Cプログラミング

第4回
連立一次方程式の解法(2)

青柳滋己
aoyagi@aoni.waseda.jp

本日の講義内容

- 文字，文字列の復習
- 動的メモリ割り当て
- 連立一次方程式の解法(2)
  - 部分ピボット選択付きガウス消去法
- 演習

文字(復習)

- 文字を扱うには，char型(1byte)を使う.
- char型の変数一つに，文字一つを記憶できる.
- 文字は'A'，'B'，'c'のように' 'で囲む.
- 文字は実際にどのような数値になっているかは，man ascii で見ることができる.
- scanf, printfでの文字の入出力では，%c を使用.
- この講義では半角文字のみを扱う

```
char ch:

ch='A';      /* char型変数chにAという文字を代入 */
scanf("%c", &ch);  /* 一文字入力 */
printf("%c", ch);  /* 一文字出力 */
```

ASCIIコード表(復習)

- 文字は下表の数字として扱う

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x0	¥0 (NUL)	(制御文字)	空白文字	0	@	P	`	p
0x1	(制御文字)	(制御文字)	!	1	A	Q	a	q
0x2	(制御文字)	(制御文字)	"	2	B	R	b	r
0x3	(制御文字)	(制御文字)	#	3	C	S	c	s
0x4	(制御文字)	(制御文字)	$	4	D	T	d	t
0x5	(制御文字)	(制御文字)	%	5	E	U	e	u
0x6	(制御文字)	(制御文字)	&	6	F	V	f	v
0x7	(制御文字)	(制御文字)	'	7	G	W	g	w
0x8	(制御文字)	(制御文字)	(	8	H	X	h	x
0x9	¥t (HT)	(制御文字)	)	9	I	Y	i	y
0xa	¥n (LF)	(制御文字)	*	:	J	Z	j	z
0xb	(制御文字)	ESC	+	;	K	[	k	{
0xc	(制御文字)	(制御文字)	,	<	L	＼	l	|
0xd	CR	(制御文字)	-	=	M	]	m	}
0xe	(制御文字)	(制御文字)	.	>	N	^	n	~
0xf	(制御文字)	(制御文字)	/	?	O	_	o	(制御文字)

文字リテラル(復習)

- 文字列を""で囲んだものを文字リテラルと呼ぶ．基本的に定数扱い．

  例
  ```
  "abcdef"
  "9876543210"
  ```

- 文字リテラルの内部表現

  例
  ```
  "abcdef"  →  'a'  'b'  'c'  'd'  'e'  'f'  '¥0'
               0x61 0x62 0x63 0x64 0x65 0x66 0x0
  ```

最後に終端を意味する'¥0'が付加される

文字列(復習)

- 文字列はchar型の配列として扱う.
- どこまでが文字列なのか，判別するために終端文字'¥0'を使う.
- 表示にはprintf を使う際には，"%s"で指定する.
- 文字列読み込みにはscanfではなく，fgetsで1行全部読み込むのが良い.
- 関数引数にする場合はポインタを使う.

```
char str[50], str1[100]:

str[0]='A';
str[1]='b';
str[2]='c';
str[3]='¥0';
printf("%s",str);
scanf("%s", str1);
fgets(str1, 100, stdin);
printf("%s",str1);
```

文字列の初期化(復習)

- 文字定数，文字コードで初期化

```
char ss[5]={ 'a', 'b', 'c', 'd', '¥0'};
char ss[5]={  97,  98,  99, 100, '¥0'};
```

- 文字リテラルで初期化

```
char ss[5]="abcd";
char ss[] ="abcd"; /* 配列の添字は省略可能 */
```

配列は'¥0'の文字分も含めて確保すること.

配列の方が十分大きいとき，残った後ろの部分は'¥0'で埋められる

文字列の操作(復習)

```
char ss[5]={ 'a', 'b', 'c', 'd', '¥0'};

ss[]="opq";   /* これは初期化時以外は駄目 */
ss[0]='o', ss[1]='p'; ss[2]='q'; ss[3]='¥0';
/* あるいは strcpy を使う */
strcpy(ss,"opq");
```

- 文字列のコピーは配列のコピーになるので，各要素を一つずつ代入するか，あるいはstrcpyを使う必要がある.
- strcpyを使うときは終端に'¥0'が付加されるのも考慮し，十分大きな配列を用意すること.

