

# ソフト系 C言語実習課題 3

## dump関数、dumpコマンドの作成

---

V4.23

# dump(ダンプ)とは

- dumpとは、メモリの内容、あるいはファイルの内容を下記のように、16進数で表示することを言います。

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
00000000	4701	001C	6B72	1175	D95B	585B	442D	A246	G...kr.u.[X[D-.F
00000010	6A03	844C	3EE7	1FC2	A8A2	78A8	7CD4	50BF	j..L>.....x...P.
00000020	4EB9	2046	0D28	9603	CF45	0C82	E709	B471	N. F. (...E.....q
00000030	1F5E	F450	BA12	079D	A959	E659	264C	ABAC	.^.P.....Y.Y&L..
00000040	1867	7725	D5DF	7A28	7F3A	285C	E363	2EF1	.gw%..z(.:(¥.c..
00000050	E4DD	A18B	D9B4	54DD	772C	226C	7811	A287	.....T.w,"lx...
00000060	F2A2	89A0	F54D	161C	8177	3E4D	0864	01D8	.....M...w>M.d..
00000070	3431	F7DF	0C0B	8923	FC51	430C	FF90	28A1	41.....#.QC...(.
00000080	EC18	374F	075C	E3DD	1236	C140	E36E	736D	..70.¥...6.@.nsm
00000090	7D84	0DB7	F450	B436	AE07	6E7B	DD40	8037	.....P.6..n..@.7
000000A0	BAA2	8660	C1EF	AE48	C5DC	E294	3990	1EC7	...`...H....9...
000000B0	5142	F019	C711	24F0	12B3	93AE			QB....\$. ....

- プログラムを作成しているとき、読み込んだデータの内容を見たいときがあります。そうしたときに、活躍するのがdump関数です。

# dumpコマンドでファイルの中を表示

- デジタルカメラで撮影した写真には、撮影時のさまざまな情報が入っています
  - その情報を、「Exif」(エグジフと発音)と言い、規格になっている
  - Exifとは、Exchangeable Image File Formatの略で、写真に埋め込んで保存する画像ファイルのフォーマットを定めている
- 下記は、実際に各社のカメラで撮影した写真をdumpしたものです
  - Exif情報は、通常JPEGファイルの先頭に入っているが、編集をすると消えたり、ファイルの後ろに追加される場合もある ← 編集ソフトの仕様による

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
000000A0	0000	0000	4170	706C	6500	6950	686F	6E65	.... Apple. iPhone
000000B0	2038	0000	0000	0048	0000	0001	0000	0048	8.... H..... H
000000C0	0000	0001	3132	2E31	0000	3230	3138	3A31	.... 12. 1.. 2018:1
000000D0	313A	3033	2031	323A	3437	3A32	3100	0020	1:03 12:47:21..
000000E0	829A	0005	0000	0001	0000	0246	829D	0005	z..... F...
000000F0	0000	0001	0000	024E	8822	0003	0000	0001	..... N. ".....

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
000000B0	2020	2020	2020	2020	2020	2020	2000	4361	. Ca
000000C0	6E6F	6E00	4361	6E6F	6E20	4958	5920	3136	non. Canon IXY 16
000000D0	3000	B400	0000	0100	0000	B400	0000	0100	0. I. . . . . I. . . .
000000E0	0000	3230	3138	3A31	313A	3033	2031	393A	. . 2018:11:03 19:
000000F0	3239	3A34	3000	2200	9A82	0500	0100	0000	29:40. " 嘯. . . . .
00000100	8802	0000	9D82	0500	0100	0000	9002	0000	. . . . 措. . . . .

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
000000A0	2020	2020	2020	2020	004E	494B	4F4E	0043	. NIKON. C
000000B0	4F4F	4C50	4958	2053	3730	3030	002C	0100	00LPIX S7000. . .
000000C0	0001	0000	002C	0100	0001	0000	0043	4F4F	. . . . . C00
000000D0	4C50	4958	2053	3730	3030	5631	2E31	0000	LPIX S7000V1. 1. .
000000E0	3230	3138	3A31	313A	3033	2031	393A	3530	2018:11:03 19:50
000000F0	3A31	3300	2400	9A82	0500	0100	0000	9E02	:13. \$. 嘯. . . . .

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
000000A0	0000	E401	0000	C823	0000	5061	6E61	736F	..... 袂#. Panaso
000000B0	6E69	6300	444D	432D	4758	3100	B400	0000	nic. DMC-GX1. 工...
000000C0	0100	0000	B400	0000	0100	0000	3230	3137	.... 工..... 2017
000000D0	3A31	313A	3032	2031	343A	3137	3A35	3700	:11:02 14:17:57.
000000E0	5072	696E	7449	4D00	3032	3530	0000	0E00	PrintIM. 0250....
000000F0	0100	1600	1600	0200	0000	0000	0300	6400	..... d.

# 8ビット(8単位)ASCII文字コード表

	上位	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
下位		0x	1x	2x	3x	4x	5x	6x	7x	8x	9x	Ax	Bx	Cx	Dx	Ex	Fx
0000	x0	NUL	DLE	SP	0	@	P	`	p				ー	タ	ミ		
0001	x1	SOH	DC1	!	1	A	Q	a	q			。	ア	チ	ム		
0010	x2	STX	DC2	"	2	B	R	b	r			「	イ	ツ	メ		
0011	x3	ETX	DC3	#	3	C	S	c	s			」	ウ	テ	モ		
0100	x4	EOT	DC4	\$	4	D	T	d	t			。	エ	ト	ヤ		
0101	x5	ENQ	NAK	%	5	E	U	e	u			・	オ	ナ	ユ		
0110	x6	ACK	SYN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
0111	x7	BEL	ETB	'	7	G	W	g	w			ア	キ	ヌ	ラ		
1000	x8	BS	CAN	(	8	H	X	h	x			イ	ク	ネ	リ		
1001	x9	HT	EM	)	9	I	Y	i	y			ウ	ケ	ノ	ル		
1010	xA	LF	EOF	*	:	J	Z	j	z			エ	コ	ハ	レ		
1011	xB	VT	ESC	+	;	K	[	k	{			オ	サ	ヒ	ロ		
1100	xC	FF	FS	,	<	L	¥	l				ヤ	シ	フ	ワ		
1101	xD	CR	GS	-	=	M	]	m	}			ユ	ス	ヘ	ン		
1110	xE	SO	RS	.	>	N	^	n	~			ヨ	セ	ホ	ゝ		
1111	xF	SI	US	/	?	O	_	o	DEL			ッ	ソ	マ	。		

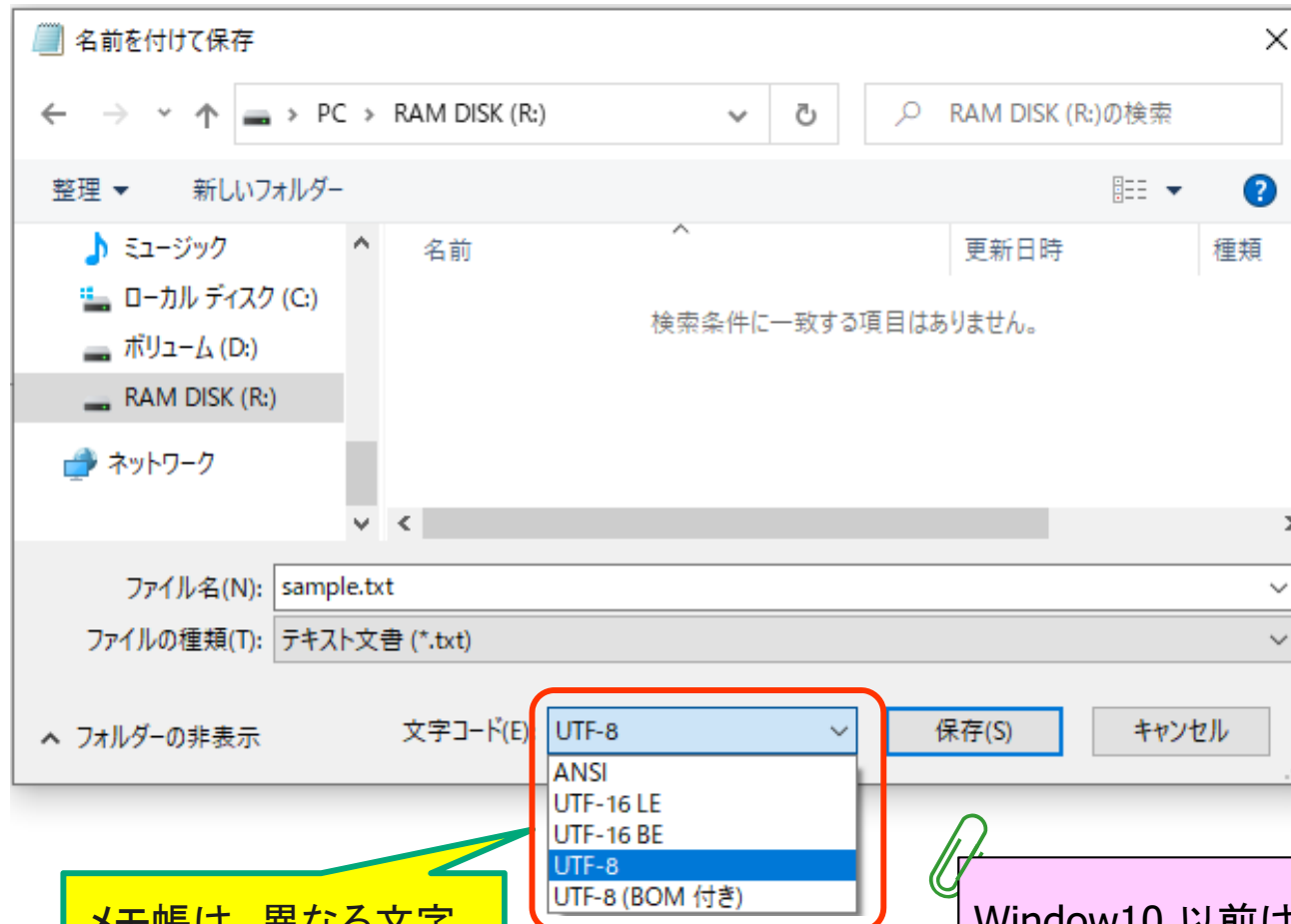
シリアル通信では、この部分のコードを、  
文字として転送できない

Shift-JISコードの第1バイト目のコード  
0x81~0x9F || 0xE0~0xEF

# 文字コードで異なるファイルの中身 1/2

ABCDEFGHIJKLMNOPQRSTUVWXYZ 123  
0 1 2 3 4 5 6 7 8 9 A B C D E F  
あいうえお かきくけこ

メモ帳で、左記のテキストを異なる文字コードで保存。  
表示結果は同じでも、ファイルの中身は異なる。



メモ帳は、異なる文字  
コードで保存ができる。

Window10 以前は、ANSI(Shift-JIS)が標準だったが、  
Windows10 1903版以降は、UTF-8が標準になった。

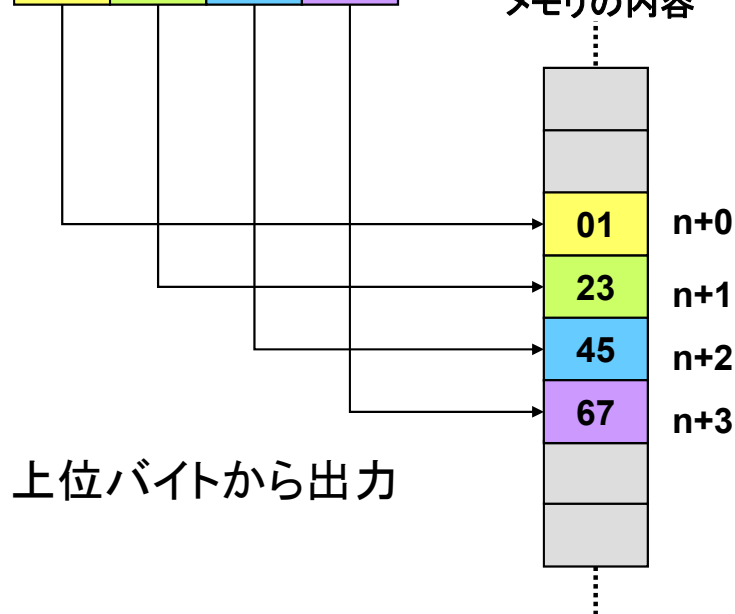
# ビッグエンディアンとリトルエンディアン

## ビッグエンディアンの動作

32ビットレジスタの内容

01	23	45	67
----	----	----	----

メモリの内容

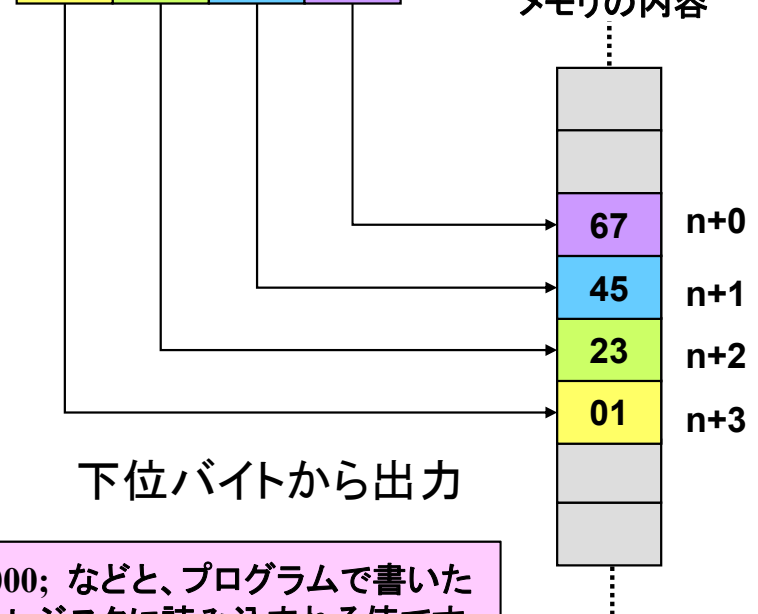


## リトルエンディアンの動作

32ビットレジスタの内容

01	23	45	67
----	----	----	----

メモリの内容

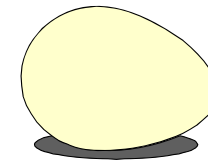


y = 0x1000; などと、プログラムで書いた数値は、レジスタに読み込まれる値です。

- ビッグエンディアンとリトルエンディアンはCPUの歴史
  - 世界初のマイクロプロセッサ4004はLittle Endian
- インテルは電卓の上位プロセッサとして発展
  - リトルエンディアン(Little Endian)でスタート
- モトローラの6800はミニコンを小さくするというコンセプトで開発
  - ビッグエンディアン(Big Endian)でスタート



ビッグ  
エンディアン



リトル  
エンディアン

# 文字コードで異なるファイルの中身 1/3

```

ABCDEFGHIJKLMN OPQRSTUVWXYZ 123
0 1 2 3 4 5 6 7 8 9 A B C D E F
あいうえお かきくけこ
    
```

メモ帳で、左記のテキストを異なる文字コードで保存。  
表示結果は同じでも、ファイルの中身は異なる。

ANSI(Shift-JIS : WIndow10以前のデフォルト)

