ASCII文字コード

上位4ビット

	0?	1?	2?	3?	4?	5?	6?	7?
?0	NUL		SP	0	@	P	`	p
?1			!	1	A	Q	a	q
?2			"	2	B	R	b	r
?3			#	3	C	S	c	s
?4			$	4	D	T	d	t
?5			%	5	E	U	e	u
?6			&	6	F	V	f	v
?7			'	7	G	W	g	w
?8	BS		(	8	H	X	h	x
?9	HT		)	9	I	Y	i	y
?A	LF		*	:	J	Z	j	z
?B		ESC	+	;	K	[	k	{
?C			,	<	L	¥	l	|
?D	CR		-	=	M	]	m	|
?E			.	>	N	^	n	~
?F			/	?	O	_	o	DEL

下位4ビット

NUL	Null	空文字
BS	Back Space	一文字後退
HT	Horizontal Tabulation	水平タブ
LF	Line Feed	ラインフィード
CR	Carriage Return	行頭復帰
ESC	Escape	エスケープ
SP	Space	空白文字
DEL	Delete	一文字削除

Enterキーは昔はReturnキーと呼ばれていた．改行を表す文字コードは，CR，LF，またはCRLF（CRに続けてLFという2バイト）のいずれかであり，OSやライブラリやファイル形式などによって異なる．
glutKeyboardFuncのコールバック関数に渡される引数keyの場合，Enterキーを押したときのASCIIコードはCRである．

演算子			
演算子	意味	使用法	
+	加算	a = b + c;	bとcを加えた値をaに代入する
−	減算	a = b − c;	bからcを引いた値をaに代入する
*	乗算	a = b * c;	bとcを掛けた値をaに代入する
/	除算	a = b / c;	bをcで割った値をaに代入する
%	剰余算	a = b % c;	bをcで割った余りの値をaに代入する

演算子	意味	使用法
++	インクリメント	a++;
−−	デクリメント	a−−;
−	符号の反転	b=−a;

演算子	使用法	通常の算術演算子による表記
=	a = b;	
+=	a += b;	a = a + b;
−=	a −= b;	a = a − b;
*=	a *= b;	a = a * b;
/=	a /= b;	a = a / b;
%=	a %= b;	a = a % b;

演算子	
>	より大
<	より小
>=	より大か等しい
<=	より小か等しい
==	等しい
!=	等しくない

演算子	
&&	論理演算のAND
||	論理演算のOR
!	論理演算のNOT

演算子	
&	ビット演算のAND
|	ビット演算のOR
~	ビット演算のNOT

- if(a==3)と書くべきところにif(a=3)と書かないように注意

データ型

種別	符号の有無	ビット長	表現	数値の範囲
整数	あり	8	(signed) char	−128〜+127
		16	(signed) short	−32768〜+32767
		32	(signed) int	−2147483648〜+2147483647
	なし	8	unsigned char	0〜255
		16	unsigned short	0〜65535
		32	unsigned int	0〜4294967295
浮動小数点数	あり	32	float	およそ10^{-38}〜10^{38}(10進数最大7桁の精度)
		64	double	およそ10^{-306}〜10^{306}(10進数最大16桁の精度)
		8など	bool	true(1など0以外の値)またはfalse(0)

- 16ビットOSでは多くのコンパイラではint型は16ビットだった
- 32ビットOSでは多くのコンパイラでint型は32ビットである
- 64ビットOSでは多くのコンパイラでは，int型は64ビット…ではなく，int型は32ビットであることが多い
- bool型はC言語ではなくC++の型で，ビット長が8ビットではないコンパイラもある
- ポインタ型のビット長は，32ビットコンパイルした場合は32ビット，64ビットコンパイルした場合は64ビット
- void型は「値のない型」
- static変数は一度記憶場所が割り当てられると，常にその場所に値を保存しておくことができる

sizeof演算子

- sizeof(式または型名)
  - 式または型名に対し，処理系が割り当てたメモリ容量をバイト単位で返す
  - 例：sizeof(short)は2
  - 例：short a = 0; sizeof(a)は2