ポインタと文字列

- 次の2つは意味が異なるので注意

```
char ss[]="abcdef";
char *p="abc123";
```

- 上はssという名前の文字列(配列)，下は"abc123"という文字リテラルへのポインタ．

  したがって，

```
{ char *q=ss;
  *q='g';
}
```

のように配列の中身の変更は可能だが，

```
*p = 'g';
```

のように文字リテラルの中身の変更ができるかどうかは処理系依存

文字や文字列を扱う関数(復習)

- 文字
  #include <ctype.h>
  isalpha, isdigit, islower, isupperなど.
- 文字列
  #include <string.h>
  strlen, strcpy, strcat, strcmpなど.
- 文字と入出力
  getchar, ~~gets~~, putchar, puts,
  fgetc, fgets, fputc, fputsなど.
- その他
  #include <stdlib.h>
  atoi, atofなど

注：**getsは使用してはいけない．** 代わりにfgetsを使用すること.
※ **manコマンドで使い方を確認すること.**

getsとfgetsの違い

- getsは文字配列の大きさを指定できないため，長い入力行があったときにはプログラムが異常終了してセキュリティホールとなる．したがって，getsは使わずにfgetsを使うこと．
- 同様の理由で sprintf も要注意． snprintf 等を使うのが望ましい．
- 同様に文字列読み込みにscanfを使わないこと．
- getsと異なり，fgetsの場合には行末の'¥n'が文字列に含まれてしまう．含まれないようにしたい場合は自分で削除する．

fgets，fputsを使ったファイルコピーの例

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

#define MAXLINE 256

int main(int argc, char *argv[])
{
  FILE *fp, *fp1;
  char ss[MAXLINE];
  int len;

  if (argc != 3) {
    printf("Usage: %s <src> <dest>¥n", argv[0]);
    exit(1);
  }
  if ((fp=fopen(argv[1], "r")) == NULL) {
    printf("Cannot open %s¥n", argv[1]);
    exit(1);
  }
  if ((fp1=fopen(argv[2], "w")) == NULL) {
    printf("Cannot open %s¥n", argv[2]);
    exit(1);
  }
```

```
  while (fgets(ss, MAXLINE, fp) != NULL)
     fputs(ss, fp1);
  fclose(fp);
  fclose(fp1);
  return 0;
}
```

コマンドライン文字列の取得法

- main関数の引数を右のサンプルのように書くことによって，プログラム起動時のコマンドラインの文字列がプログラム中で使用できる．
- 慣習で，第一引数はargc，第二引数はargvを使う．
- コマンドラインオプション等の処理が可能になる．
- argcに文字列の個数，argv[0]にコマンド名，argv[1]以降に引数が格納される．

commandline.c

```
#include <stdio.h>

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

  printf("argc=%d¥n", argc);
  for (i=0; i<argc; i++)
     printf("%s¥n", argv[i]);

  return 0;
}
```

実行例

```
% gcc commandline.c -o commandline
% ./commandline -t abc
argc=3          /* argc: 文字列は3つ */
./commandline   /* argv[0]を表示 */
-t              /* argv[1]を表示 */
abc             /* argv[2]を表示 */
 %
```

sizeof演算子

```
sizeof(単項式);

sizeof(型名);
```

```
#include <stdlib.h>
#define MAX 3
.....
int *a;
.....
  a=(int *)malloc(MAX*sizeof(int));
  if (a != NULL) {
    a[0]=0; a[1]=1; a[2]=2;
  }
```

- 引数で示されたオブジェクトのサイズをバイト数で返す.
- 単項式には変数や配列，構造体などの名前を指定できる.
- malloc等動的にメモリ確保する際によく用いる.
- 式が評価されないので注意

  例: sizeof(n++); のn++は評価されない

動的メモリ割り当て(1/2)

- mallocを使う．引数に確保したいメモリのバイト数を指定する．
- 確保に成功すると先頭のアドレスが返ってくる．確保に失敗するとNULLが代入されるので，必ずチェックする．
- ヒープ領域にメモリが確保される．
- 使わなくなったら，freeを使って開放してやること．
- 必要なサイズに応じた効率よいメモリ確保ができる．