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000000	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	A	B	C	D	E	F	1	2
00000010	51	52	53	54	55	56	57	58	59	5A	20	31	32	33	0D	0A	Q	R	S	T	U	V	W	X
00000020	82	4F	82	50	82	51	82	52	82	53	82	54	82	55	82	56	0	1	2	3	4	5	6	7
00000030	82	57	82	58	82	60	82	61	82	62	82	63	82	64	82	65	8	9	A	B	C	D	E	F
00000040	0D	0A	82	A0	82	A2	82	A4	82	A6	82	A8	81	40	82	A9	..	あ	い	う	え	お	か	
00000050	82	AB	82	AD	82	AF	82	B1									き	く	け	こ				

ANK(半角)は、1バイトに、  
漢字は2バイトになっている。

様々な文字コード(一部抜粋)

JIS	SJIS	EUC	UTF-8	UTF-16	文字	文字種
41	41	41	41	0041	A	半角(ASCII)
42	42	42	42	0042	B	半角(ASCII)
2641	83BF	A6C1	CEB1	03B1	α	全角
2642	83C0	A6C2	CEB2	03B2	β	全角
2341	8260	A3C1	EFBCA1	FF21	A	全角
2342	8261	A3C2	EFBCA2	FF22	B	全角
2422	82A0	A4A2	E38182	3042	あ	全角
2424	82A2	A4A4	E38184	3044	い	全角

## 文字コードで異なるファイルの中身 2/3

ABCDEFGHIJKLMNOPQRSTUVWXYZ 123  
0 1 2 3 4 5 6 7 8 9 A B C D E F  
あいうえお かきくけこ

メモ帳で、左記のテキストを異なる文字コードで保存。  
表示結果は同じでも、ファイルの中身は異なる。

## UTF-8 (Windows10のデフォルト)

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
00000000	4142	4344	4546	4748	494A	4B4C	4D4E	4F50	ABCDEFGHIJKLMNOP
00000010	5152	5354	5556	5758	595A	2031	3233	0D0A	QRSTUVWXYZ 123..
00000020	EFBC	90EF	BC91	EFBC	92EF	BC93	EFBC	94EF	・撰シ托シ抵シ難シ費
00000030	BC95	EFBC	96EF	BC97	EFBC	98EF	BC99	EFBC	シ包シ厄シ曆シ假シ呻シ
00000040	A1EF	BCA2	EFBC	A3EF	BCA4	EFBC	A5EF	BCA6	。・「・」・、・・・・ヲ
00000050	0D0A	E381	82E3	8184	E381	86E3	8188	E381	.. 纏ゆ> 纏・∴ 纏
00000060	8AE3	8080	E381	8BE3	818D	E381	8FE3	8191	翫.. 纏九” 纏上 ㇿ
00000070	E381	93							纏凍

ANK(半角)は、1バイトに、  
漢字は2バイトになっている。

UTF-8(BOM付)

UTF8では、先頭の3バイト(BOM: Byte Order Mark)でビッグエンディアンかリトルエンディアンを判断

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
00000000	EFBB	BF41	4243	4445	4647	4849	4A4B	4C4D	・ ヲ ABCDEFGHIJKLM
00000010	4E4F	5051	5253	5455	5657	5859	5A20	3132	NOPQRSTUVWXYZ 12
00000020	330D	0AEF	BC90	EFBC	91EF	BC92	EFBC	93EF	3. . . 撰シ 托シ 抵シ 難
00000030	BC94	EFBC	95EF	BC96	EFBC	97EF	BC98	EFBC	シ 費シ 包シ 厄シ 曆シ 假シ
00000040	99EF	BCA1	EFBC	A2EF	BCA3	EFBC	A4EF	BCA5	呻シ。・「・」・、・
00000050	EFBC	A60D	0AE3	8182	E381	84E3	8186	E381	・ヲ. . 纏ゆ>纏. . .
00000060	88E3	818A	E380	80E3	818B	E381	8DE3	818F	纏翫. . 纏九” 纏上
00000070	E381	91E3	8193						¢ 纏凍

文字コードは、1バイトから  
3バイトまで可変



# 文字コードで異なるファイルの中身 3/3

ABCDEFGHIJKLMNOPQRSTUVWXYZ 123  
0 1 2 3 4 5 6 7 8 9 A B C D E F  
あいうえお かきくけこ

メモ帳で、左記のテキストを異なる文字コードで保存。  
表示結果は同じでも、ファイルの中身は異なる。

## UTF16 ビッグエンディアン

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
00000000	FEFF	0041	0042	0043	0044	0045	0046	0047	. . . A. B. C. D. E. F. G
00000010	0048	0049	004A	004B	004C	004D	004E	004F	. H. I. J. K. L. M. N. O
00000020	0050	0051	0052	0053	0054	0055	0056	0057	. P. Q. R. S. T. U. V. W
00000030	0058	0059	005A	0020	0031	0032	0033	000D	. X. Y. Z. . 1. 2. 3. .
00000040	000A	FF10	FF11	FF12	FF13	FF14	FF15	FF16	. . . . .
00000050	FF17	FF18	FF19	FF21	FF22	FF23	FF24	FF25	. . . . . !. ". #. \$. %
00000060	FF26	000D	000A	3042	3044	3046	3048	304A	. &. . . . 0B0D0F0H0J
00000070	3000	304B	304D	304F	3051	3053			0. 0K0M000Q0S

すべての文字を16ビットで表現。  
バイトの並びがビッグエンディアン(先頭の2バイトで判断)

## UTF16 リトルエンディアン

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2	4 6	8 A	C E
00000000	FFFE	4100	4200	4300	4400	4500	4600	4700	..	A. B. C. D. E. F. G.		
00000010	4800	4900	4A00	4B00	4C00	4D00	4E00	4F00	H. I. J. K. L. M. N. O.			
00000020	5000	5100	5200	5300	5400	5500	5600	5700	P. Q. R. S. T. U. V. W.			
00000030	5800	5900	5A00	2000	3100	3200	3300	0D00	X. Y. Z. . 1. 2. 3. . .			
00000040	0A00	10FF	11FF	12FF	13FF	14FF	15FF	16FF	.....			
00000050	17FF	18FF	19FF	21FF	22FF	23FF	24FF	25FF	..... !. ". #. \$. %.			
00000060	26FF	0D00	0A00	4230	4430	4630	4830	4A30	& . . . . B O D O F O H O J O			
00000070	0030	4B30	4D30	4F30	5130	5330			. O K O M O O O Q O S O			

UTF16では、先頭の2バイトで、ビッグエンディアンかリトルエンディアンを判断

すべての文字を16ビットで表現。  
バイトの並びがリトルエンディアン(先頭の2バイトで判断)

# 本課題は、5つのSetpに分けて実習します。

- 実習を通じて、StepごとにC言語の基礎と下記のことを学びます。
  - Step0: dump関数を理解するため初学者向けです
    - ◆ 可能であれば、Setp0はスキップして、Step1からやってください
  - Step1:指定されたバイト数の数だけを16進数で表示
    - ◆ 対応する16進数が印字可能な文字コードであれば文字の表示
  - Step2:ファイルを開いてデータを読み込む
    - ◆ バイナリデータの読み込みになります
  - Step3:Shift-JISのコードと扱いについて学ぶ
    - ◆ 2バイトのShift-JISコードの泣き別れ時の処理
  - Step4:ファイルからの読み込みバイト数を16バイトずつにする
    - ◆ while()ループで、読み込みバイト数が0になるまでループする
  - Step5: mydumpコマンドの作成(コマンドプロンプトから直接実行できる形式)
    - ◆ コマンドプロンプトから実行できるコマンドを作成する
    - ◆ 自分独自のデータ型(構造体)の定義
    - ◆ 構造体への代入方法、“.”と”->”(アロー演算子)の違い
    - ◆ コマンドパラメータ(引数)の解析方法、および Visual Studio でのデバッグ方法
    - ◆ リダイレクト機能
    - ◆ 環境変数とPath

Step5は、新たにプロジェクトを作成してから行う

## Step0 : dump\_0でdumpやりの方を学ぶ

---

dump\_0は必ずしも作成する必要はありません。  
どのようにするかが分かるは、Step0をスキップして、Step1から始めてください。

# Step0: dump\_0関数のStep-1の仕様と実行結果

```
dump_0 (staddr, dsize); // dump関数の呼び出し
```

staddr = ダンプをするバッファへの符号なしchar型ポインタ

dsize = ダンプをするバイト数(int型)

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27	28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37	38	39	3A	3B	3C	3D	3E	3F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F
50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F
60	61	62	63	64	65	66	67	68	69	6A	6B	6C	6D	6E	6F
70	71	72	73	74	75	76	77	78	79	7A	7B	7C	7D	7E	7F
80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
90	91	92	93	94	95	96	97	98	99	9A	9B	9C	9D	9E	9F
A0	A1	A2	A3	A4	A5	A6	A7	A8	A9	AA	AB	AC	AD	AE	AF
B0	B1	B2	B3	B4	B5	B6	B7	B8	B9	BA	BB	BC	BD	BE	BF
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
D0	D1	D2	D3	D4	D5	D6	D7	D8	D9	DA	DB	DC	DD	DE	DF
E0	E1	E2	E3	E4	E5	E6	E7	E8	E9	EA	EB	EC	ED	EE	EF
F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	FA	FB	FC	FD	FE	FF

dump\_0 関数の実行結果

# Step0: dump\_0のmain関数

mainは、下記をコピーして使用する

```
#include <stdio.h>

#define TBLSIZE 256      // テスト用バッファの大きさ
#define COLSIZE 16      // 1列の桁数

// プロトタイプ宣言
void dump_0 (unsigned char staddr, int dsize);

int main(void) {
    unsigned char tbl[TBLSIZE];

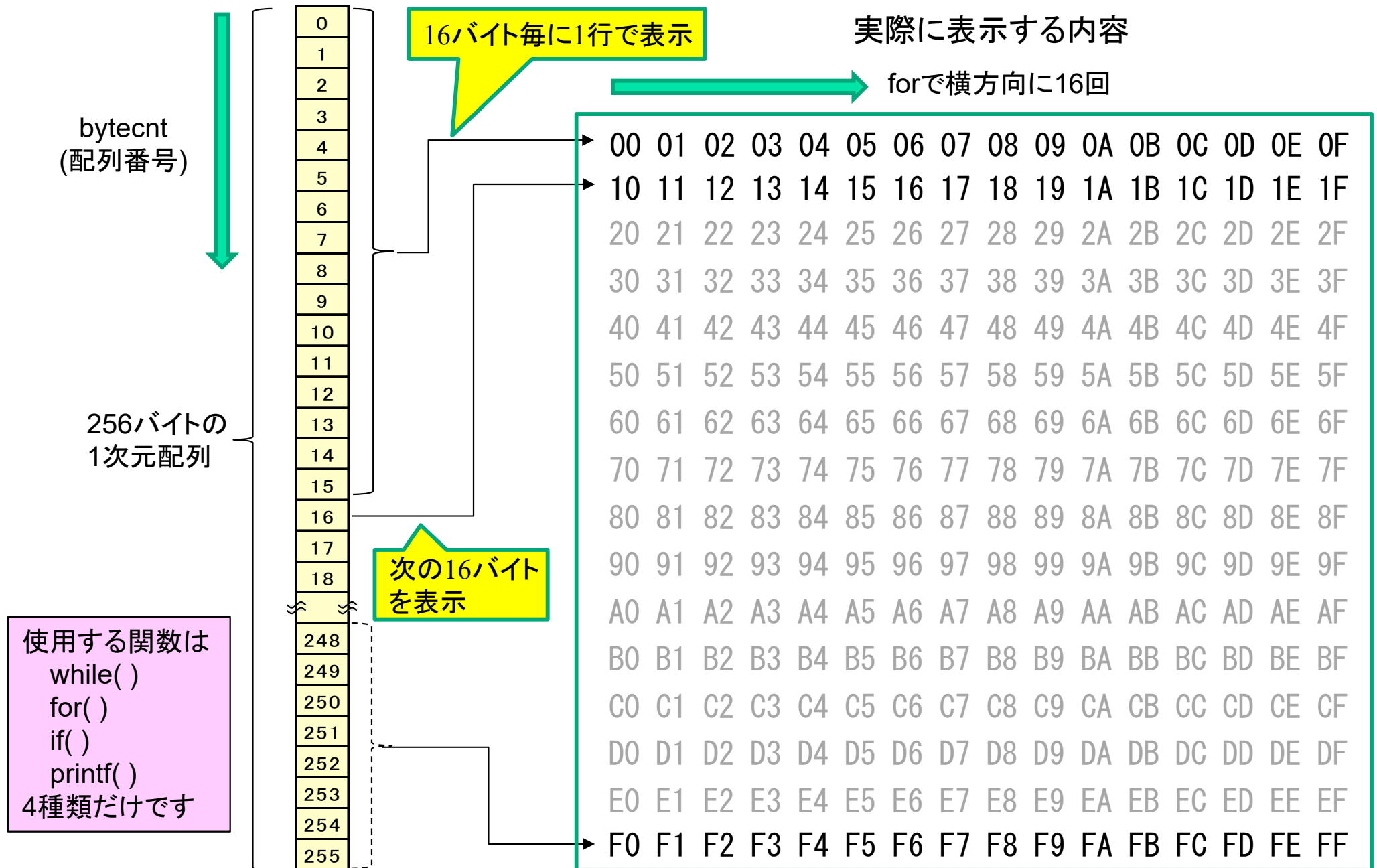
    for (int cnt = 0; cnt < TBLSIZE; cnt++) {
        tbl[cnt] = cnt;      // 一次元配列にデータを詰める
    }

    printf("256文字の表示\n");
    dump_0(tbl, 256); // dump関数の呼び出し
    printf("100文字の表示\n");
    dump_0(tbl, 100); // dump関数の呼び出し

    return 0;
}
```

作成するdump\_0 関数

# Step0: 表示のイメージ



# Step0: dump\_0 のフローチャート

void dump\_0(char tbl[ ], int dsize)