定数

- 例えば0x1aのように16進数をあらわす
  - 0x1aは10進数では26である
- 例えば3.14はdouble型の浮動小数点数，3.14fはfloat型の浮動小数点数をあらわす
- 浮動小数点数はeを付けて指数部を表現することができる
  - 例えば3.14e4は31400をあらわす
- 一文字は'A'のようにシングルクォーテーションで囲む
- 文字列は"STRING"のようにダブルクォーテーションで囲む
- const int a = 1;によりaは定数として扱われる

浮動小数点

- 浮動小数点には誤差があるので，数値が完全に一致することを前提としたコードを書かないこと

プログラム

```
double a = 1.0;
for (int i = 0; i < 10; i++) {
    a -= 0.1;
}
if (a == 0.0) printf("a is zero¥n");
else printf("a is not zero¥n");
if (a >= -1.0e-15 && a <= 1.0e-15) printf("a is zero¥n");
else printf("a is not zero¥n");
```

実行結果

```
a is not zero
a is zero
```

ローカル変数とグローバル変数

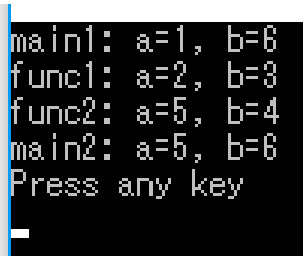
```
#include <stdio.h>)
#include <conio.h>

int a = 1;

void func1()
{
    int a = 2, b = 3;
    printf("func1: a=%d, b=%d¥n", a, b);
}

void func2()
{
    int b = 4;
    a = 5;
    printf("func2: a=%d, b=%d¥n", a, b);
}

int main(int argc, char* argv[])
{
    int b = 6;

    printf("main1: a=%d, b=%d¥n", a, b);
    func1();
    func2();
    printf("main2: a=%d, b=%d¥n", a, b);

    printf("Press any key¥n");
    _getch();

    return 0;
}
```

グローバル変数aのスコープ

ローカル変数aとbのスコープ

ローカル変数bのスコープ

ローカル変数bのスコープ



型の変換

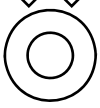
- c = (int)a; // aをint型に変換してcに代入する例
- int a = 1; int b = 10;のとき
  - a / bは0である
  - (double)a / (double)bは0.1である
- unsigned char a = 0; unsigned char b = 1; unsigned char c = a - b;のときcは255である
  - c = (int)a - (int)bのときのcも255である
  - int d = (int)a - (int)bのときdは-1である
- 右の関数(CではなくC++の関数)が定義されているとき
  - int i0 = 0; int i1 = 1;のときminus(i0,i1)は-1である
  - double d0 = 0.0; double d1 = 1.0;のときminus(d0,d1)は-1.0である
  - unsigned char uc0 = 0; unsigned char uc1 = 1;のときminus(uc0,uc1)は255である
  - 同じ関数を呼び出したとしてもデータ型によって結果が変わることがある

```
template <typename T>
T minus(T a, T b) {
   return a - b;
};
```

型はC言語が自動的に変換してくれるから型キャストしなくてもいいんじゃない？
型を指定しなくても自動的に型を定義して変換してくれる他のプログラミング言語にしたほうがいいんじゃない？

プログラミング言語が型を自動的に変換してくれる…その通り.
でも，君は「自分の都合のいいように」型を自動的に変換してくれる，と勘違いしていないか？
「そのプログラミング言語のルールに従って」型を自動的に変換してくれるんだよ.
君の思い通りに自動的に型を変換してくれるわけじゃないよ.
C言語に限らず他のどのプログラミング言語もその点は同じだよ.