```
#include <stdlib.h>

void *malloc(size_t size);
void free(void *ptr);
```

```
#include <stdlib.h>
#define MAX 3
.....
int *a,i;
double *data;
.....
  a=(int *)malloc(sizeof(int));
  if (a == NULL) {
    exit(1);
  }
  data=(double *)malloc(MAX*sizeof(double));
  if (data == NULL) ......
  for (i=0;i<MAX;i++) {
    data[i]=.......
  }
```

動的メモリ割り当て(2/2)

- 例えばint型など，システムやコンパイラによって何バイトか異なるものがある.
- 互換性を保つためにsizeof演算子を使うと良い.
- mallocで確保されたメモリは，必要に応じて初期化すること

```
typedef struct
{
  char id[4];
  int size;
....
} HEADER;

int main(void)
{
  char buffer *buf;
  HEADER *header;
  .....
  header=(HEADER *)malloc(sizeof(HEADER));
  if (header == NULL) {
    printf("cannot allocate memory.¥n");
    exit(1);
  }
  .....
  free(header); /* 使い終わったら開放 */
  .....
}
```

メモリ操作関数

- メモリを扱う関数に，memsetやmemcpyなどがある．

```
#include <string.h>

void *memset(void *s, int c, size_t n);
void *memcpy(void *dest, void *src, size_t n);
void *memmove(void *dest, void *src, size_t n);
```

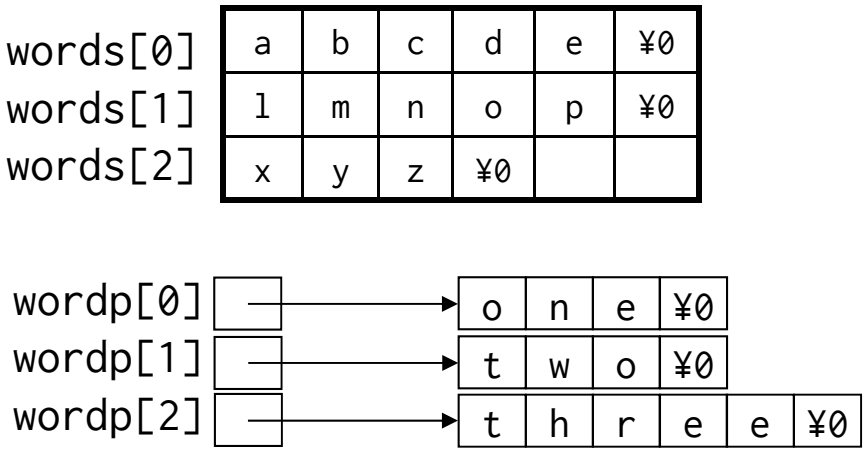
- 配列はメモリ上にまとめて確保されているので，これらの関数が使える．
- 詳細はmanで調べること．

```
typedef struct
{
....
} HEADER;

int main(void)
{
  char buffer *buf;
  HEADER *header;
  .....
  header=(HEADER *)malloc(sizeof(HEADER))
  if (header == NULL) {
    printf("cannot allocate memory.¥n");
    exit(1);
  }
  memset(header, 0, sizeof(HEADER));
  .....
}
```

ポインタの配列

- 右のwordsは文字列の配列．char型2次元配列が確保され，その中に文字列も入れられる．
- wordpは文字リテラルへのポインタの配列．文字リテラルはどこに確保されるかわからない．

```
char words[3][6] = {
    "abcde",
    "lmnop",
    "xyz"
};


char *wordp[3]={
    "one",
    "two",
    "three"
};
```

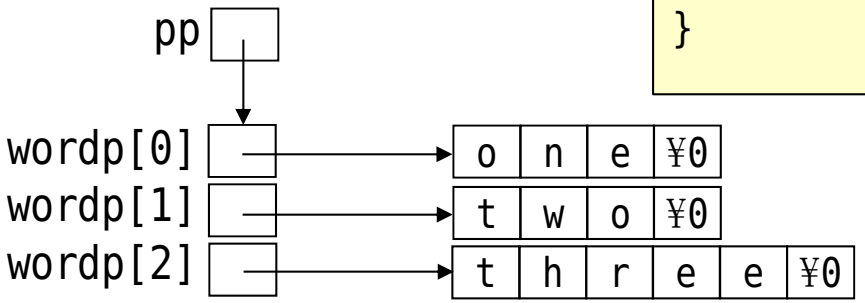


※コマンドライン文字列取得時の
`int main(int argc, char *argv[]) {..}`
における argv は左の wordp と同様の
文字リテラルへのポインタの配列．

ポインタのポインタ

- ポインタ変数を指すポインタ変数を作ることが可能. 変数宣言の際，変数名の前に"**"をつける.
- 右の `char **pp;` では，char型ポインタ変数を指すポインタ変数が1つ確保される.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    char *wordp[3]={
         "one",
         "two",
         "three"
    };
    char **pp;

    pp=wordp;
    printf("%s¥n", *(pp+1));
    .......

}
```