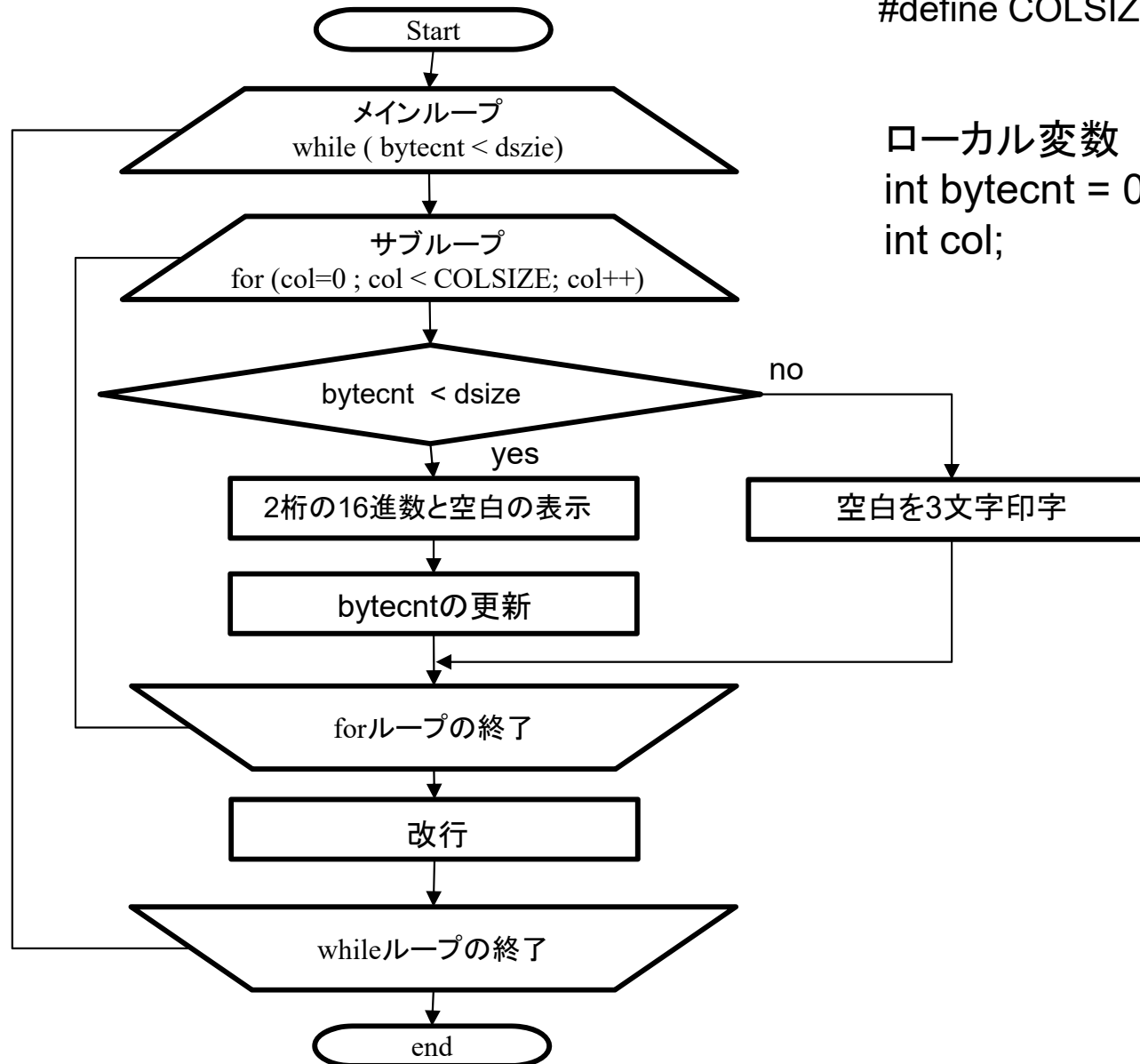
定数定義

#define COLSIZE 16

ローカル変数

int bytecnt = 0;

int col;



# Step1 : dumpの基本体系を作成する

---



# Step1 : dump関数の仕様と実行結果

```
void dump(char *title, unsigned char *staddr, int offset, int dsize, char opt)
```

char \*title = タイトル文字列へのポインタ

unsigned char \*staddr = ダンプをするバッファへのポインタ

int offset = 与えられたバッファポインタからオフセット(実際のダンプ開始位置)

int dsize = ダンプをするバイト数

char opt = オプション (ヘッダ表示のオン/オフ、文字列表示のオン/オフ)

Title ← タイトル文字列

Addr	0 1	2 3	4 5	6 7	8 9	A B	C D	E F	0 2 4 6 8 A C E
00000000	4701	001C	6B72	1175	D95B	585B	442D	A246	G...kr.u.[X[D-.F
00000010	6A03	844C	3EE7	1FC2	A8A2	78A8	7CD4	50BF	j..L>.....x...P.
00000020	4EB9	2046	0D28	9603	CF45	0C82	E709	B471	N. F. (...E.....q
00000030	1F5E	F450	BA12	079D	A959	E659	264C	ABAC	.^.P.....Y.Y&L..
00000040	1867	7725	D5DF	7A28	7F3A	285C	E363	2EF1	.gw%...z(.:(¥.c..
00000050	E4DD	A18B	D9B4	54DD	772C	226C	7811	A287	.....T.w,"lx...
00000060	F2A2	89A0	F54D	161C	8177	3E4D	0864	01D8	.....M...w>M.d..
00000070	3431	F7DF	0C0B	8923	FC51	430C	FF90	28A1	41.....#.QC... (.
00000080	EC18	374F	075C	E3DD	1236	C140	E36E	736D	..70.¥...6.@.nsm
00000090	7D84	0DB7	F450	B436	AE07	6E7B	DD40	8037	.....P.6..n..@.7
000000A0	BAA2	8660	C1EF	AE48	C5DC	E294	3990	1EC7	...`...H....9...
000000B0	5142	F019	C711	24F0	12B3	93AE			QB....\$.....

ヘッダ表示

アドレス表示      16進数表示      文字表示

# Step1: dump関数の作成条件と引数の説明

dump関数を使うために、テスト用として、main() を作成。

main()関数の中で、ダンプする文字列を定義して、下記のパラメータに従ってdump関数を呼出す。

```
void dump(char *title, unsigned char *staddr, int offset, int dsize, char opt)
```

char \*title

0x????????

“タイトル文字へのポインタです。”

unsigned char \*staddr

0x????????

ダンプをするバッファ

int offset

整数の値

staddr + offset  
(実際の開始アドレス)

int dsize

整数の値

dsize

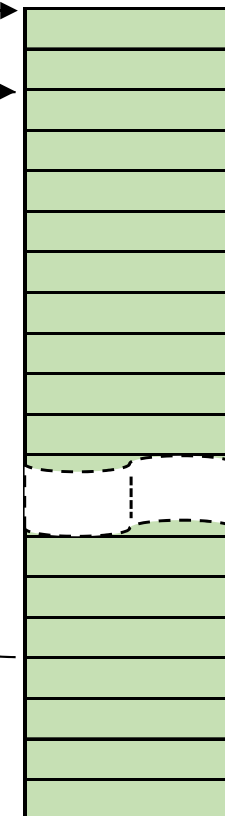
(このサイズ分だけダンプ)

char opt

整数の値

ビット操作で行う

opt=0x1(bit0)= 1 → 文字の印字をする  
opt=0x1(bit0)= 0 → 文字の印字をしない  
opt=0x2(bit1)= 1 → 2回目以降のヘッダの印字をしない  
opt=0x2(bit1)= 0 → 16行毎にヘッダの印字をする



# Step1 : ダンプの基本体系を作成する

## ■ Step1で学ぶこと

- 指定されたバイト数の数だけ16進数で表示
- 対応する16進数が印字可能な文字コードであれば文字の表示
  - ◆ 印字できない文字の場合は、“.” を出力する
- dump関数を呼び出すために、main()を作成し、その中でダミーデータを定義する。

### main関数の参考例

```
#define H_PRT 0x02    // ヘッダ印字オプション
#define C_PRT 0x01    // 文字印字オプション

int main(void) {
    char bin_data[512];           // テストデータ用のバッファ
    char asc_data[] = "01234567809 ABCあいうえおかきくけこDEFGHIJKLMNOPQRSTUVWXYZ 漢字表示のテスト
abcdefghijklmnopqrstuvwxyz 01234567809ABCDEFGHIJKLMNOPQRSTUVWXYZ abcdefghijklmnopqrstuvwxyz";

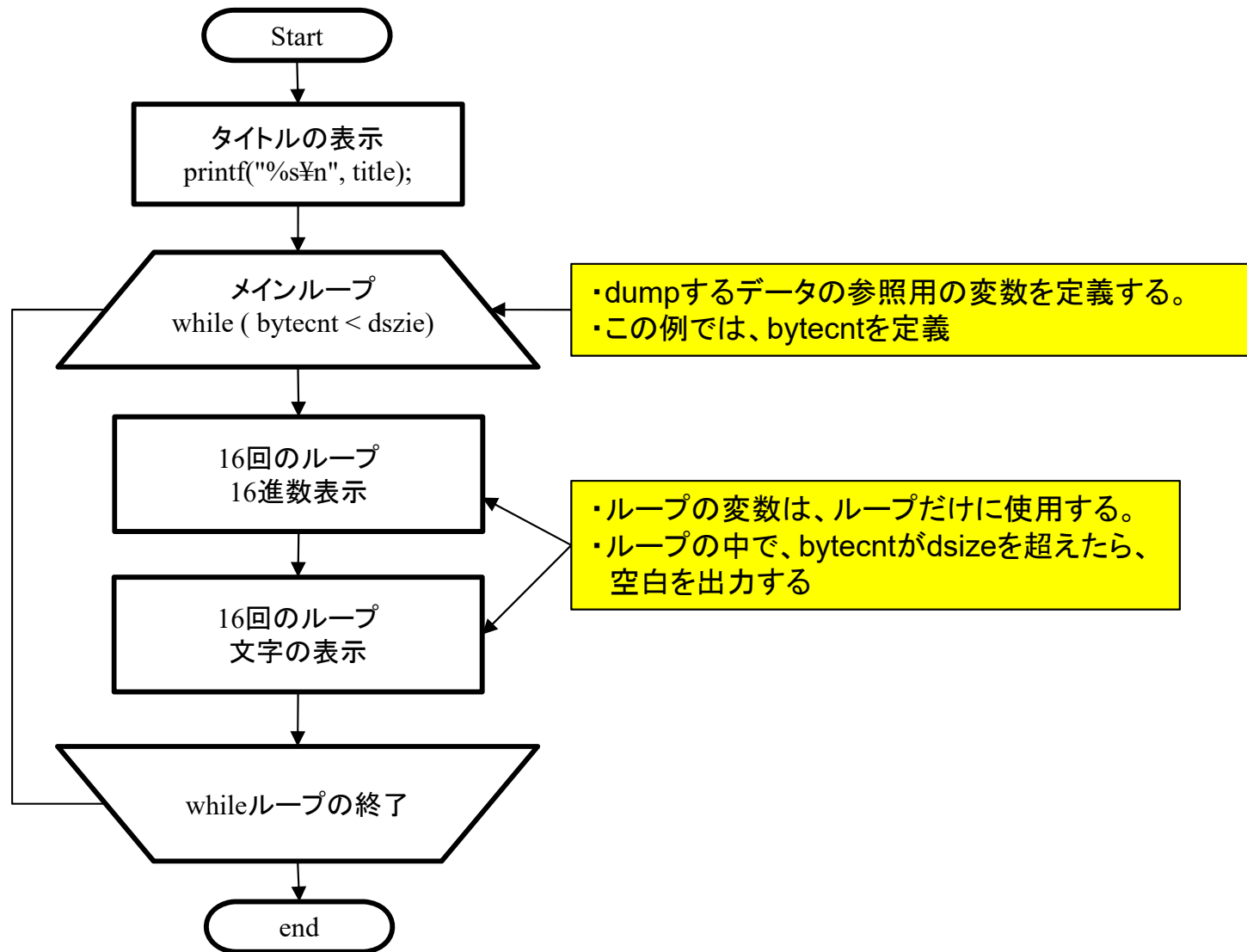
    for (int filcnt = 0; filcnt<512; filcnt++)
        bin_data[filcnt] = filcnt & 0xff; // テスト用のテーブルを0からFFで埋める

    dump("ASCII data 文字印字あり", asc_data, 0, 65, H_PRT + C_PRT);
    dump("BINARY data 文字印字あり", bin_data, 0, 512, C_PRT);
    return 0;
}
```

PDFをコピーすると、ここに改行コードが入ります。貼り付けたあとに削除をしてください。

確認用にdumpするサイズは16の倍数 +1にする

# 基本フローチャート



## Step2 : ファイルから読み込んで出力する

---

# Step2 : ファイルから読み込んで出力する

---

## ■ Step2で学ぶこと

- fopen()でファイルを開く ← バイナリモードでオープン
  - ◆#pragma warning(disable:4996) ← エラー回避のため、先頭に入れる
- データを読み込むため、1024バイトのバッファを定義し、そこに読み込む
  - ◆Step5のdumpコマンドでは、バッファサイズを16バイトで定義します
  - ◆Step2では、バッファサイズを1024バイトにして、1回で読み込みます
- ファイルからデータを読み込み
  - ◆バイナリデータの読み込みになるので、fread( )を使う
- 開いたファイルのクローズ

# ファイルの操作

## ■ ファイルの操作は下記の4つの操作が必要

- (1) FILE型でファイルポインタを定義する(オープンした時の情報が入る)
- (2) ファイルを開く(オープン) → 成功するとファイルポインタに情報が入る
- (3) ファイルポインタに対して、ファイルの操作(読み込み/書き込みなど)を行う
- (4) ファイルを閉じる(クローズ) → ファイルポインタを閉じる

## ■ ファイルポインタとはオープンされたファイルの情報が入っている

- FILE型のポインタとして定義 → 例: FILE \*fp;
- FILE型には下記の情報が含まれているが、現在のVisual Studioでは見えない。

```
struct _iobuf {  
    char *_ptr;  
    int  _cnt;  
    char *_base;  
    int  _flag;  
    int  _file;  ← システムのファイル番号(0,1,2は標準入出力のため3から始まる)  
    int  _charbuf;  
    int  _bufsiz;  
    char *_tmpfname;  
};  
typedef struct _iobuf FILE;
```

# ファイルのオープン

## ■ ファイルのオープンは、fopen( ) あるいは fopen\_s( ) を使う

### – fopen()の使用例

```
FILE *fp; // ファイルポインタの定義
```

```
fp = fopen( fileName , mode); // ファイルのオープン
```

オープンに成功すると、fpにファイル情報のポインタが入る  
オープンに失敗すると、fpにNULLが返される

### – C言語では、if文の中に入れるとオープンの成功/失敗が判断できる

```
if ( ( fp = fopen("s-jis2.txt", "rb") ) != NULL )  
    // オープン成功  
else  
    // オープン失敗
```

dumpは、バイナリデータを読むために、“b”が必要



# fopenのmode指定

## ■ ファイル名がパスリスト(ドライブ名やフォルダ名がある)場合

– 区切り記号は、“¥¥”にする。

```
char *filename = "C:¥¥Users¥¥admin¥¥source¥¥repos¥¥test¥¥test¥¥source.c";  
fp = fopen ( filename, "r"); ← dumpでは、“rb”にする
```

## ■ fopenのモードとは、ファイルをどのように開くかを文字列で指定

– 基本は、次の3のモード

モード	機能 と 開始位置	ファイルがないとき
"r"	読み取り(ファイルの先頭から)	エラー
"w"	書き込み(ファイルの先頭から)	新規作成
"a"	追加書き込み(ファイルの最後から)	新規作成

– 基本モードに対して、次のファイル形式と動作を指定

モード	ファイルと種類と動作
"b"	バイナリ形式("b"がないとテキスト形式になる)
"+"	更新モードの指定

– 実際の指定は、上記の組合わせで指定する

- ◆ “rb” バイナリデータの読み込み
- ◆ “w+” テキストデータの追加書き込み

# ファイルのRead/Write操作

---

## ■ 代表的なファイルの読み込み関数

- fgetc()            1文字の読み込み
- fgets()            1行の読み込み(¥n)まで読む
- fread()            バイナリの読み込み ← dumpで使用
- fscanf()           書式指定の読み込み