プログラマーの意図をプログラミング言語が汲み取って，プログラミング言語がプログラマーの意図した通りに，プログラミング言語が自らの意志で適切に判断して，プログラムが想定通りの動作をするように型を正しく変換してくれる



プログラマーの意思や要望，プログラムの目的などを完全に無視して，プログラマーの都合など全く考えず，事前に定義されたルールに従って，プログラミング言語の独自の判断で，型を勝手に変えやがる

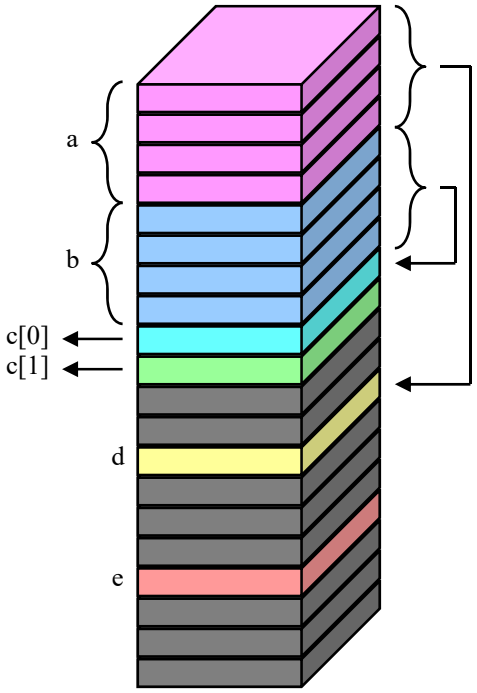
ポインタ

```
char* a;
char* b;
char c[2];
char d;
char e;
d = 13;
a = &d;
b = &e;
*b = 14;
c[0] = 15;
b = c;
b[1] = 12;
*b = 11;
```

⇨

これにより
aにはdのアドレスが入っている
bにはc[0]のアドレスが入っている
c[0]には11という数値が入っている（15ではない）
c[1]には12という数値が入っている
dには13という数値が入っている
eには14という数値が入っている
*aを見ると13という数値が入っている
*bを見ると11という数値が入っている
*cを見ると11という数値が入っている
b[0]を見ると11という数値が入っている
b[1]を見ると12という数値が入っている

このとき，b[2]というのがソースコードにあるとき，その部分は完全にバグであるため，
本来はエラーになって欲しいのに実際にはエラーメッセージが発生しないことが多い．
つまり，エラーメッセージがでなくてもバグが存在することがある．
ポインタ関係でバグが発生するとエラーメッセージがでないことが多いため，
バグがあることを気づかず，また，バグがあることに気づいたとしても
どこがバグであるか発見することが難しいことが多い．
もちろん，たまにエラーメッセージが発生することもある．



※この図ではアドレスは32ビットとしている
（Visual C++の現在のバージョンでは，例え64ビットWindows上で開発しても，
デフォルトでは32ビットプログラムとしてコンパイルされる）
（もちろん，64ビットプログラムとしてコンパイルすることは可能）

※実際にそれぞれの変数のアドレスがいくつになるかはコンパイラ依存でもあるし，
そもそも実行のたびに毎回違う値になる．
（char dのあとにchar eを書いたからといってdとeのアドレスが近い値になるとは限らない）

配列

- 3次元配列の初期化の一例　中括弧{}を使うべきところを丸括弧()を使わないように注意

```
int arr[2][3][4] = {
    {
        {  0,   1,   2,   3 },
        {  10,  11,  12,  13 },
        {  20,  21,  22,  23 }
    }, {
        { 100, 101, 102, 103 },
        { 110, 111, 112, 113 },
        { 120, 121, 122, 123 }
    } };
```

構造体

- 構造体の一例

```
typedef struct _vertex {
    double x;
    double y;
    double z;
} vertex;
vertex data[2];
data[0].x = 1.0;
data[0].y = 0.0;
data[0].z = 0.0;
data[1].x = 0.0;
data[1].y = 1.0;
data[1].z = 0.0;
vertex* p;
p = &(data[1]);
printf("%f %f %f¥n", p->x, p->y, p->z);
```