ガウス消去法･前進消去(1/2) (復習)

- 第1列の2行目以降を0にする
  - 第1行×(-5/3)＋第2行

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & -\frac{2}{3} & -\frac{1}{3} \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ -\frac{5}{3} \\ 13 \end{pmatrix}$$

  - 第1行×(-4/3)＋第3行

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & -\frac{2}{3} & -\frac{1}{3} \\ 0 & \frac{2}{3} & -\frac{5}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ -\frac{5}{3} \\ -\frac{13}{3} \end{pmatrix}$$

$$\begin{pmatrix} 3 & 1 & 2 \\ 5 & 1 & 3 \\ 4 & 2 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ 20 \\ 13 \end{pmatrix}$$

前進消去(2/2) (復習)

- 第2列の2行目以降を0にする
  - 第2行×1＋第3行

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & -\frac{2}{3} & -\frac{1}{3} \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ -\frac{5}{3} \\ -6 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & -\frac{2}{3} & -\frac{1}{3} \\ 0 & \frac{2}{3} & -\frac{5}{3} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ -\frac{5}{3} \\ -\frac{13}{3} \end{pmatrix}$$

後退代入(復習)

$$\begin{pmatrix} 3 & 1 & 2 \\ 0 & -\frac{2}{3} & -\frac{1}{3} \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 13 \\ -\frac{5}{3} \\ -6 \end{pmatrix}$$

- x_3, x_2, x_1 の順に代入して答えを求める
  - 第3行目より

$$x_3 = \frac{-6}{-2} = 3$$

  - 第2行目より

$$x_2 = -\frac{3}{2}\left(-\frac{5}{3} + \frac{1}{3} \times 3\right) = 1$$

  - 第1行目より

$$x_1 = \frac{1}{3}(13 - 1 \times 1 - 2 \times 3) = 2$$

ガウス消去法アルゴリズム(復習)

前進消去

```
for k=1,2,...,N-1
   for i=k+1,k+2,...,N
      alpha = - a[i][k] / a[k][k]
      for j=k+1,....N
         a[i][j]=a[i][j]+alpha*a[k][j]
      end
      b[i]=b[i]+tmp*b[k]
    end
end
```

後退代入

```
b[N]=b[N]/a[N][N]

for k=N-1,...,1

   b[k]=(b[k]-Σ_{j=k+1}^{N}(a[k][j]*b[j])) / a[k][k]

end
```

前回の演習の解答例(1/3)

kadai3_1.c

```
.......

void gauss(double a[][N+1], double b[])
{
  int i,j,k;
  double alpha, tmp;

  /* 前進消去 */
  for (k=1; k<=N-1; k++) {
    for (i=k+1; i<=N; i++) {
      alpha=-a[i][k]/a[k][k];
      for (j=k+1; j<=N; j++) {
          a[i][j] = a[i][j]+alpha*a[k][j];
      }
      b[i] = b[i]+ alpha*b[k];
    }
  }

/* 右へ続く */
```

kadai3_1.c

```
/* 後退代入 */

  b[N] /= a[N][N];
  for (k=N-1; k>=1; k--) {
    tmp=b[k];
    for (j=k+1; j<=N; j++) {
      tmp = tmp-a[k][j]*b[j];
    }
    b[k] = tmp/a[k][k];
  }
}
```

演習3-1 補足

- ローカル変数は参照前に必ず初期化すること！

kadai3_1.c

```
/* 後退代入 */

  b[N] /= a[N][N];
  for (k=N-1; k>=1; k--) {
    tmp=0.0;
    for (j=k+1; j<=N; j++) {
      tmp += a[k][j]*b[j];
    }
    b[k] = (b[k]-tmp)/a[k][k];
  }
}
```

前回の演習の解答例(2/3)

kadai3_2.c

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

#define SIZE 100
#define RESULT_FILE "result.txt"

int main(void)
{
  int num, i;
  FILE *fp;

  srand((unsigned int)time(NULL));
  if ((fp=fopen(RESULT_FILE, "w")) == NULL) {
    printf("Cannot open output file. Exit.¥n");
    exit(1);
  }