## ■ 代表的なファイルの書き込み関数

- fputc()            1文字の書き込み
- fputs()            1行の書き込み
- fwrite()           バイナリの書き込み
- fprintf()           書式指定で書き込み

Step3 : 表示文字をShift-JISで表示する。

---

# Step3 : 表示文字をShift-JISで表示する。

## ■ Step3で学ぶこと

- Shift-JISのコードと扱いについて学ぶ
- 出力する文字が Shift-JISコードの第1バイトであれば、その文字と次の文字とを合わせて、2バイトをまとめて出力する。
  - ◆ 2バイトまとめて出力とは、書式を”%c%c”で、現在の文字と次の文字を出力
  - ◆ 2バイトまとめて出力した場合、bytecntとforループの変数の更新(+1)も忘れずに
- 16列目が、Shift-JISコードの第1バイトだった場合、その文字を保留して、表示も改行もせずに戻る。
- そして、次の行の最初に、保留した文字と次の行の最初の1バイトを合わせて出力し改行をする。
- Shift-JISのコード体系
  - ◆ 1バイト目に来るコード 0x81 ~ 0x9F || 0xE0 ~ 0xEF
  - ◆ 2バイト目に来るコード 0x40 ~ 0x7E || 0x80 ~ 0xFC

今回は、2バイト目の判定は省略して良い。

# Shift-JISコードの特徴

- Shift-JISは、2バイトで漢字を表し、半角文字との共存ができるために、マイクロソフトで古くから利用されてきた。
- しかし、漢字コードの途中から、そのコードが、1バイト目か、2バイト目かの判定ができない。なぜなら、1バイト目と2バイト目で、同じコードが使われているからである。
- そのため、Shift-JISの漢字コードを調べるには、文字列の最初から調べていくしかない。
- また、Shift-JISでは、2バイト目に、”0x5C”(‘¥’)のコードも使われている。
  - 今回は、直接関係ないがC言語でパスリストを指定するときは、これが厄介となる。
  - パスリストの区切り記号を、”¥¥”にしなければならない。
- Shift-JISの文字コードであるかの判定は、第1バイト目を調べ下記の条件で判定する。
  - 0x00～0x1F : JIS X 0201 制御コード
  - 0x00～0x7E : JIS X 0201 英数文字コード
  - 0x81～0x9F と 0xE0～0xEF : シフトJIS漢字の第1～第3水準の1バイト目となる
  - 0xA0～0xDF : JIS X 0201 カタカナ文字(通称半角カナ)
  - 0x40～0x7E と 0x80～0xFC : シフトJIS漢字の第1～第3水準の2バイト目となる
- その他、Shift-JISに関しては、Webに掲載されているので、そちらを参照のこと。

# Step3: Shift-JISの表示

Shift-JISは、半角文字と漢字コードが混在できる。

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000000	8E84	82BD	82BF	4A54	4543	2043	6F72	706F									私	た	ち	J	T	E	C	
00000010	7261	7469	6F6E	2082	CD81	4131	3939	3694									r	a	t	i	o	n		
00000020	4E38	8C8E	82CC	916E	8BC6	88C8	9788	8141									は	、	1	9	9	6	年	
00000030	93FA	967B	82AA	8E9D	82C2	8175	8FA0	8176									8	月	の	創	業	以	来	、
00000040	82CC	90B8	905F	82F0	8C70	8FB3	82B5	8141									日	本	が	持	つ	「	匠	」
00000050	926D	8EAF	82C9	8AEE	82C3	82AD	926D	8C62									の	精	神	を	継	承	し	、
00000060	82F0	88B5	82A4	8175	8B5A	8F70	8FA4	8ED0									知	識	に	基	づ	く	知	識
00000070	76	82C6	82D5	82C4	8141	91BD	82AD	82CC									を	扱	う	「	技	術	商	社
00000080	8A46	976C	82C9														」	と	し	て	、	多	く	の
00000090	D8B0	BD90	EA96														お	取	引	先	の	皆	様	に
000000A0	6E20	436F	6D70														技	術	職	知	財	リ	ス	専
000000B0	6172	746E	6572														門	の	S	o	l	u	t	
000000C0	82C8	82AA	82E7														a	n	y	と	し	て	P	a
000000D0	94AD	9357	82B5	82C4	82DC	82A2	82E8	82DC									s	h	i	p	を	築	き	な
000000E0	82B5	82BD	8142	0D0A	93FA	967B	82F0	8EE6									が	ら						
000000F0	82E8	8AAA	82AD	8ED0	89EF	8AC2	8BAB	82CD									発	展	し	て	ま	い	り	ま
																	し	た	。	..	日	本	を	取
																	り	巻	く	社	会	環	境	は

行で保留されている文字(Shift-JISの1バイト目)あれば、保留されている文字と、次に表示する最初の文字0x4eを合わせて、0x944eを出力して改行する。前行の最後に"年"が表示され、カーソルここに来る。その後、普通に16進数表示を行う。

16バイト目がShift-JISの1バイト目なので、ここでは、0x94を保留とし、改行しはしないようにする。

前行で保留されている文字(Shift-JISの1バイト目)があれば、保留されている文字と、次に表示する最初の文字0x4eを合わせて、0x944eを出力して改行する。前行の最後に”年”が表示され、カーソルがここに来る。その後、普通に16進数表示を行う。

最初の4Eは、Shift-JISの第2バイトとして、前の行で表示をしているので、最初に空白を入れ、次の文字(38)から表示する。

泣き別れ時の処理として、第2バイトの先読みして表示(先読みをした分、ファイルポインタを1文字戻す)する方法と、第1バイトを保留しておき(改行をしない)、次の行を表示するときに、保留していた第1バイトと次の行の最初のバイトを第2バイト目として表示する方法がある。

ファイルポインタを戻すという方法は、ファイル操作でしか利用できず、パイプラインやストリームデータではできない。

そのため、本課題では、後者の方法で実装をしてください。

# Step4:ファイルからの読み込みを16バイトずつにする

## ■ Step4は、Step5の前哨戦となります

## ■ Step3からの変更点

### – (1)バッファサイズを16バイトにする

- ◆ fread() で1回で読んでいたファイルの読み込みを16バイトずつに変更します

➤ while() の中で、dsize=fread() を実行し、読込んだバイト数が0より大きい限りループをします

```
while ( dsize = fread() が0より大きい){  
    dump() 関数の呼び出し  
}
```

演算子の優先順位に注意!!

dsize = fread() > 0     ×  
(dsize = fread()) > 0     ○

### – (2)dump関数の変更

- ◆ (2)-1 printf (“%s¥n”, title); と staddr += offset; の2行を削除

➤ 16バイト毎に呼び出されるために、タイトル表示はしない

- ◆ (2)-2 ローカル変数は、関数から戻ると消えるので、3つの変数の前に、static宣言を付ける

➤ アドレス ➔ 例: static int addr;

➤ 行のカウント ➔ 例: static int linecnt;

➤ Shift-JISの1バイト目の保留データ ➔ 例: static unsigned char sjis1;

ローカル変数は、関数から戻ると消滅するが、static 宣言をした変数は、関数から戻っても値を保持される

# 変数定義時の指定子

## ■ 変数を定義する時、変数の型の前に指定子を書く場合がある

– 指定子には、次のようなものがある

**auto** 普通のローカル変数の指定で通常、autoは省略して書かない。  
関数内でのみ使用可能で、変数は関数から戻ると消滅する。  
自動的に消滅することからauto変数と言っている。

**register** 使用頻度が高い変数をメモリでなく、CPUのレジスタに割り当てる。他はautoと同じ。  
レジスタの少なくレジスタの割り当てができない場合は普通のauto変数となる。

**static** 関数から戻っても値が保持されている。関数の中でも外でも使用可。  
通常のauto変数(ローカル変数)は、関数から戻ると消滅するがstaticを指定すると  
関数から戻ってもその関数内で値が保持される。  
ただし、関数内で定義をした場合は、auto変数同様他の関数からは参照できない。  
関数の外でstaticを指定した場合、そのファイルにおいてグローバル変数となる。

**extern** グローバル変数の二重定義を防ぐため、定義された変数名の参照だけを行う。  
ソースコードが複数のファイルから構成されるとき、最初のファイルで実際の変数を  
定義を行い、他のファイルはextern を付けて名前だけ参照するようにする。

**typedef** 型に別名を付ける。構文の便宜上記憶クラス指定子に分類されている。  
構造体の定義時などによく使用される。



## Step5 : mydumpコマンドの作成

---

Step5は、新規にプロジェクトを作成し、  
Step4までのソースを再利用してください。  
一部変更はありますが、ほとんどそのまま使えます。

# Step5 : mydumpコマンドを作成する

---

## ■ Step5で学ぶこと

- コマンドプロンプトから実行できるコマンドを作成する
  - ◆ コマンド引数の解析
- コマンド引数(パラメータ)の解析方法
  - ◆ switch – case 文を使う
  - ◆ Visual Studioで、デバッグをするために引数を設定する
- 構造体の作成 (自分独自のデータ型の定義)
  - ◆ 構造体へのアクセス方法、“.”と“->”(アロー演算子)の違い
- 標準入出力とリダイレクト機能
  - ◆ そのため、コマンドでは、バッファサイズは、16バイトにします
- 環境変数とPath
  - ◆ 作成したプログラムを環境変数のPathを追加してWindowsに登録する
  - ◆ Windowsの環境変数の設定方法

# コマンド引数と argc, argvとの関係

mydump infilename /h /c >outfilename

①

②

③

④

⑤

標準出力のリダイレクト指定

```
main( int argc, char *argv[ ])
```

argcは、コマンド名を含む引数の数  
argvは、各引数へのポインタの配列

コマンドプロンプト(Shell)がリダイレクトでファイルを生成するので、引数としては入って来ない。

argcとargvの関係を調べるために、mainの最初に下記のプログラムを実行すると、argcの番号と実際の引数が表示されます。

```
for (int cnt = 0; cnt < argc; cnt++) {  
    printf("argc=%d %s¥n", cnt, argv[cnt]);  
}
```

argv →

[0]	mydump
[1]	infilename
[2]	/h
[3]	/c

# Step5 : mydumpコマンドのヘルプメッセージ例

## ■これは mydump /? で表示されるヘルプメッセージの例です

C:\Users\¥m-hoshi>mydump /?

mydump コマンド

構文: mydump [<opts>] [<inpath>] [<opts>]

機能: ファイルやデバイスの内容を16進で表示する

オプション:

/o[=]<hex> オフセットアドレスの指定(省略の場合は0000)

/p[=]<dec> 指定の行数を表示したら一時停止(省略の場合は一時停止なし) ← 追加機能

/a アドレスを物理アドレス表示(デフォルトは相対表示) ← 追加機能

/c ダンプのあとに文字を表示(デフォルトは表示)

/h 16行毎にヘッダーの出力(デフォルトは表示)

/? 使用方法の表示(このメッセージを表示)

構文で、“[]”は省略可能なパラメータを意味する。

16進文字列は、`strtol()`でバイナリに変換できる。

- (1) <inpath>が与えられたときは、ファイルをオープンして、その内容を読みだしdumpする。
- (2) オプション文字は、大文字、小文字は区別しません。  
「o」、「p」オプション解析は、文字の後に、「=」があってもなくてもどちらでも良いようにしてください。
- (3) オプションの種類は、データ構造で各フラグ類を定義をする。
- (4) 16進文字列の16進バイナリ変換は、`strtol()`を使うと良いです。
- (5) オフセットは、オープン直後に、数値が与えられていれば、`fseek()`関数で開始位置を移動します。
- (6) 追加機能は、dumpコマンドが完成してから追加する追加課題です。

# Step5 : mydumpで新たにプロジェクトを作成

## 作成した dump関数からの変更点

- (1) 新しくプロジェクトを作成する
- (2) dump関数のソースコードをコピーする。

当然、main関数のdump関数の呼び出し時の引数も削除

- (3) dump関数の引数から、titleとoffsetを削除

dump (~~char \*title,~~ unsigned char \*staddr, ~~int offset,~~ int dsize, char opt)



dump (unsigned char \*staddr, int dsize, opts\_s \*opts)

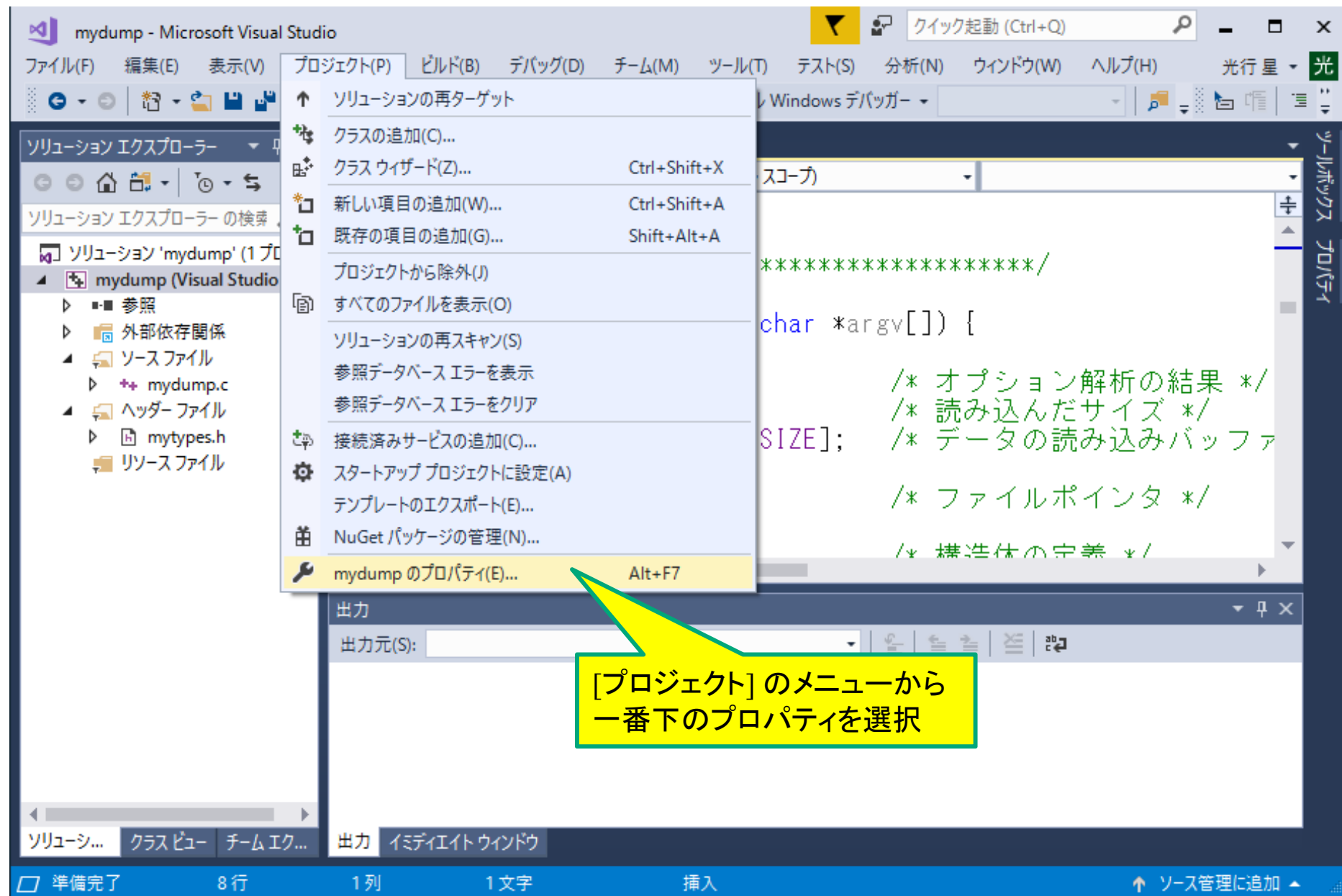
dump関数からコピーしたプログラムの動作確認ができたなら、構造体の opts\_s \*opts に変更する

