mybasicのマニュアル

ver 8.4

作者　秋山 正和 (北大電子科学研究所)

はじめに

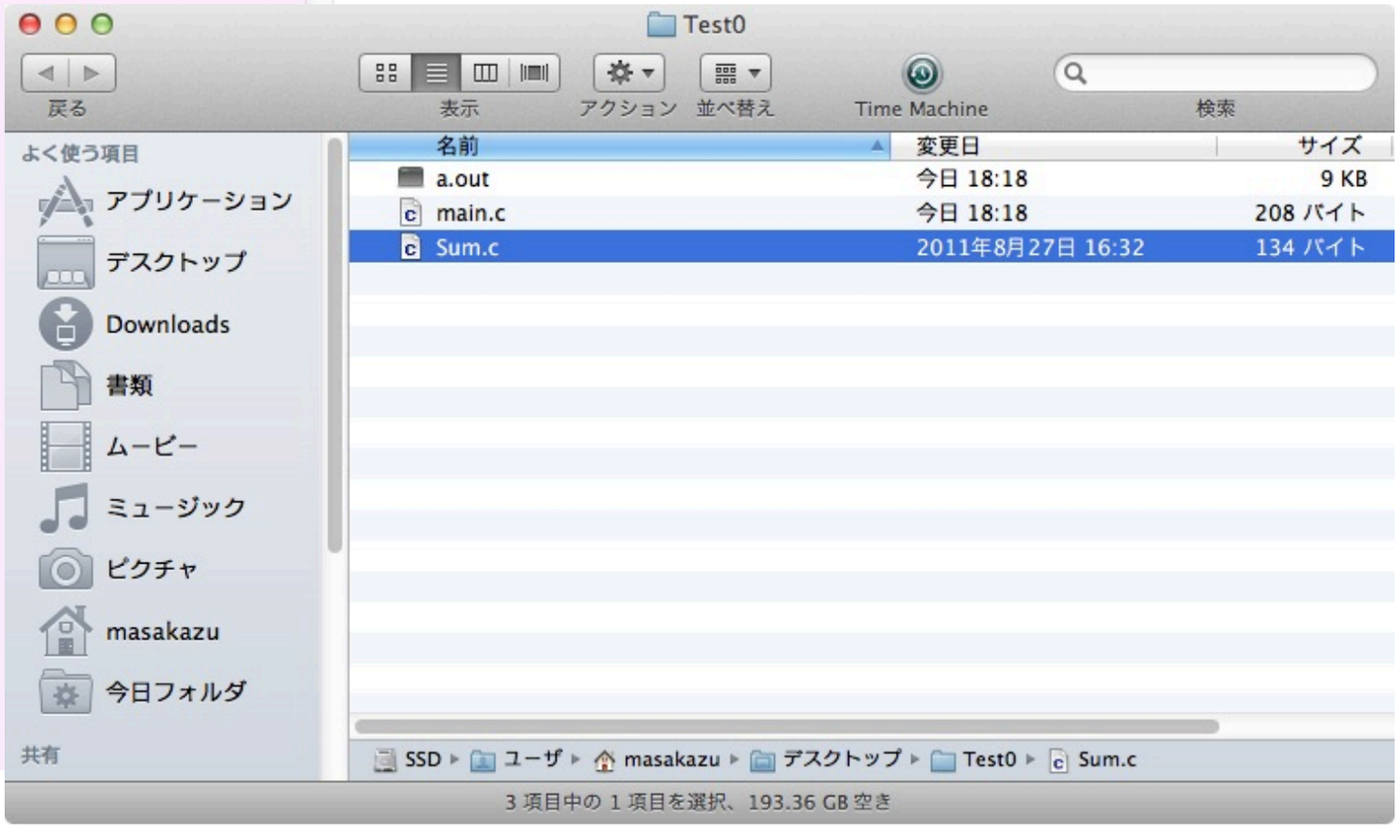
mybasicは作者が学生時代の頃から数値計算で便利だと思えるものを集め改良を重ね関数群としてまとめたものである．ソースファイル単体で使うもよし，ライブラリとして使うもよしである．ただ便利で高速である反面，使い方には十分注意せねばならない．本マニュアルをしっかり読み，理解すればあなたの数値計算ライフはより明るくなるだろう．

インストール

ソースファイル単体で使う場合は自身のプログラム（ここではmain.cとする．）に以下のようにincludeすればよい．(サンプルはTest0/)

main.c

```
#include <stdio.h>

#include "Sum.c"

int main(void)
{
    double u[3];
    double  wa;

    u[0] = -1;
    u[1] = 0;
    u[2] = 1;

    Sum(3,u,&wa);
    printf("wa = %f\n",wa);

    return 0;
}
```



インストール

ライブラリを作成する場合(作者はこちらを推奨)は以下のようにする.

1 mybasicx.yに移動する.

2 makeとうち，libbasic.aが出来るのを確認する.

3 出来上がったlibbasic.aを適当なフォルダ(homeにlibフォルダを作ることを推奨)に移す.

4 basic.hを適当なフォルダ(homeにincludeフォルダを作ることを推奨)に移す.

