  j += 1 ; k -= 2 ;
  l = 1 ; l <<= 1 ;
  printf("%d, %d, %d, %d\n", i,j,k,l) ;

  i = 0 ;
  l = k = j = i + 1 ;
  printf("%d, %d, %d, %d\n", i,j,k,l) ;

  i = 0 ;
  printf("%d, %d\n", i,j=i+1) ;

  /* ここからあとは実際には「やってはいけないこと」の例 */
  i = 0 ;
  printf("%d, %d, %d, %d\n", i,j=i+1,k=j+1,l=k+1) ;
  printf("%d, %d, %d, %d\n", i++,++i,--i,i--) ;

  i = 1 ; j = 2 ;
  printf("%d\n", i+++j) ;
  /* これらはエラーになる
     (i++)++ ; i+++++j ;
  */
  i = 1 ; j = 2 ;
  printf("%d\n", i+--+-+-+-+-j) ;
  return 0 ;
}
```

【課題】

☆ exercise-03-3 文字型変数に値を代入することにより，「画面」に “Hello World” と表示するプログラムを書きなさい．（要するに「一文字づつ」出力するプログラムを書けと言うこと.）また, このプログラムを書き換えて, “Hello World” のアルファベットを１つづ後ろにずらした出力をするプログラムを書きなさい. すなわち “Ifmmp Xpsme” と出力するプログラムを書きなさい.

☆ exercise-03-4 先週のプログラム ex02-4.c の出力がなぜそうなるかを，「int 型の値の演算結果」，「演算の結合規則」というキーワードを用いて正しく説明しなさい.

★ exercise-04-1 ２回目の実習のプログラムと今回のプログラムをもとにして, **乗法演算子を用いずに** int 型の値が偶数か奇数かを判定するプログラムを書きなさい.

★ exercise-04-2 ビット演算子と + のみを用いて int 型の値の符号を反転させるプログラムを書きなさい.

★ exercise-04-3 ex04-2.c の中の「やってはいけないこと」となっている部分を, なぜ「やってはいけないのか」の理由を述べなさい.

★ exercise-04-4 ex04-3.c の中の「エラーになる」となっている部分を, なぜ「やってはいけないのか」の理由を述べなさい.

【注意】

- ex04-3.c の中の「やってはいけないこと」となっている部分のうち, 最初の２つの printf 関数の部分の理由は, 今の段階では解説することはできない.
- ex04-3.c の中の「やってはいけないこと」となっている部分のうち, 最後２つの printf 関数の部分の理由は明らかである．「何をやっているかわけがわからない」からである．つまり，「**文法的に意味があっても, わけのわからないプログラムは書いてはならない**」ということ.

● 前回の講義のキーポイント及び補足

- 「**１ビット**」 (1 bit) とは「２進数１桁の情報量」のこと.
- 「**１バイト**」 (1 byte) とは「その計算機において, 自然にアクセス可能なデータ長の最小値」のこと．多くの場合「１バイト＝８ビット」であるが, いつもそうなるとは限らない.
- 計算機内部での整数の表現は, **環境に依存した長さの**バイト数（ビット数）の２進数で表現され, 符号付き整数と符号無し整数がある．ただし, int は 16 ビット以上, long は 32 ビット以上であることが保証されている. (cf. K&R p. 32)
- 計算機内部での整数は, 最大値と最小値を持ち, 最大値を越える計算結果は一般には保証されない．ただし, k ビットの符号なし整数に関しては 2^k を法とする演算が保証される.
- 負の整数は２の補数による表現で表されることが多い.
- C言語では, 変数は利用する前に宣言（定義）を行わなければならない.
- いくつかのC言語の本では「long long」という型が紹介されているが, long long は ANSI の規格の範囲では定義されていない．（多くの処理系で利用することは可能だが, 規格の範囲のプログラムとはならない.）

● 前回の課題の解説

- int 型の「正の最大の整数」を定数として代入する場合には,

```
i = 0x7fffffff ;
```

として１６進定数で代入すべきである．これを１０進定数で

```
i = 2147483647 ;
```

と書くと, **１文字が異なっていてもそれに気が付くとは限らない**.

- より正しくは, このような「システムに依存した定数値」を用いるときには,

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

int main(int argc, char **argv)
{
  int i=INT_MAX ;
  printf("%d\n",i) ;
  return 0 ;
}
```

のように「システムヘッダファイル」（この場合は limits.h）を用いて, INT_MAX のようなあらかじめ定義された値を用いることが望ましい．なお limits.h の CHAR_BIT には１バイトのビット数が与えられている.

- int 型の「正の最大の整数」に 1 を加えると，「負の最小の整数」（絶対値が最大の負の整数）になると思われる．しかし, この事実は**必ずしもいつでも正しいわけではない**．そのため, この事実を利用してプログラミングを行ってはいけない.
- 一方, unsigned int 型の「最大の整数」に 1 を加えると 0 となる．この事実は**いつでも正しい**.