- (4) デバッグ用に、Visual Studioでコマンド引数を設定

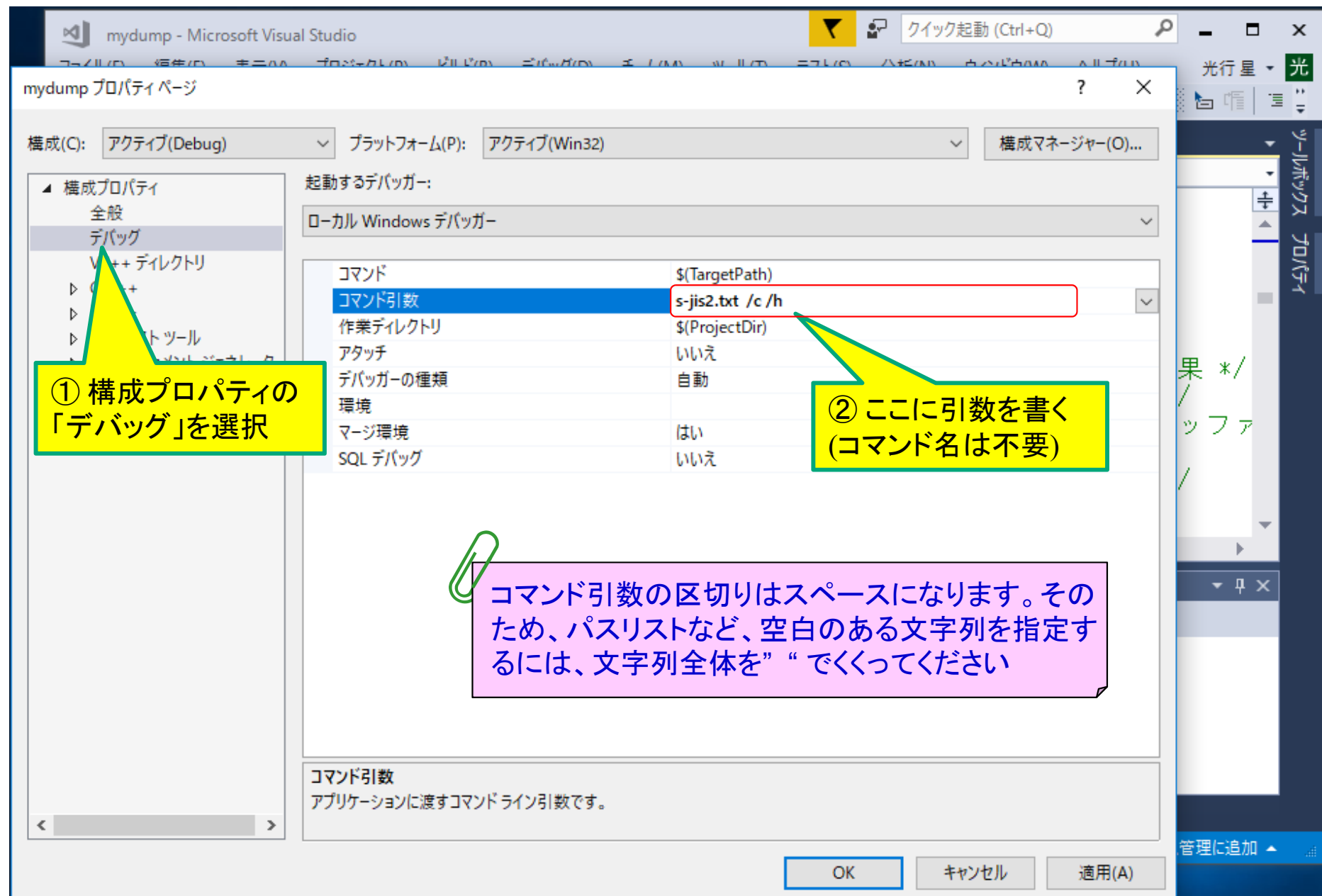
VisualStudioの[プロジェクト]→[プロパティ]

詳細は、次ページを参照のこと

# Visual Studioでコマンド引数を設定する 1/2



# Visual Studioでコマンド引数を設定する 2/2



# 構造体の話

---



# 構造体の書き方

- 構造体とは、複数異なるデータ型を一つのデータ型として扱う

## 構造体の書き方(宣言)

```
struct タグ名 {  
    データ型    メンバー名;  
    データ型    メンバー名;  
    :  
    データ型    メンバー名;  
};
```

## 実際の宣言例

```
struct mydata {  
    int    count;  
    int    age;  
    char *filename;  
    char name[10];  
};
```

構造体の宣言だけで、実際のメモリは割当てられていない

ここで、mydata という型で実際のメモリが割り当てられる

```
struct mydata data1;  
struct mydata data2[10];
```

# typedef で新しい型を定義

- typedef とは、自分で新しいデータ型を定義すること
  - `#include <stdint.h>` を入れると下記のタイプが使用できる

<code>typedef signed char</code>	<code>int8_t;</code>
<code>typedef short</code>	<code>int16_t;</code>
<code>typedef int</code>	<code>int32_t;</code>
<code>typedef long long</code>	<code>int64_t;</code>
<code>typedef unsigned char</code>	<code>uint8_t;</code>
<code>typedef unsigned short</code>	<code>uint16_t;</code>
<code>typedef unsigned int</code>	<code>uint32_t;</code>
<code>typedef unsigned long long</code>	<code>uint64_t;</code>

元の型                      新たに定義された型

stdint.h で定義  
されている

- 同様に構造体の名前を定義する

```
typedef struct mydata {  
    int    count;  
    int    age;  
    char *filename;  
    char name[10];  
} mydate_t;
```

typedefで定義された新しい型の名前

# typedef で構造体を宣言

```
struct mydata {  
    int    count;  
    int    age;  
    char  *filename;  
    char  name[10];  
};
```

構造体の宣言

```
typedef struct mydata {  
    int    count;  
    int    age;  
    char  *filename;  
    char  name[10];  
} mydata_t;
```

構造体をtypedefで宣言する

Typedefを使う場合は、タグ名は省略できる。

typedefで宣言された新しい型の名前

## 楮遺体の定義

```
mydata_t data1;
```

→ data1.age = 20;

```
mydata_t data2[10];
```

→ data2[1].age = 20;

# 構造体へのアクセス方法

```
typedef struct mydata { // 構造体の宣言
    int    count;
    int    age;
    char *filename;
    char name[10];
} mydata_t;
```

```
int main(void){
    mydata_t myinfo; // 構造体の定義
```

```
    myinfo.count = 0;
    myinfo.age = 0; }
```

実態がある場合は、"." でアクセス

```
    func(&myinfo);
    return 0;
}
```

構造体の実態は、mainにある。

& を付けて変数のアドレスを引数にする。

```
Void func(mydata_t *data){
```

```
    data->count = 0;
    data->age = 0; }
```

実態がない場合は、"->"(アロー演算子) でアクセス

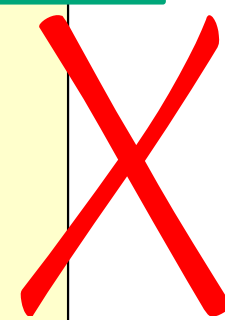
構造体は、ポインタで受けていて、実態はない

# 構造体の初期化

## ■ 間違った初期化

```
struct mydata {  
    int    count = 0;  
    int    age  = 0;  
    char *filename = NULL;  
    char name[10] = "myname";  
};
```

ここは、単なる構造体の宣言で、実態がないため、このような初期化はできない。



## ■ 定義時に初期化する

```
struct mydata data = { 0, 0, NULL, "myname" };
```

## ■ 定義後に個別に初期化する

```
data.count = 0;  
data.age = 0;  
data.filename = NULL;  
data.name = "myname";
```

ここで、実際の構造体を作成されるため、このような初期化は可能

# Step4: オプション解析関数の構成

## ■ コマンドプロンプトから渡されたコマンドの引数の解析を行う

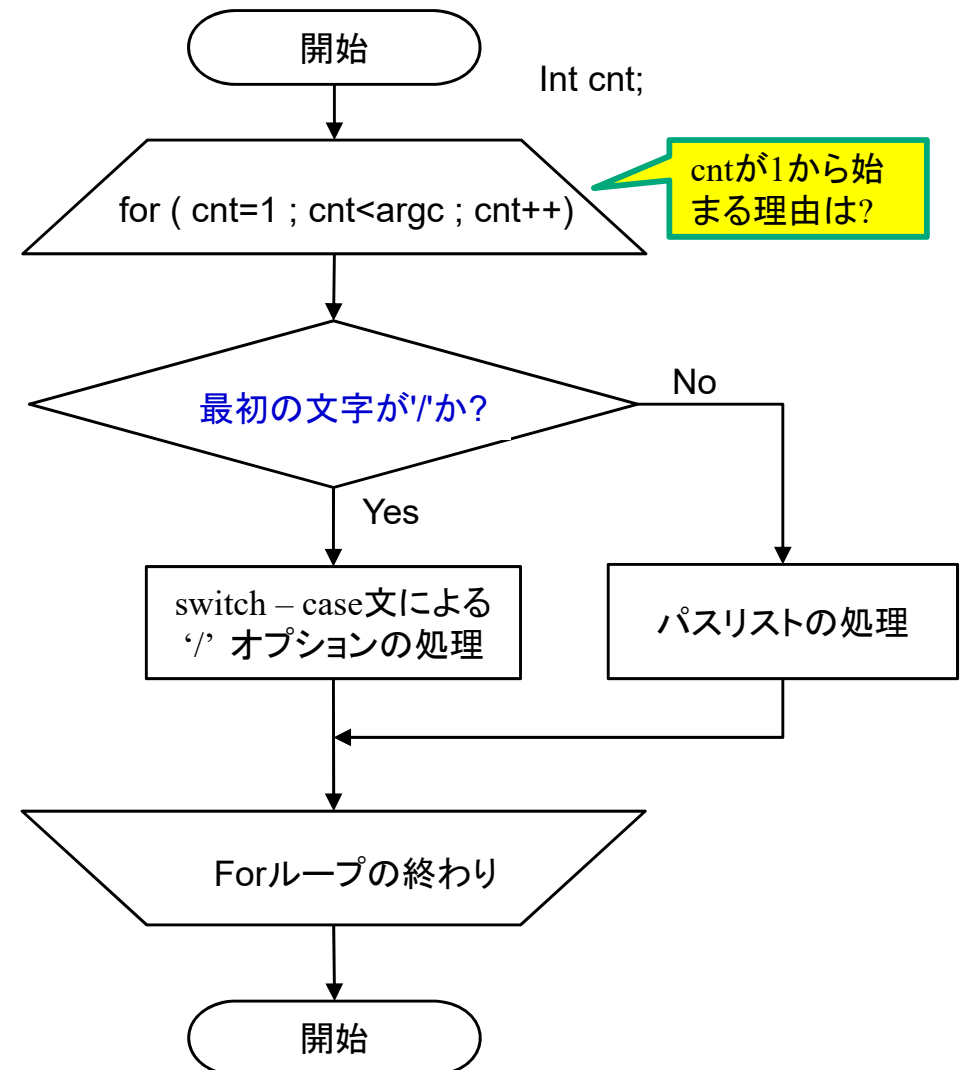
オプション類は、下記のデータ構造を宣言

```
typedef struct {  
    char *infileName;    // ファイル名  
    int  offset;         // オフセット  
    char prt_charflag;   // 文字の表示  
    char prt_headerflag; // ヘッダの表示  
    char prt_usageflag;  // ヘルプ表示  
} opts_t;
```

main()の中で、下記のようにデータ構造を確保して、やってみましょう。

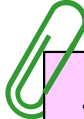
```
int main(int argc, char * argv[]) {  
  
    int  result; // オプション解析の結果  
    opts_t opts; // データ構造の定義  
  
    result = opts_analysys(argc, argv, &opts);  
    if(result != 0 || opts.prt_usageflag) {  
        resultを調べてエラー表示とヘルプを表示  
    }  
}
```

opts\_analysysのフロー



# 構造体使用に伴うオプション処理の変更点

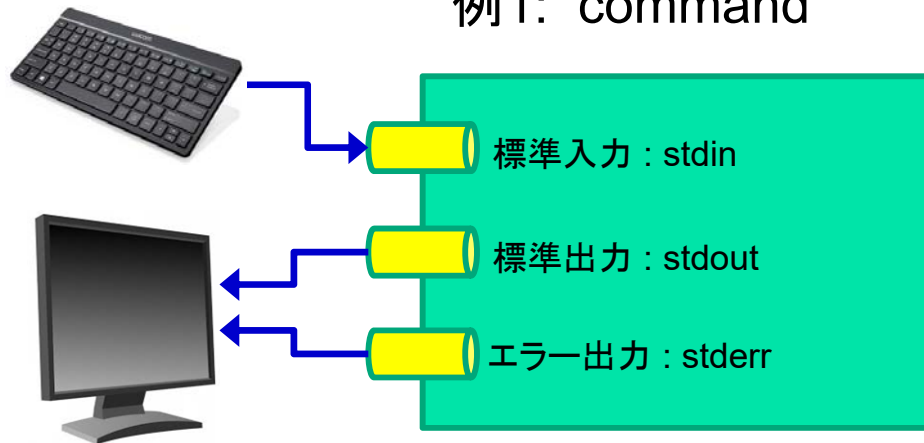
- dump関数では、メモリ節約でオプション指定をビット操作にしていた
  - `opt & H_PRT`
  - `opt & C_PRT`
- 構造体では、これらの変数をビット操作でなく、1バイトの変数として指定
  - `opt & C_PRT` → `prt_charflag`;      // 文字の表示
  - `opt & H_PRT` → `prt_headerflag`;      // ヘッダの表示
- 実際の変更例
  - `if(opt & C_PRT)` → `if(opts->prt_charflag)`
  - `if(opt & H_PRT) == 0)` → `if(opts->prt_headerflag == 0)`



構造体を定義したからといって、必ずしも1バイトの変数にする必要はありません。  
dump関数と同じようにビット操作にしてもかまいません。  
その場合は、オプション解析ルーチンでも、オプションの設定をビット操作にします。  
今回は、色々な手法を学ぶために、1バイト変数で使います。