⇨ 実行結果は
0.000000 1.000000 0.000000

main関数

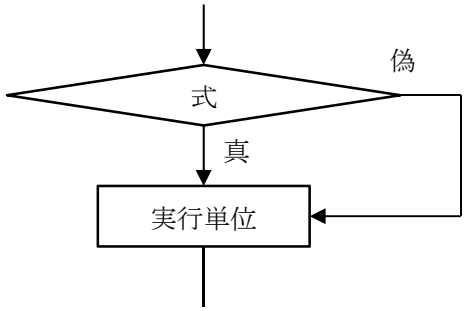
- int main(int argc, char* argv[])
- 正常終了なら0を返し，異常終了なら1を返す
- .exeファイルを実行したときのコマンドライン引数はargvで取得できる
  - このとき，実行ファイル名を含めた引数の合計はargcで得られる
  - cg2.exe texture.bmp miku.obj
    のように実行された場合，argcは3で，
    argv[0]はcg2.exe
    argv[1]はtexture.bmp
    argv[2]はmiku.obj
    となる
- Visual C++上で実行するときにコマンドライン引数を指定したい場合
  - Visual C++でslnファイルを開いて，プロジェクトのプロパティを開く
  - [構成プロパティ]-[デバッグ]-[コマンド引数]で引数と書く
    - 上の例の場合は
      texture.bmp miku.obj
      と書く

プリプロセッサ

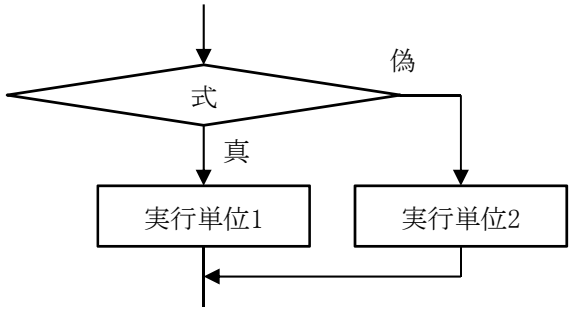
- #include ファイルをインクルード
- #define マクロ定義
- #pragma 指令（コンパイラにコンパイラ特有の情報を伝えるために使用する）

if文

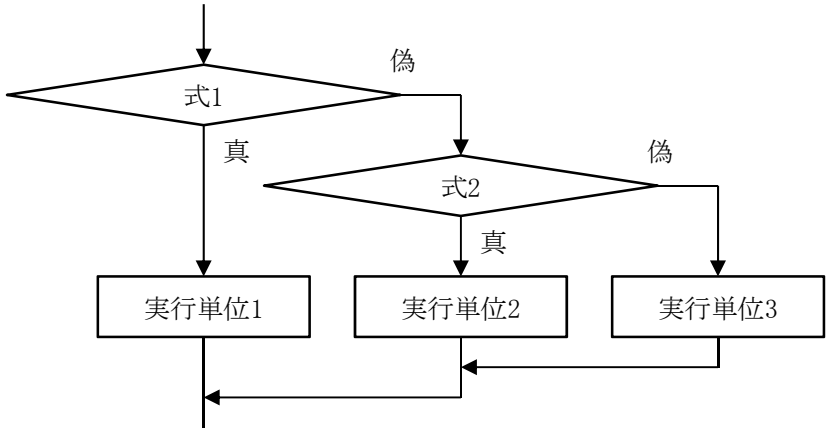
```
if (式)
   実行単位
```




```
if (式)
   実行単位1
else
   実行単位2
```

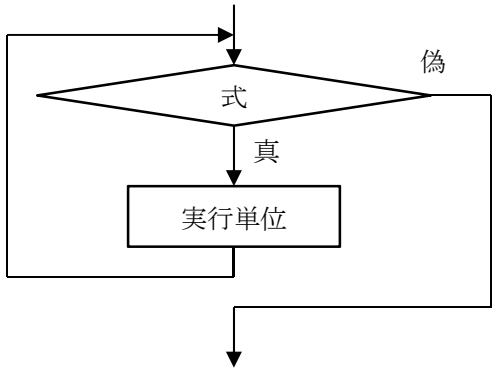



```
if (式1)
   実行単位1
else if (式2)
   実行単位2
else
   実行単位3
```