/* 右へ続く */
```

kadai3_2.c

```
  for (i=0; i<SIZE; i++) {
    num=(int)(rand( ) / (RAND_MAX+1.0) * 6) + 1;
    fprintf(fp, "%d¥n", num);
  }
  fclose(fp);
  return 0;
}
```

演習3-2 補足

- fopenが失敗した時はプログラムを終了すること.
- #includeについて

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

システムのstdio.hを探し，なければカレントディレクトリのstdio.hを探す.

```
#include "stdio.h"
```

カレントディレクトリのstdio.hを探し，なければシステムのstdio.hを探す.

kadai3_2.c

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

#define SIZE 100
#define RESULT_FILE "result.txt"

int main(void)
{
  int num, i;
  FILE *fp;

  srand((unsigned int)time(NULL));
  if ((fp=fopen(RESULT_FILE, "w")) == NULL) {
    printf("Cannot open output file. Exit.¥n");
    exit(1);
  }
  for (i=0; i<SIZE; i++) {
    num=(int)(rand() / (RAND_MAX+1.0) * 6) + 1;
    fprintf(fp, "%d¥n", num);
  }
```

time関数以外を使う(参考)

- time関数は秒単位なので，場合によっては同じ結果になる場合がある.
- 例えば，gettimeofdayを使うとマイクロ秒まで求められる(Linuxのみ)
- 右の例は画面出力のみ

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <limits.h>
#define MAX 100

int main(void)
{
  int num, i;
  long tm;
  struct timeval tv;

  gettimeofday(&tv,NULL);
  tm=(long)tv.tv_sec*1000000
      + tv.tv.usec;
  srand(tm % UINT_MAX);
  for (i=0; i<MAX; i++) {
    num=(int)(rand() / (RAND_MAX+1.0) * 6) + 1;
    printf("%d¥n", num);
  }

  return 0;
}
```

前回の演習の解答例(3/3)

kadai3_3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 100
#define MAXNUM 6
#define INPUT_FILE "result.txt"

int main(void) {
  int i, val;
  int histgram[MAXNUM]; /* 出現個数記憶用 */
  FILE *fp;

  /* initialize */
  memset(histgram, 0, MAXNUM*sizeof(int));

  /* open file */
  if ((fp=fopen(INPUT_FILE, "r")) == NULL) {
    printf("Cannot open input file. Exit.¥n");
    exit(1);
  }
/* 右へ続く */
```

kadai3_3.c

```
  for (i=0;i<SIZE; i++) {
    fscanf(fp, "%d", &val);
    histgram[val-1]++;
  }
  fclose(fp);
  for (i=0; i<MAXNUM; i++)
    printf("%d: %d¥n", i+1, histgram[i]);

  return 0;
}
```

演習3-3 補足

kadai3_3.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE 100
#define MAXNUM 6
#define INPUT_FILE "result.txt"

int main(void) {
  int i, val;
  int histgram[MAXNUM];
  FILE *fp;

  /* initialize */
  memset(histgram, 0, MAXNUM*sizeof(int));

  /* open file */
  if ((fp=fopen(INPUT_FILE, "r")) == NULL) {
    printf("Cannot open input file. Exit.¥n");
    exit(1);
  }
.....
```

同じ値を連続して代入する場合，1文で書ける

```
int a,b,c,d,e,f;