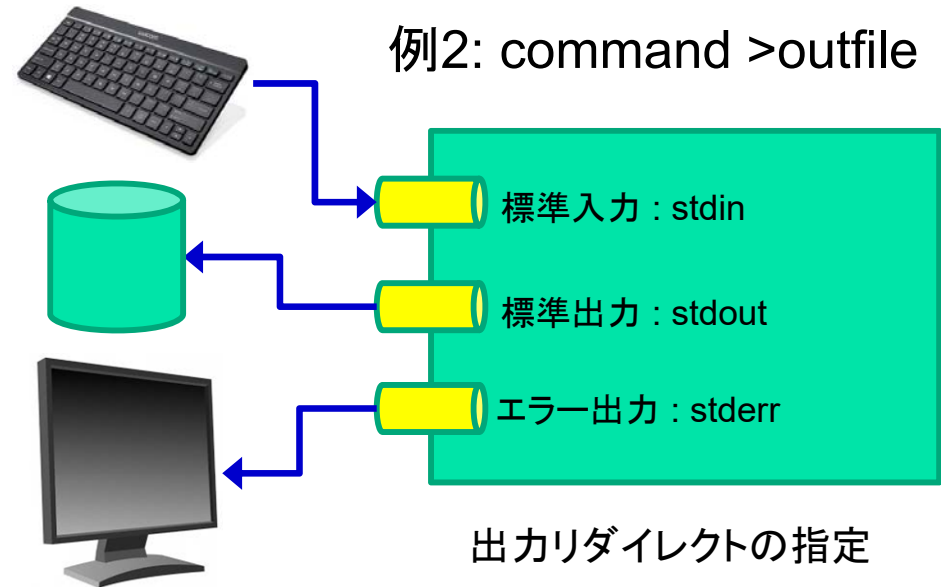
# 標準入出力とリダイレクトについて

例1: command



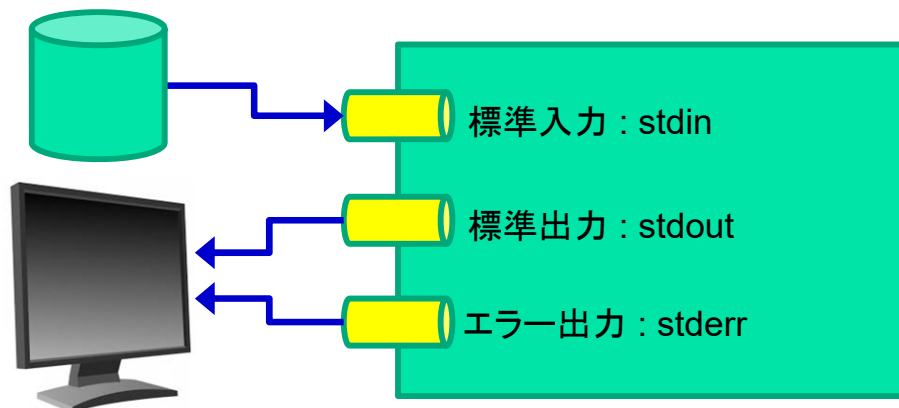
標準(リダイレクトなし)

例2: command >outfile



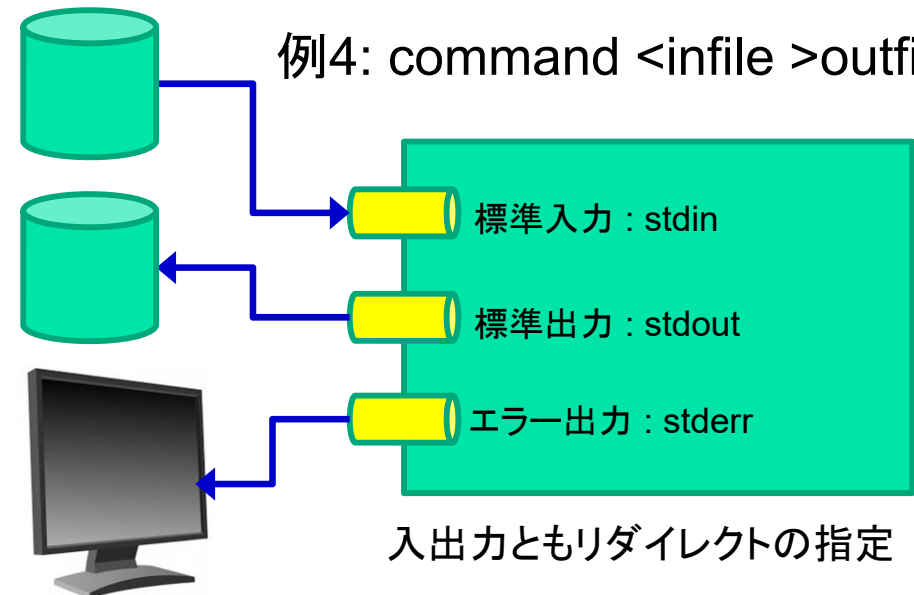
出力リダイレクトの指定

例3: command <infile



入力リダイレクトの指定

例4: command <infile >outfile

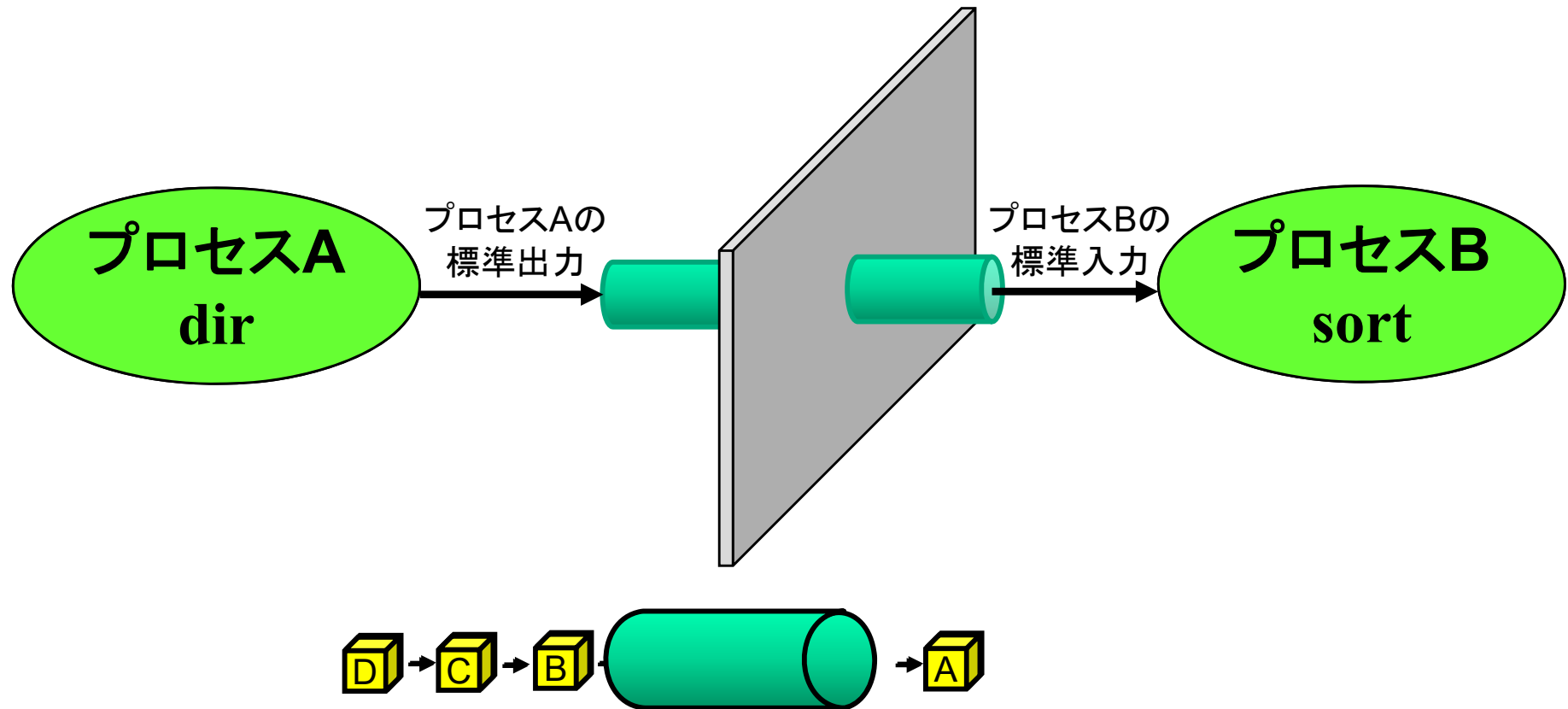


入出力ともリダイレクトの指定



# 参考: リダイレクトの機能によるパイプライン

プロセスAの標準出力がパイプを通して、プロセス標準入力につながる



リダイレクトとプロセス間通信の動作

# 3種類のprintf( )

- データの出力に最も多く使用するprintf( )は3種類ある

`printf ( "hello World¥n" );` → 普通のprintfで標準出力に出力  
`fprintf (fp, "hello World¥n");` → fpで指定したファイルに出力  
`sprintf (buf, "hello World¥n");` → 出力をbufに格納

- fprintf( )は、指定したファイルポインタに出力する

- 実際は、ファイルをオープンして、そのファイルポインタを与える
- ファイル名が与えられていなければ、fpはstdoutにする。

```
FILE *fp;  
If ( filename != 0)           ← ファイル名が与えられていれば、  
    fp = fopen(filename, "r"); ← ファイルをオープンする  
else  
    fp = stdout;              ← ファイル名がなければ、fpをstdoutにする  
fprintf(fp, "hello World¥n"); ← fprintf()で fpに出力
```

- sprintf( )は、指定したchar型の配列に書き込む

- 画面などに出力する内容を文字列として指定した配列に格納する
- 出力する文字列の長さなどを調べたりできるので便利です。

```
char buf[256];                ← 文字列を格納するバッファを確保  
sprintf(buf, "%d年%d月%d日", year, month, day);  
printf("%s¥n", buf);          ← %sでbufを出力する
```

# 3種類のscanf( )関数

- データの入力に最も多く使用するscanf( )もprintf( )同様3種類ある

<code>scanf (        "%d", &amp;intval);</code>	➔ 普通のscanfで標準入力から入力
<code>fscanf (fp,    "%d", &amp;intval);</code>	➔ fpで指定したファイルから入力
<code>sscanf (buf, "%d", &amp;intval);</code>	➔ 文字列(buf)から入力

- 標準入力(通常はキーボード)から入力

```
int intval;  
printf("数字を入力してください:");  
scanf (        "%d", &intval);
```

- ファイルから入力

```
FILE *fp;  
int intval;  
fp = fopen("infile","r");  
fscanf (fp,    "%d", &intval);
```

- 文字列から入力

```
char buf[] = "1234";  
int intval;  
sscanf (buf, "%d", &intval);
```

# 環境変数の設定



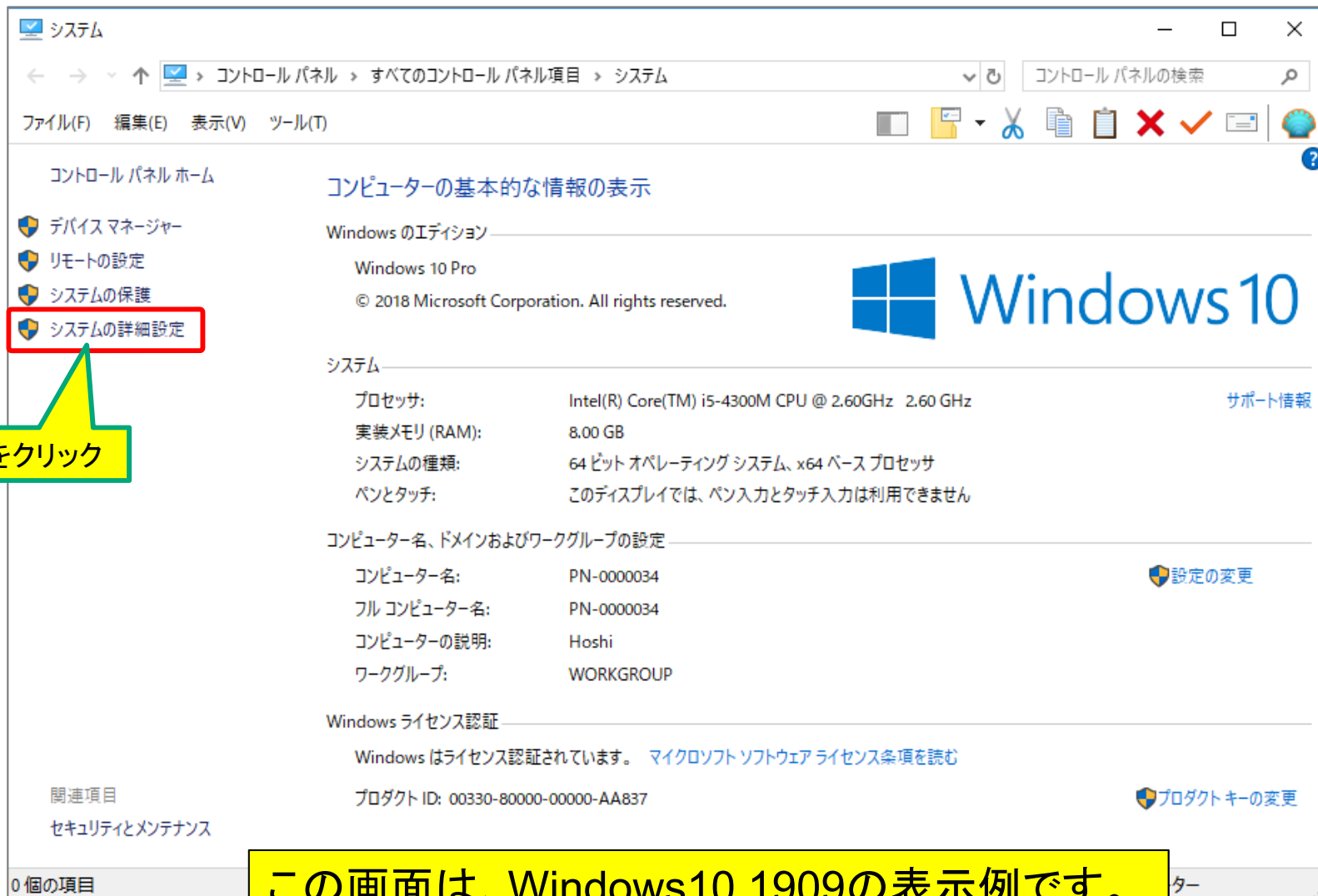
← 現在インストールされているWindowsの詳細バージョン情報

現在のWindowsの詳細なバージョン番号は、「ファイル名を指定して実行」から「winver」と入力すると、左の画面が表示されます。

「ファイル名を指定して実行」は、[Windowsキー]を押し、「Windows システム ツール」の中にあります。  
あるいは、[Windowsキー]を押しながら[R]キーを押しても構いません。