while文

```
while (式)
   実行単位
```

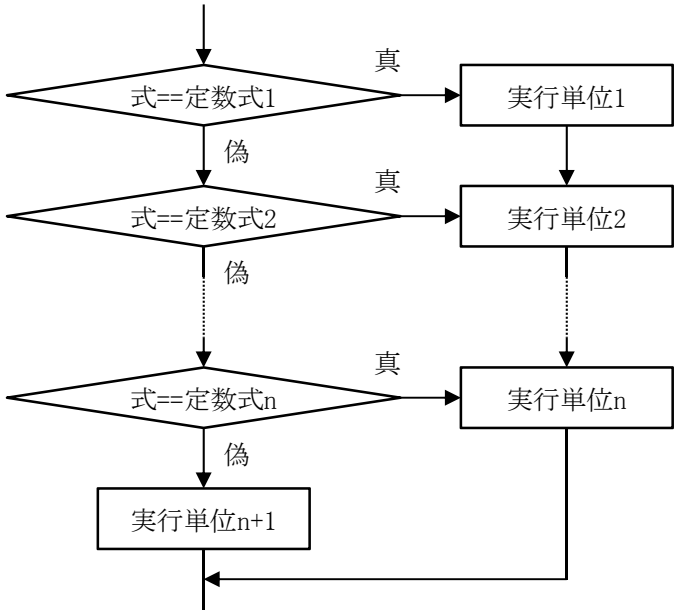


return文

- 現在実行中の関数を抜ける
- 関数が返り値を持つ持つ場合はreturn (式);のように書く

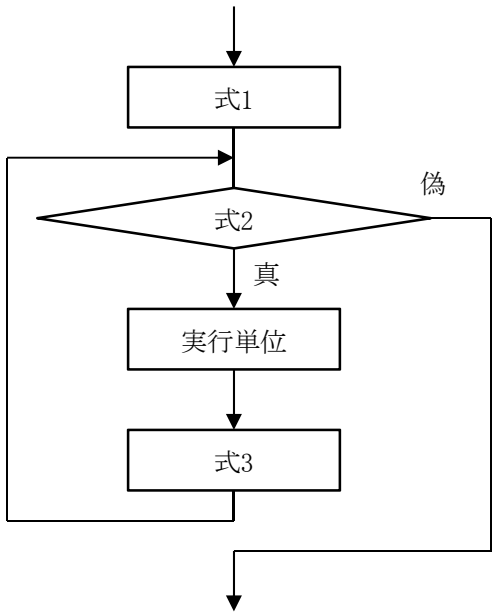
switch～case文

```
switch (式)
{
   case 定数式1:
      実行単位1
      break;
   case 定数式2:
      実行単位2
      break;
    ・
    ・
    ・
   case 定数式n:
      実行単位n
      break;
   default:
      実行単位n+1
}
```



for文

```
for (式1;式2;式3)
   実行単位
```



- for文の中で変数を意図せず上書きしないように注意
  - for(i=...
    - for(i=...など

break文

- for文とwhile文とswitch文で使われる
- 現在実行中の実行単位を抜ける

continue文

- for文とwhile文で使われる
- ループの最後に行く

printf文

- int printf(char *format[, arg1, arg2, ...]);
  - 文字列を表示

使用例	出力結果
int a = 123; printf("%d", a);	123
short a = 123; printf("%d", a);	123
char a[] = "ABC"; printf("%s", a);	ABC
double a = 3.14; printf("%e", a);	3.140000e+00
float a = 3.14f; printf("%e", a);	3.140000e+00
double a = 3.14; printf("%f", a);	3.140000
float a = 3.14f; printf("%f", a);	3.140000
double a = 3.14; printf("%g", a);	3.14
float a = 3.14f; printf("%g", a);	3.14
double a = 3.14; printf("%.3f", a);	3.140
double a = 3.14; printf("%6.2f", a);	3.14
double a = -3.14; printf("%6.2f", a);	-3.14
int a = 1; printf("%04d", a);	0001
int a = 1; printf("% 4d", a);	1