a=b=c=d=e=f=0;
```

配列や構造体等の初期化にはmemset を使うと簡単.

部分ピボット選択付きガウス消去法(1/3)

- ガウス消去法では対角成分が0となるものは解けない

$$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 3 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 2 \\ -3 \end{pmatrix}$$

前進消去の際，対角成分a[k][k]は絶対値が最大となるものに行全体を交換する．上の例ではまず第1列の最大値は3．よって第3行と第1行を入れ替える．

$$\begin{pmatrix} 3 & 1 & 0 \\ 1 & 0 & 3 \\ 0 & 1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ 2 \end{pmatrix}$$

部分ピボット選択付きガウス消去法(2/3)

$$\begin{pmatrix} 3 & 1 & 0 \\ 1 & 0 & 3 \\ 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ 2 \end{pmatrix}$$

- 第1列について前進消去を行う

$$\begin{pmatrix} 3 & 1 & 0 \\ 0 & -\frac{1}{3} & 3 \\ 0 & 1 & 2 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 3 \\ 2 \end{pmatrix}$$

- 第2列については第2行目以降について絶対値が最大なものを探して第2行目と入れ替える.

$$\begin{pmatrix} 3 & 1 & 0 \\ 0 & 1 & 2 \\ 0 & -\frac{1}{3} & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ 3 \end{pmatrix}$$

部分ピボット選択付きガウス消去法(3/3)

$$\begin{pmatrix} 3 & 1 & 0 \\ 0 & 1 & 2 \\ 0 & -\frac{1}{3} & 3 \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ 3 \end{pmatrix}$$

- 第2列について前進消去を行う

$$\begin{pmatrix} 3 & 1 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & \frac{11}{3} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} -3 \\ 2 \\ \frac{11}{3} \end{pmatrix}$$

- 以降は後退代入を行い解を求める

次回予定

- 非線形方程式の解法(1)
  - 2分法

本日の演習(1/3)

- 前回作成したガウス消去法プログラムに次の改良を加えよ(kadai4_1.c).
  - 連立N元一次方程式に対応
  - ファイルからデータを読み込む
  - データファイル名をコマンドラインから指定

```
% gcc kadai4_1.c -o kadai4_1
% ./kadai4_1 data1.txt
Answer:
-2.000000 -1.000000 1.000000 2.000000
%
```

注意：データファイルの構造は，$Ax=b$ の連立方程式の時，

Aの次数
Aの中身
bの中身

とする.

data1.txt

次数 → 4

1.0 2.0 1.0 1.0
A → 4.0 5.0 -2.0 4.0
4.0 3.0 -3.0 1.0
2.0 1.0 1.0 3.0

b → -1.0 -7.0 -12.0 2.0

データファイルの例

本日の演習(2/3)

- 次頁のkadai4_1.cを基に部分ピボット選択機能を追加したkadai4_2.cを作成し，次の連立方程式を解け.

$$\begin{cases} 2x_1 + 4x_2 + x_3 - 3x_4 = 0 \\ -x_1 - 2x_2 + 2x_3 + 4x_4 = 10 \\ 4x_1 + 2x_2 - 3x_3 + 5x_4 = 2 \\ 5x_1 - 4x_2 - 3x_3 + x_4 = 6 \end{cases}$$

参考

kadai4_2.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void gauss2(double **a, double *b, int N);

int main(int argc, char *argv[])
{
  double **mat1, *mat2;
  FILE *fp;
  int i,j,N;

  if ((fp=fopen(..... )) == NULL) { /* open file */
    printf("cannot open %s¥n", argv[1]);
    exit(1);
  }
  fscanf(fp, "%d", &N); /* 次数読み込み */
  if ((mat1=(double **)malloc((N+1)*sizeof(double *)))
      == NULL) {
  ................
  }
  /* 右へ続く */
```

kadai4_2.c

```
 /* 行列のメモリ確保とファイルからの */
 /* データの読み込み */

  gauss2(mat1, mat2, N);

  printf("¥nAnswer:¥n"); /* 解の表示 */
  for (i=1; i<=N; i++)
        printf("%f ", mat2[i]);
  printf("¥n");
  return 0;
}

void gauss2(double **a, double *b, int N)
{
  /* ピボット選択 */
  /* 前進消去 */

  /* 後退代入 */

}
```

本日の演習(3/3)

- 文字列を引数に取り，英小文字があればそれを英大文字に書き換えるupperという関数を作成せよ．また，文字列中の英大文字を英小文字に書き換えるlowerという関数を作成せよ．

  例:　upper("123abcDEfg") → "123ABCDEFG"

  lower("123abcDEfg") → "123abcdefg"

※ toupper,tolower等は使わないこと．kadai4_3.cに関数を入れて提出．

※ Course N@viにてプログラムを提出

(締め切り：本日中)

演習のヒント

- Cでは二次元以上の動的な配列は関数引数にできない．(コンパイル時，配列引数の一次元目は定数でなければならない)
- 一次元の配列を複数作り，その一次元配列へのポインタ変数の配列と次数を引数にする．

kadai3_1c

```
void gauss(double a[][N+1], double b[]){...}
```

kadai4_2.c

```
void gauss2(double **a, double *b, int N){...}
```