# 環境変数の設定 1/3 (Windows10「1909」以前)

- 環境変数は、システムの環境に合わせて設定する変数のことです。
- Windowsでは、PCの[プロパティ] → [システムの詳細設定]にあります。

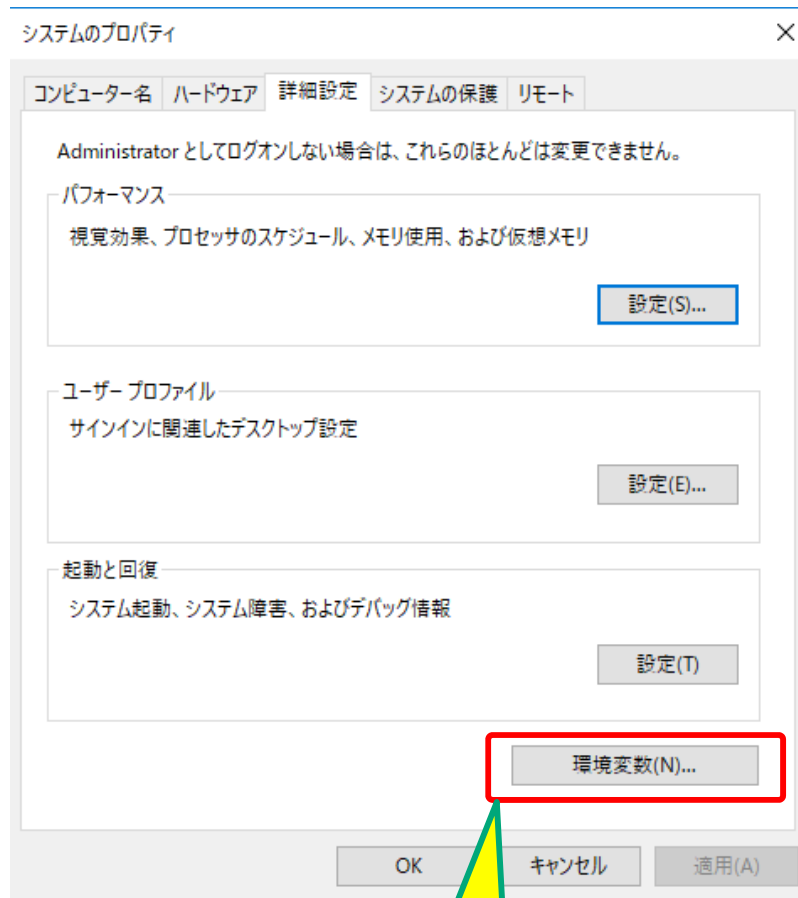


# 環境変数の設定 1/3 (Windows10「20H2」以降)

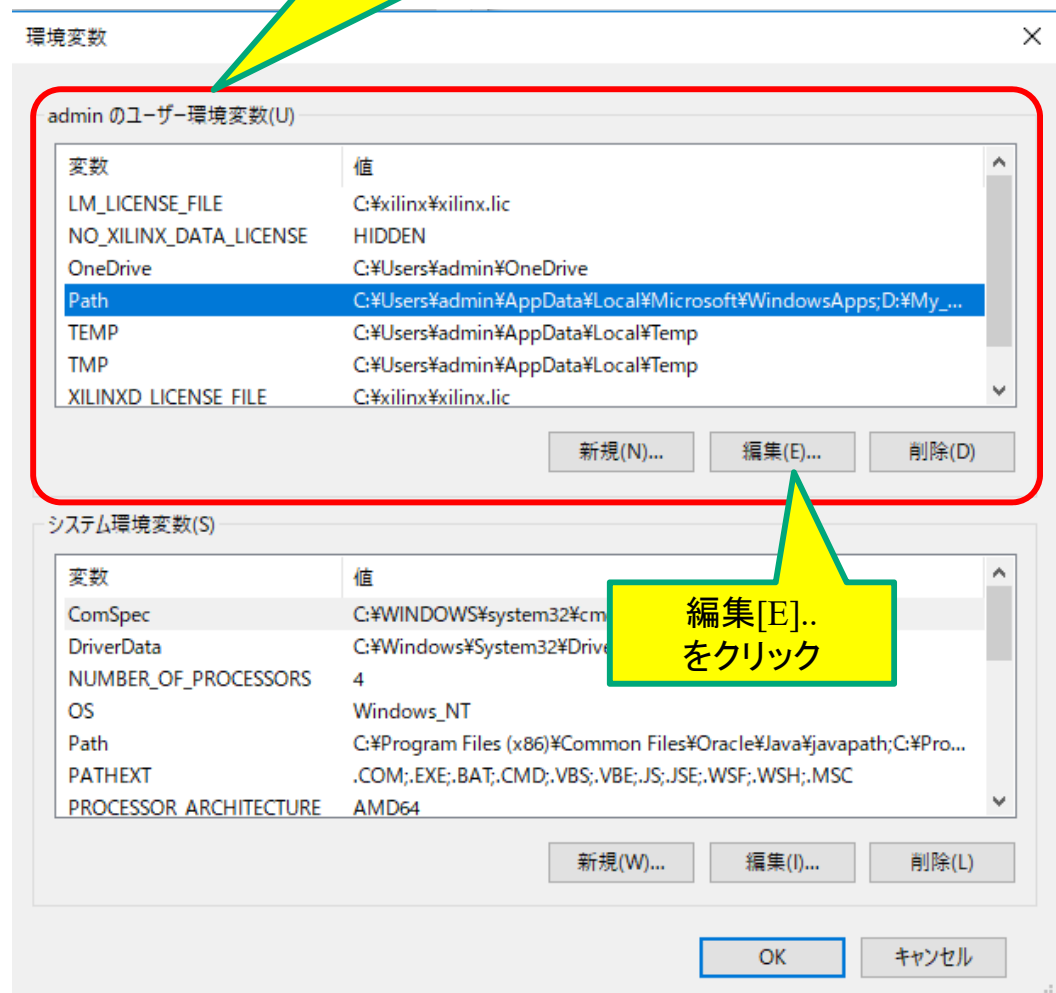
- Windows10 20H2 以降はプロパティ画面が下記のように変わりました。



# 環境変数の設定 2/3



環境変数[N]..  
をクリック



XXXXのユーザー環境変数の[Path]  
を選択して、[編集]をクリック

編集[E]..  
をクリック

# 環境変数の設定 3/3

- コマンドプロンプトで、コマンドを入力したとき、環境変数 Pathで指定されたフォルダを順番にコマンドを探す。

VisualStudioのプロジェクトが保存されているフォルダ

The diagram illustrates the process of setting the environment variable Path to include the path of a Visual Studio project folder. On the left, a file explorer view shows a project folder named '.vs' containing a 'Debug' folder, a 'mydump' folder, and a 'mydump.sln' file. The 'Debug' folder is highlighted with a red box. A yellow callout points to the 'Debug' folder with the text 'このフォルダのパスを設定する' (Set the path of this folder). Another yellow callout points to the 'mydump.exe' file in the 'Debug' folder with the text 'Debugフォルダにある、「xxxx.exe」が実行形式のコマンド' (The command 'xxxx.exe' is in the Debug folder). On the right, a screenshot of the '環境変数名の編集' (Edit Environment Variable Name) dialog box shows the list of environment variables. The variable 'D:\My\_Doc\Documents\Visual Studio 2010\Projects\mydump\Debug' is selected, and a yellow callout points to it with the text 'このフォルダのパスを設定する' (Set the path of this folder). The dialog box also shows other variables like '%USERPROFILE%\AppData\Local\Microsoft\WindowsApps' and 'D:\My\_Doc\Documents\Visual Studio 2010\Projects\testdump\Debug'.

このフォルダのパスを設定する

Debugフォルダにある、「xxxx.exe」が実行形式のコマンド

環境変数名の編集

%USERPROFILE%\AppData\Local\Microsoft\WindowsApps  
D:\My\_Doc\Documents\Visual Studio 2010\Projects\testdump\Debug  
D:\My\_Doc\Documents\Visual Studio 2010\Projects\forTest\Debug  
C:\Users\admin\source\repos\check\_src\Debug  
D:\My\_Doc\Documents\Visual Studio 2010\Projects\mydump\Debug

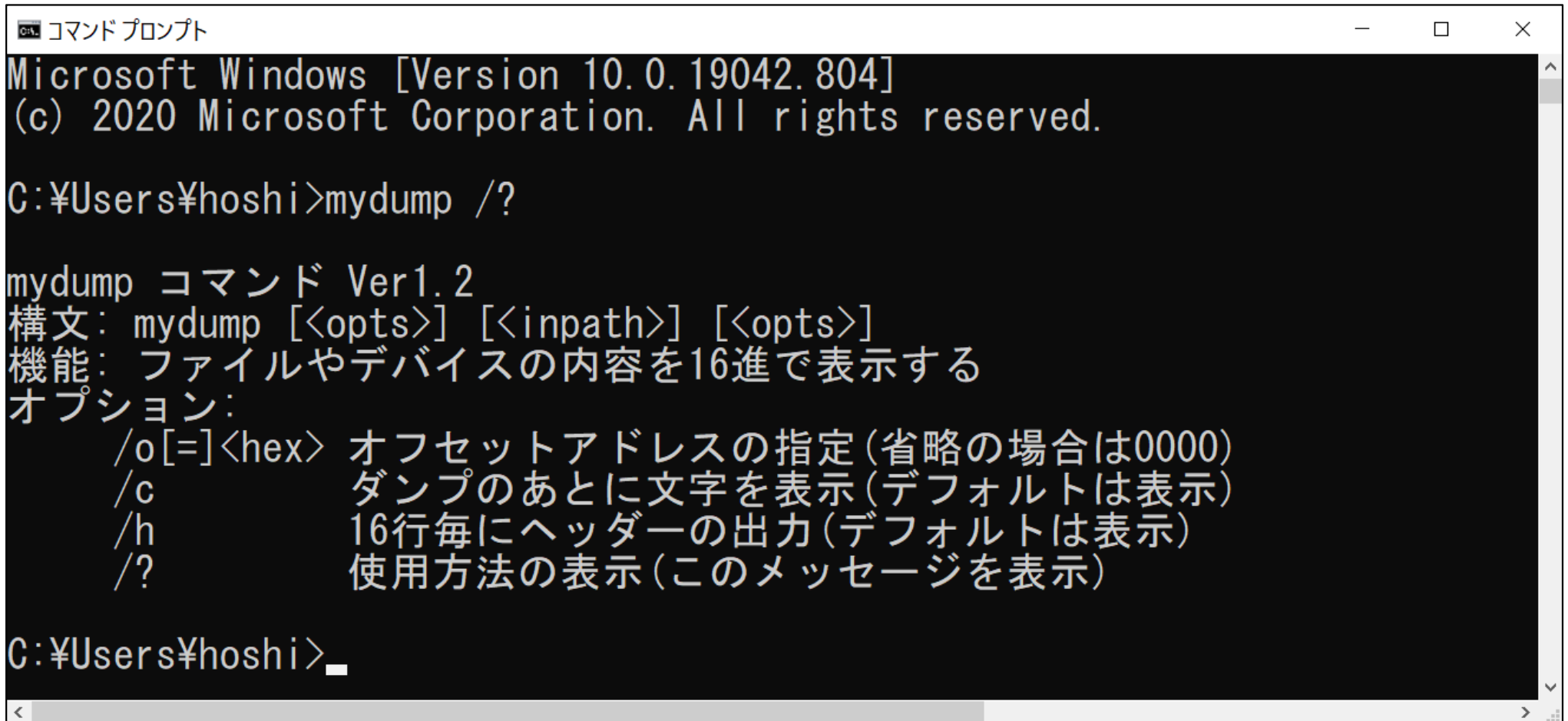
新規(N) 編集(E) 参照(B)... 削除(D) 上へ(U) 下へ(O) テキストの編集(T)...

OK キャンセル



# コマンドプロンプトからコマンドを実行

- 環境変数でPathが指定されると、自分が作成したコマンドが、Windowsのコマンドとして、コマンドプロンプトから実行できる。



```
コマンド プロンプト
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\¥hoshi>mydump /?

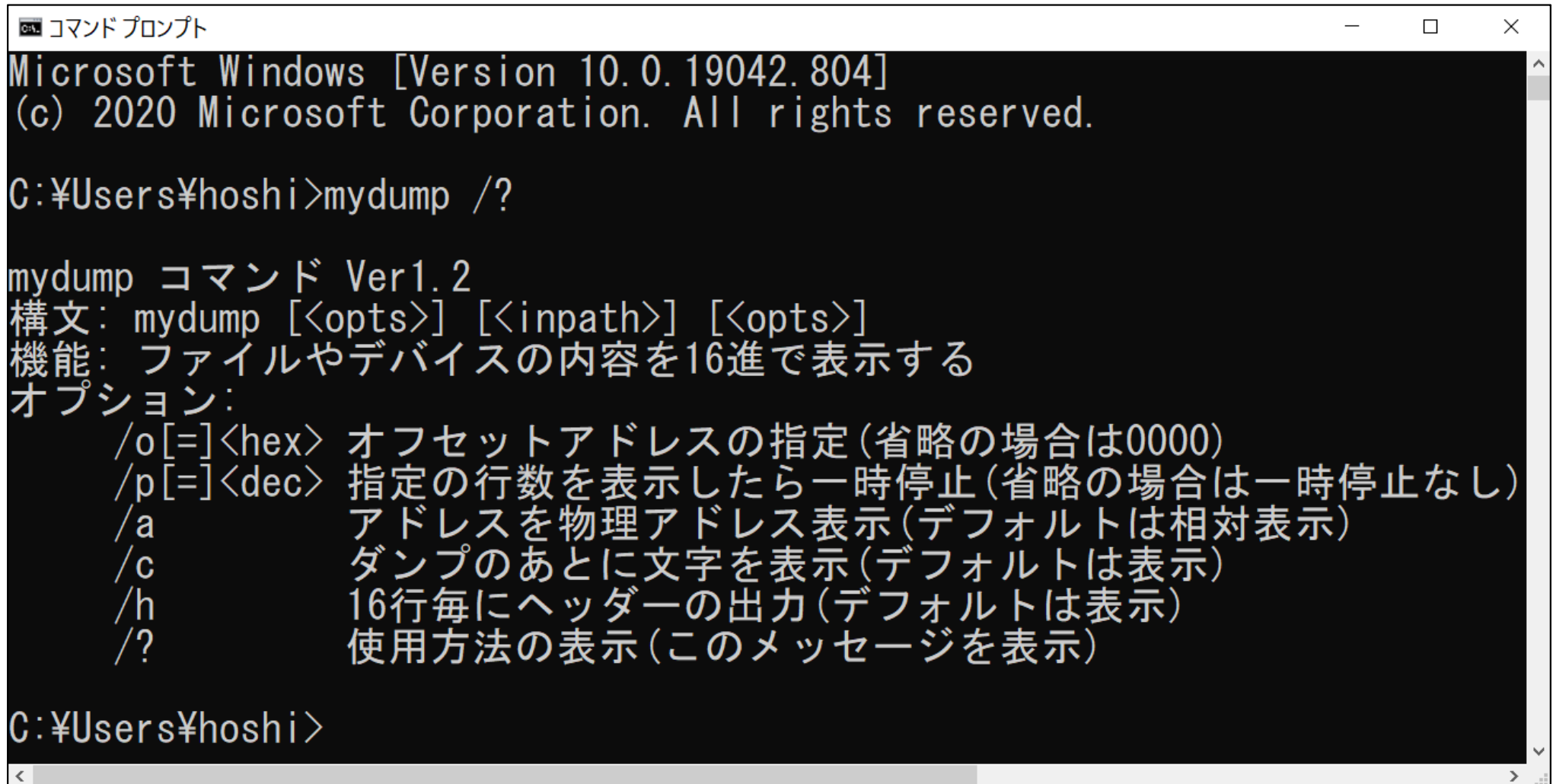
mydump コマンド Ver1.2
構文: mydump [<opts>] [<inpath>] [<opts>]
機能: ファイルやデバイスの内容を16進で表示する
オプション:
    /o[=]<hex> オフセットアドレスの指定(省略の場合は0000)
    /c          ダンプのあとに文字を表示(デフォルトは表示)
    /h          16行毎にヘッダーの出力(デフォルトは表示)
    /?          使用方法の表示(このメッセージを表示)

C:\Users\¥hoshi>_
```