制御文字	機能
¥n	改行
¥r	行の先頭に戻す
¥¥	¥記号を出力
¥'	シングルクォーテーションを出力
¥"	ダブルクォーテーションを出力
¥0	ヌル文字

stdio.h

- int printf(char *format[, arg1, arg2, ...]);
  - 文字列を表示
  - 詳細は前のページで
- int scanf(const char *format [, argument]...);
  - 文字を読みとります
  - %d　10進整数．intへのポインタ．
  - %f　浮動小数点．floatへのポインタ．%eや%gも同様．
  - %lf　浮動小数点．doubleへのポインタ．%leや%lgも同様．
  - %s　文字列．文字配列．
- FILE *fopen(char *fname, char *mode);
  - fnameで示されるファイルをmodeで示されるモードでオープン
  - モード："rt"でテキストファイルの読み込み
  - 戻り値：ファイルポインタ（正常時），NULL（エラー時）
- int fclose(FILE *fp);
  - ファイルポインタfpのファイルをクローズ
- int fscanf(FILE *fp, char *format[, arg1, arg2, ...]);
  - ファイルポインタfpから書式formatにしたがって引数arg1, arg2, ... を入力
  - 戻り値：入力した項目数．ファイルの終りの場合はEOF(-1)
- char *fgets(char *str, int n, FILE *stream);
  - streamから1行，最大n文字を読み込んでstrに格納
  - 戻り値：str（通常時），NULL（エラーやファイルの終端）（エラーかファイルの終端かはfeofやferrorを使う）
- int fread(void *buffer, unsigned int size, unsigned int n, FILE *fp);
  - ファイルポインタfpからsizeバイトのデータブロックをnブロック分バッファbufferに読み込む
- int feof(FILE *fp);
  - 現在位置がファイルエンドならば0以外の値，ファイルエンドでないときは0
- int sprintf(char *buffer, char *format[, arg1, arg2, ...]);
  - 書式formatにしたがって，arg1，arg2，…のデータを文字列bufferに出力
  - 戻り値：バッファに格納した文字数
- int sscanf(char *buffer, char *format[, arg1, arg2, ...]);
  - 文字列bufferから書式formatにしたがって，arg1, arg2, ...のデータを入力
  - 戻り値：入力した項目数．エラー時はEOF(-1)
- int fread(void *buffer, unsigned int size, unsigned int n, FILE *fp);
  - ファイルポインタfpからsizeバイトのデータブロックをnブロック分バッファbufferに読み込む
- int fseek(FILE *fp, long offset, int mode)
  - ファイルポインタfpの現在位置をmodeで指定した位置（0:先頭，1:現在位置，2:終端）からoffsetバイト移動

time.h

- time_t time(time_t *timer);
  - 1970年1月1日午前0時からの経過時間を秒単位で返す
  - timer　時刻が格納される（NULLの場合は格納されない）

conio.h

- int _getch(void);
  - コンソールから文字を取得
  - 戻り値：読みとられた文字

stdlib.h

- int atoi(char *string);
  - 文字列stringをint型数値に変換
- double atof(char *string);
  - 文字列stringをdouble型数値に変換
- void *malloc(unsigned int n);
  - nバイトのメモリブロックを確保
  - 戻り値：正常なら確保したメモリへのポインタ，エラーならNULL
- void free(void *p);
  - mallocで確保したメモリpで示されるメモリブロックを解放
- void exit(int status);
  - プログラムを終了させて，statusの値を親プロセスに返す
- int rand(void);
  - 擬似乱数発生
  - 0からRAND_MAXまでの整数
    - RAND_MAXの値は処理系によって異なるが，Visual C++の場合は32767など
- void srand(unsigned int seed);
  - 擬似乱数のシードを指定する

math.h

- M_PI
  - 3.14159265358979323846
  - #define _USE_MATH_DEFINES
    #include <math.h>
    のように，math.hのインクロードの前に_USE_MATH_DEFINESをマクロ定義すると使える
- double sqrt(double x);
  - xの平方根
- double fmod(double x, double y);
  - xをyで割った余り
- double cos(double x);
  - コサイン
- double sin(double x);
  - サイン

string.h

- char *strchr(char *string, int c);
  - 文字列stringの先頭から文字cをサーチ
  - 戻り値：見つかった位置のポインタ、見つからなかった場合はNULL
- char *strstr(char *string1, char *string2);
  - 文字列string1の先頭から文字列string2をサーチ
  - 戻り値：見つかった位置のポインタ、見つからなかった場合はNULL
- int strcmp(char *string1, char *string2);
  - 文字列string1と文字列string2を比較
  - 戻り値：string1>string2なら正、string1=string2なら0、string1<string2なら負
- int strncmp(char *string1, char *string2, unsigned int n);
  - 文字列string1と文字列string2の先頭からn文字を比較
- char *strcpy(char *string1, char *string2);
  - string1が示すポインタに文字列string2をコピー
- char *strncpy(char *string1, char *string2, unsigned int n);
  - string1が示すポインタに文字列string2の先頭からn文字をコピー
- unsigned int strcspn(char *string1, char *string2);
  - string1の中で，string2中のどれか1文字が最初に見つかった位置を返す．
- unsigned int strlen(char *string);
  - 文字列stringの長さを返す
- char *strtok(char *string1, char string2);
  - string2の各要素の文字をデリミタとして，文字列string1からデリミタにより区切られた文字を取り出す．string1の代わりにNULLを指定すると，次のトークンのポインタが返される．

乱数の使い方の一例

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <conio.h>

int main(int argc, char* argv[])
{
    int i;

    printf("Random: default¥n");
    for (i = 0; i < 10; i++) {
        printf("%2d ", rand() % 100);
    }
    printf("¥n¥n");

    printf("Random: srand constant¥n");
    srand(4649);
    for (i = 0; i < 10; i++) {
        printf("%2d ", rand() % 100);
    }
    printf("¥n¥n");

    printf("Random: srand time¥n");
    srand((unsigned)time(NULL));
    for (i = 0; i < 10; i++) {
        printf("%2d ", rand() % 100);
    }
    printf("¥n¥n");

    printf("time = %d¥n¥n", (int)time(NULL));

    printf("Press any key¥n");
    _getch();