**注意: 環境変数を変更した場合、コマンドプロンプトの再起動が必要 (Windowsの再起動ではない)**

# コマンドプロンプトからコマンドを実行

- /pオプションと/aオプションが追加されたときのヘルプメッセージ



```
コマンド プロンプト
Microsoft Windows [Version 10.0.19042.804]
(c) 2020 Microsoft Corporation. All rights reserved.

C:\Users\¥hoshi>mydump /?

mydump コマンド Ver1.2
構文: mydump [<opts>] [<inpath>] [<opts>]
機能: ファイルやデバイスの内容を16進で表示する
オプション:
  /o[=]<hex> オフセットアドレスの指定(省略の場合は0000)
  /p[=]<dec> 指定の行数を表示したら一時停止(省略の場合は一時停止なし)
  /a          アドレスを物理アドレス表示(デフォルトは相対表示)
  /c          ダンプのあとに文字を表示(デフォルトは表示)
  /h          16行毎にヘッダーの出力(デフォルトは表示)
  /?          使用方法の表示(このメッセージを表示)

C:\Users\¥hoshi>
```

# 実際の実行例

通常の操作(ファイル名だけ指定)  
mydump s-jis2.txt

```
コマンド プロンプト
R:¥>mydump s-jis2.txt
```

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000000	8E84	82BD	82BF	4A54	4543	2043	6F72	706F	私たちJTEC Corpo															
00000010	7261	7469	6F6E	2082	CD81	4131	3939	3694	ration は、1996年															
00000020	4E38	8C8E	82CC	916E	8BC6	88C8	9788	8141	8月の創業以来、															
00000030	93FA	967B	82AA	8E9D	82C2	8175	8FA0	8176	日本が持つ「匠」															
00000040	82CC	90B8	905F	82F0	8C70	8FB3	82B5	8141	の精神を継承し、															
00000050	926D	8EAF	82C9	8AEE	82C3	82AD	926D	8C62	知識に基づく知恵															
00000060	82F0	88B5	82A4	8175	8B5A	8F70	8FA4	8ED0	を扱う「技術商社															
00000070	8176	82C6	82B5	82C4	8141	91BD	82AD	82CC	」として、多くの															
00000080	82A8	8EE6	88F8	90E6	82CC	8A46	976C	82C9	お取引先の皆様に															
00000090	8B5A	8F70	9045	926D	8DE0	D8B0	BD90	EA96	技術職知財リス専門															
000000A0	E582	CC53	6F6C	7574	696F	6E20	436F	6D70	のSolution Comp															
000000B0	616E	7982	C682	B582	C450	6172	746E	6572	anyとしてPartner															
000000C0	7368	6970	82F0	927A	82AB	82C8	82AA	82E7	shipを築きながら															
000000D0	94AD	9357	82B5	82C4	82DC	82A2	82E8	82DC	発展してまいりま															
000000E0	82B5	82BD	8142	0D0A	93FA	967B	82F0	8EE6	した。..日本を取															
000000F0	82E8	8AAA	82AD	8ED0	89EF	8AC2	8BAB	82CD	り巻く社会環境は															

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000100	91E5	82AB	82C8	95CF	89BB	82F0	908B	82B0	大きな変化を遂げ															
00000110	82E6	82A4	82C6	82B5	82C4	82A2	82DC	82B7	ようとしています															
00000120	8142	8E84	82C7	82E0	82AA	92BC	90DA	8C57	。私どもが直接係															
00000130	82E9	90BB	91A2	8BC6	82F0	9286	9053	82C6	る製造業を中心と															
00000140	82B5	82BD	8AE9	8BC6	8D5C	91A2	82E0	82DF	した企業構造もめ															
00000150	82DC	82AE	82E9	82B5	82AD	95CF	9665	82B5	まぐるしく変貌し															
00000160	82C4	82E4	82AD	8FF3	8BB5	89BA	82C9	82A8	てゆく状況下にお															
00000170	82AB	82DC	82B5	82C4	8141	8D91	93E0	82CD	きまして、国内は															
00000180	82E0	82C6	82E6	82E8	906C	8DE0	97AC	93AE	もとより人財流動															
00000190	82CC	8D91	8DDB	89BB	82E0	8E8B	96EC	82C9	の国際化も視野に															
000001A0	93FC	82EA	8141	8E84	82BD	82BF	4A54	4543	入れ、私たちJTEC															

/o=80 /p=10 と /a オプションを指定(追加課題)  
mydump s-jis2.txt /o=80 /p=10  
mydump s-jis2.txt /o=80 /p=10 /a

```
コマンド プロンプト
R:¥>mydump s-jis2.txt /o=80 /p=10
```

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000000	82A8	8EE6	88F8	90E6	82CC	8A46	976C	82C9	お取引先の皆様に															
00000010	8B5A	8F70	9045	926D	8DE0	D8B0	BD90	EA96	技術職知財リス専門															
00000020	E582	CC53	6F6C	7574	696F	6E20	436F	6D70	のSolution Comp															
00000030	616E	7982	C682	B582	C450	6172	746E	6572	anyとしてPartner															
00000040	7368	6970	82F0	927A	82AB	82C8	82AA	82E7	shipを築きながら															
00000050	94AD	9357	82B5	82C4	82DC	82A2	82E8	82DC	発展してまいりま															
00000060	82B5	82BD	8142	0D0A	93FA	967B	82F0	8EE6	した。..日本を取															
00000070	82E8	8AAA	82AD	8ED0	89EF	8AC2	8BAB	82CD	り巻く社会環境は															
00000080	91E5	82AB	82C8	95CF	89BB	82F0	908B	82B0	大きな変化を遂げ															
00000090	82E6	82A4	82C6	82B5	82C4	82A2	82DC	82B7	ようとしています															

10 行目で一時停止: いずれかのキーを押したら続行, Escキーで終了

```
R:¥>mydump s-jis2.txt /o=80 /p=10 /a
```

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000080	82A8	8EE6	88F8	90E6	82CC	8A46	976C	82C9	お取引先の皆様に															
00000090	8B5A	8F70	9045	926D	8DE0	D8B0	BD90	EA96	技術職知財リス専門															
000000A0	E582	CC53	6F6C	7574	696F	6E20	436F	6D70	のSolution Comp															
000000B0	616E	7982	C682	B582	C450	6172	746E	6572	anyとしてPartner															
000000C0	7368	6970	82F0	927A	82AB	82C8	82AA	82E7	shipを築きながら															
000000D0	94AD	9357	82B5	82C4	82DC	82A2	82E8	82DC	発展してまいりま															
000000E0	82B5	82BD	8142	0D0A	93FA	967B	82F0	8EE6	した。..日本を取															
000000F0	82E8	8AAA	82AD	8ED0	89EF	8AC2	8BAB	82CD	り巻く社会環境は															
00000100	91E5	82AB	82C8	95CF	89BB	82F0	908B	82B0	大きな変化を遂げ															
00000110	82E6	82A4	82C6	82B5	82C4	82A2	82DC	82B7	ようとしています															

10 行目で一時停止: いずれかのキーを押したら続行, Escキーで終了

```
R:¥>
```



# 標準入力とパイプラインによる実行例

引数がない場合、標準入力からの入力  
キーボードからの入力を表示

```
コマンド プロンプト - mydump
R:¥>
R:¥>mydump
1234567890qwertyuiop@[

Addr    0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
-----
00000000 3132 3334 3536 3738 3930 7177 6572 7479 1234567890qwerty
```

入力をリダイレクト(mydumpはオープンしていない)  
mydump <s-jis2.txt

```
コマンド プロンプト
R:¥>mydump <s-jis2.txt

Addr    0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
-----
00000000 8E84 82BD 82BF 4A54 4543 2043 6F72 706F 私たちJTEC Corpo
00000010 7261 7469 6F6E 2082 CD81 4131 3939 3694 ration は、1996年
00000020 4E38 8C8E 82CC 916E 8BC6 88C8 9788 8141 8月の創業以来、
00000030 93FA 967B 82AA 8E9D 82C2 8175 8FA0 8176 日本が持つ「匠」
00000040 82CC 90B8 905F 82F0 8C70 8FB3 82B5 8141 の精神を継承し、
00000050 926D 8EAF 82C9 8AEE 82C3 82AD 926D 8C62 知識に基づく知恵
00000060 82F0 88B5 82A4 8175 8B5A 8F70 8FA4 8ED0 を扱う「技術商社
00000070 8176 82C6 82B5 82C4 8141 91BD 82AD 82CC 」として、多くの
00000080 82A8 8EE6 88F8 90E6 82CC 8A46 976C 82C9 お取引先の皆様に
00000090 8B5A 8F70 9045 926D 8DE0 D8B0 BD90 EA96 技術職知財リス専門
000000A0 E582 CC53 6F6C 7574 696F 6E20 436F 6D70 のSolution Comp
000000B0 616E 7982 C682 B582 C450 6172 746E 6572 anyとしてPartner
000000C0 7368 6970 82F0 927A 82AB 82C8 82AA 82E7 shipを築きながら
000000D0 94AD 9357 82B5 82C4 82DC 82A2 82E8 82DC 発展してまいりま
000000E0 82B5 82BD 8142 0A93 FA96 7B82 F08E E682 した。日本を取り
000000F0 E88A AA82 AD8E D089 EF8A C28B AB82 CD91 巻く社会環境は大

Addr    0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
-----
00000100 E582 AB82 C895 5F89 BB82 F090 8B82 B082 きな変化を遂げよ
00000110 E682 A482 C687 5F82 C482 A282 DC82 B781 うとしています。
00000120 428E 8482 C787 5F82 AA92 BC90 DA8C 5782 私どもが直接係る
00000130 E990 BB91 A287 5F82 F092 8690 5382 C682 製造業を中心とし
00000140 B582 BD8A E987 5F82 5C91 A282 E082 DF82 た企業構造もめま
00000150 DC82 AE82 E987 5F82 AD95 CF96 6582 B582 ぐるしく変貌して
00000160 C482 E482 5F82 5F82 B589 BA82 C982 A882 ゆく状況下におき
00000170 AB82 DC82 5F82 5F82 418D 9193 E082 CD82 まして、国内はも
00000180 E082 C682 5F82 5F82 06C8D E097 AC93 AE82 とより人財流動の
00000190 CC8D 918D 5F82 5F82 2E08E 8B96 EC82 C993 国際化も視野に入
000001A0 FC82 FA82 5F82 5F82 2BD82 BF4A 5445 4320 れ、私たちJTEC
```

パイプライン(dir の出力がmydumpの入力になる)  
dir /b | mydump

```
コマンド プロンプト
C:¥>dir /b
PerfLogs
Program Files
Program Files (x86)
Scratch
trdock_debug.log
Users
Windows

C:¥>dir /b | mydump

Addr    0 1  2 3  4 5  6 7  8 9  A B  C D  E F  0 2 4 6 8 A C E
-----
00000000 5065 7266 4C6F 6773 0A50 726F 6772 616D PerfLogs. Program
00000010 2046 696C 6573 0A50 726F 6772 616D 2046 Files. Program F
00000020 696C 6573 2028 7838 3629 0A53 6372 6174 iles (x86).Scratch.
00000030 6368 0A74 7264 6F63 6B5F 6465 6275 672E ch.trdock_debug.
00000040 6C6F 670A 5573 6572 730A 5769 6E64 6F77 log.Users.Window
00000050 730A s.

C:¥>
```

0D0Aが、0Aだけになっている

# 追加課題

---

# 追加課題1 /p オプションの追加

## ■ /pオプションの追加して、表示を一時停止する

– /p[=]<dec> を追加 ← option\_analysisの中に追加(これは簡単ですね)

– 使用例:

◆ /p=16 ← 16行表示して停止

◆ /p32 ← 32行表示して停止

– 指定した行数を表示したら下記のメッセージを表示しキー入力待ちとする

◆ "xx行目で一時停止: いずれかのキーを押したら続行, Escキーで終了"

◆ このキー入力待ちは、getch( )関数を使うと良い

– ただし、ポーズの機能は自分でファイルをオープンした時だけ動作

◆ ファイル名がないと、データはstdinから読まれるため、キーボードからのキーの読み込みができないため

– プログラムのヒント

◆ ポーズの機能は、dump関数のwhileループで、S-JISの泣き別れ処理をした後に入れる

◆ dump関数は、void型からint型にして、Escキーの値を返す。

◆ mianのwhileループの中で、dump関数から戻り値を調べESCキーなら処理を終了する

## 追加課題2 /a オプションの追加

- /a オプションを追加して、アドレスを実際の位置表示にする
  - アドレス表示は、0から始まる相対値から、実際のアドレス表示にする
  - fseek() を実行後に、シークした値をアドレスの初期値とするだけ
  - そのため、アドレスの変数は、構造体の中に移動する

「mydump ファイル名 /o=a0 /p=6」で実行

アドレスの相対表示  
0 から始まっている

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
00000000	0000	0000	4170	706C	6500	6950	686F	6E65	.... Apple. iPhone															
00000010	2038	0000	0000	0048	0000	0001	0000	0048	8.....H.....H															
00000020	0000	0001	3132	2E31	0000	3230	3138	3A31	.... 12. 1.. 2018:1															
00000030	313A	3033	2031	323A	3437	3A32	3100	0020	1:03 12:47:21..															
00000040	829A	0005	0000	0001	0000	0246	829D	0005	z.....F...															
00000050	0000	0001	0000	024E	8822	0003	0000	0001	.....N.".....															

「mydump ファイル名 /o=a0 /p=6 /a」で実行

アドレスの絶対表示  
A0 から始まっている

Addr	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	2	4	6	8	A	C	E
000000A0	0000	0000	4170	706C	6500	6950	686F	6E65	.... Apple. iPhone															
000000B0	2038	0000	0000	0048	0000	0001	0000	0048	8.....H.....H															
000000C0	0000	0001	3132	2E31	0000	3230	3138	3A31	.... 12. 1.. 2018:1															
000000D0	313A	3033	2031	323A	3437	3A32	3100	0020	1:03 12:47:21..															
000000E0	829A	0005	0000	0001	0000	0246	829D	0005	z.....F...															
000000F0	0000	0001	0000	024E	8822	0003	0000	0001	.....N.".....															

# ヘッダ表示の関数化

===== 2バイト表示 =====

文字表示あり

Addr    0 1   2 3   4 5   6 7   8 9   A B   C D   E F   0 2 4 6 8 A C E

文字表示なし

Addr    0 1   2 3   4 5   6 7   8 9   A B   C D   E F

===== 1バイト表示 =====

文字表示あり

Addr    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F   0 2 4 6 8 A C E

文字表示なし

Addr    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F