    return 0;
}
```

コメント

- /* この中がコメント */
- // これ以降，改行するまでがコメント(C言語ではなくC++のコメント機能)

変数の中身には想定通りの値が入っていますか？

- 例えば
  a = 1;
  a = 2;
  というプログラムがあったとします．変数aに1という数値が入っている場合に正しく動作するものとします．このとき，このプログラムの場合，a=2;という行がバグの原因，ということになります．エラーメッセージは出ないからバグを見つけるのは難しいですよね．
- 全ての変数に正しい値が入っているでしょうか．その時々で適切な値というものは変わってくると思います．for文の1つ1つのループそれぞれで，全ての変数に適切な値が入っているでしょうか．if文による分岐や，関数呼び出しの引数なども考慮して変数の中身を想像してみてください．ポインタ型かそうでないか，関数の引数の中身は変化するかしないか，なども確認してみてください．配列の添え字の値は適切でしょうか．プログラムを最初から最後まで実行することを想像しながら，そのときそのときで全ての変数に想定通りの値が入っているかどうか，ソースコードを1行1行，頭の中で実行しながら想像してみてください．
- デバッガ(この場合はVisual Studio)の機能で，実行時の変数の値を調べることができます．printf文でそのときそのときの変数の値を出力してもいいでしょう．配列の値を調べるのが難しいようであれば，ファイルに出力してもいいと思います．「この変数があやしい」「ここがあやしい」とピンポイントで候補場所を絞って調べることで効率化することもできます．

C言語による文字列処理のヒント

- あらかじめchar line[256]のように定義しておき，ファイルポインタ（例えばfp）を設定したあと，fgets(line, 256, fp)を実行すると，ファイルから一行読み込み，それを変数lineに入れてくれる

- 以降の例ではstr, s1, s2, s3, s4はchar[256]型変数，i1, i2, i3, i4はint型変数，f1, f2, f3, f4はfloat型変数，d1, d2, d3, d4はdouble型変数，sはchar*型変数とする

- strに"a b"という文字列が入っているときにsscanf(str, "%s %s %s", s1, s2, s3)を実行すると，s1には"a"，s2には"b"という文字列が入り，s3は何も変化しない
- strに"1 2/3 4 5//6"という文字列が入っているときにsscanf(str, "%d %d %d %d", &i1, &i2, &i3, &i4)を実行すると，i1には1，i2には2という数値が入り，i3とi4は何も変化しない
- strに"1 2/3 4 5//6"という文字列が入っているときにsscanf(str, "%s %s %s %s", s1, s2, s3, s4)を実行すると，s1には"1"，s2には"2/3"，s3には"4"，s4には"5//6"という文字列が入る
- s1に"1"という文字列が入っているときにsscanf(s1, "%d", &i1)を実行すると，i1に1という数値が入る
- s2に"2/3"という文字列が入っているときにsscanf(s2, "%d", &i2)を実行すると，i2に2という数値が入る
- s3に"4"という文字列が入っているときにi3 = atoi(s3)を実行すると，i3に4という数値が入る
- s4に"5//6"という文字列が入っているときにi4 = atoi(s4)を実行すると，i4に5という数値が入る
- strに"111.1 222.2 333.3"という文字列が入っているときにsscanf(str, "%f %f %f %f", &f1, &f2, &f3, &f4)を実行すると，f1に111.1，f2に222.2，f3に333.3という数値が入り，f4は何も変化しない
- strに"111.1 222.2 333.3"という文字列が入っているときにsscanf(str, "%lf %lf %lf %lf", &d1, &d2, &d3, &d4)を実行すると，d1に111.1，d2に222.2，d3に333.3という数値が入り，d4は何も変化しない
- strに"111.1 222.2 333.3"という文字列が入っているときにsscanf(str, "%s %s %s %s", &s1, &s2, &s3, &s4)を実行すると，s1に"111.1"，s2に"222.2"，s3に"333.3"という文字列が入り，s4は何も変化しない
- s1に"111.1"という文字列が入っているときにsscanf(s1, "%f", &f1)を実行すると，f1に111.1という数値が入る
- s2に"222.2"という文字列が入っているときにsscanf(s2, "%lf", &d2)を実行すると，d2に222.2という数値が入る
- s3に"333.3"という文字列が入っているときにf1 = (float)atof(s3)を実行すると，f3に333.3という数値が入る
- s4に"abc"という文字列が入っているときにd4 = atof(s4)を実行すると，d4に0という数値が入る
- s1に"vn"という文字列が入っていてs2に"vn"という文字列が入っているとき，strcmp(s1, s2)は0を返す
- s1に"vn"という文字列が入っていてs3に"v"という文字列が入っているとき，strcmp(s1, s3)は1を返す
- s1に"v "という文字列が入っていてs2に"vn 1.1"という文字列が入っているとき，strncmp(s1, s2, 2)は-1を返す
- s1に"v "という文字列が入っていてs3に"v 1.1"という文字列が入っているとき，strncmp(s1, s3, 2)は0を返す
- strに"f 1"という文字列が入っているとき，sscanf(&str[2], "%d", &i1)を実行すると，i1に1という数値が入る
- strに"f 1"という文字列が入っているとき，sscanf(str + 2, "%d", &i1)を実行すると，i1に1という数値が入る
- strに"f 1/1/1 2/2/2"という文字列が入っているとき，s = strchr(str, ' ')を実行すると，sが参照する文字列は" 1/1/1 2/2/2"になる
- sが" 1/1/1 2/2/2"という文字列を参照しているとき，s++を実行すると，sが参照する文字列は"1/1/1 2/2/2"になる