インストール

標準的な環境ではMakefileを変更する必要はないが高速化を望むならばMakefile.fastを使うほうがよい．(make -f Makefile.fastとせよ)

使い方

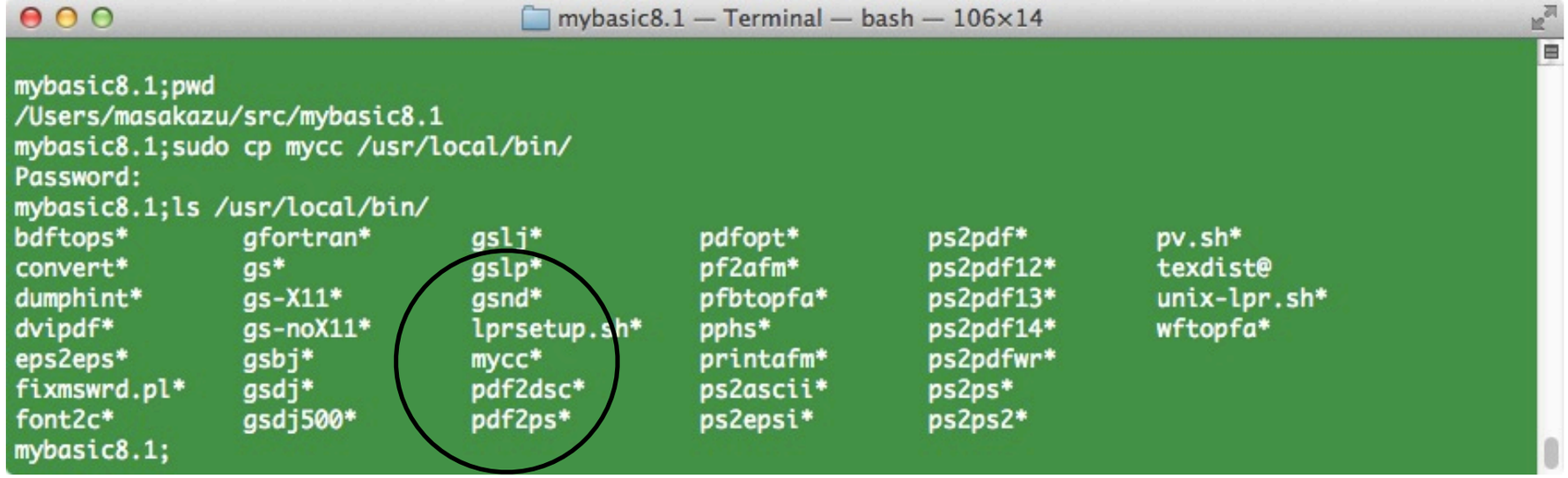
以下では，作者の推奨する方法を選択している場合について説明する.

(i)　専用のコマンド(mycc)を利用する場合.

(ii)　makeを利用する場合.

(i)　専用のコマンド(mycc)を利用する場合.

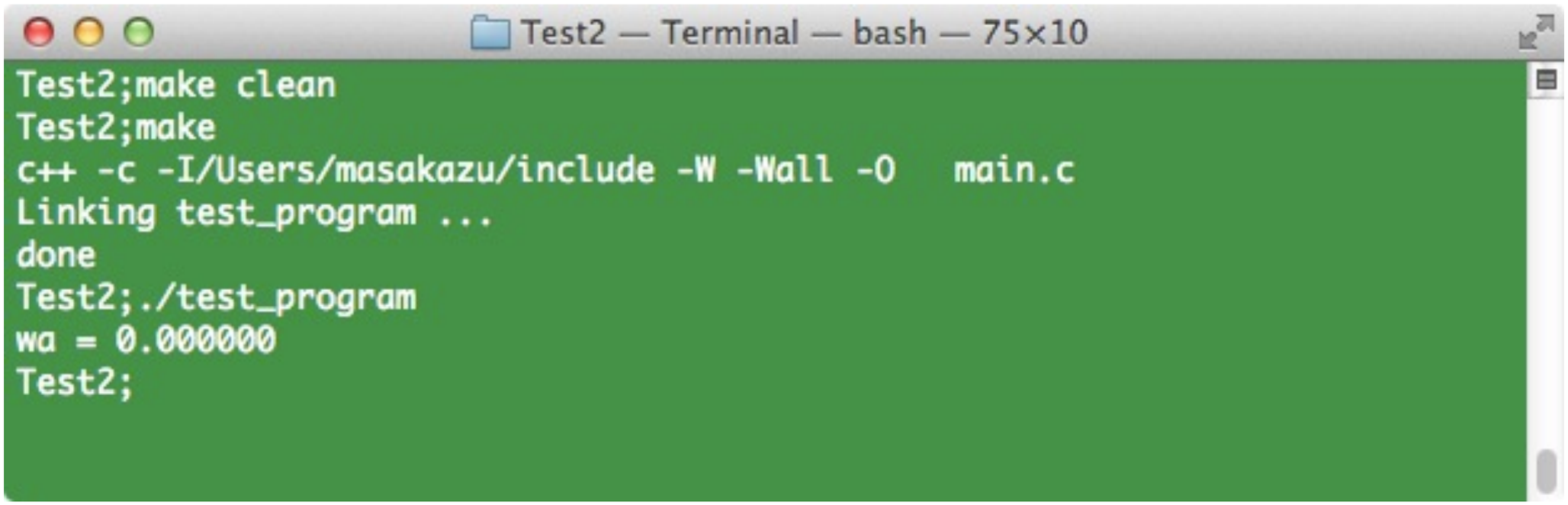
mybasicx.yフォルダにあるmyccファイルをパスの通ったフォルダに置く. Terminalに以下のように打てば/usr/local/binの下にmyccが置かれる.



Test1/ではmyccを試すことができる.

(ii)　makeを利用する場合.

実際上，高度なプログラムを書く場合はmakeによる実行ファイルの作成の方が効率が良い．Test2/ではmybasicを簡単に使うことができるようにMakefileがおいてある.



本makefileでは以下のような状況を仮定している.

✔ 実行ファイル名は常に「test_program」

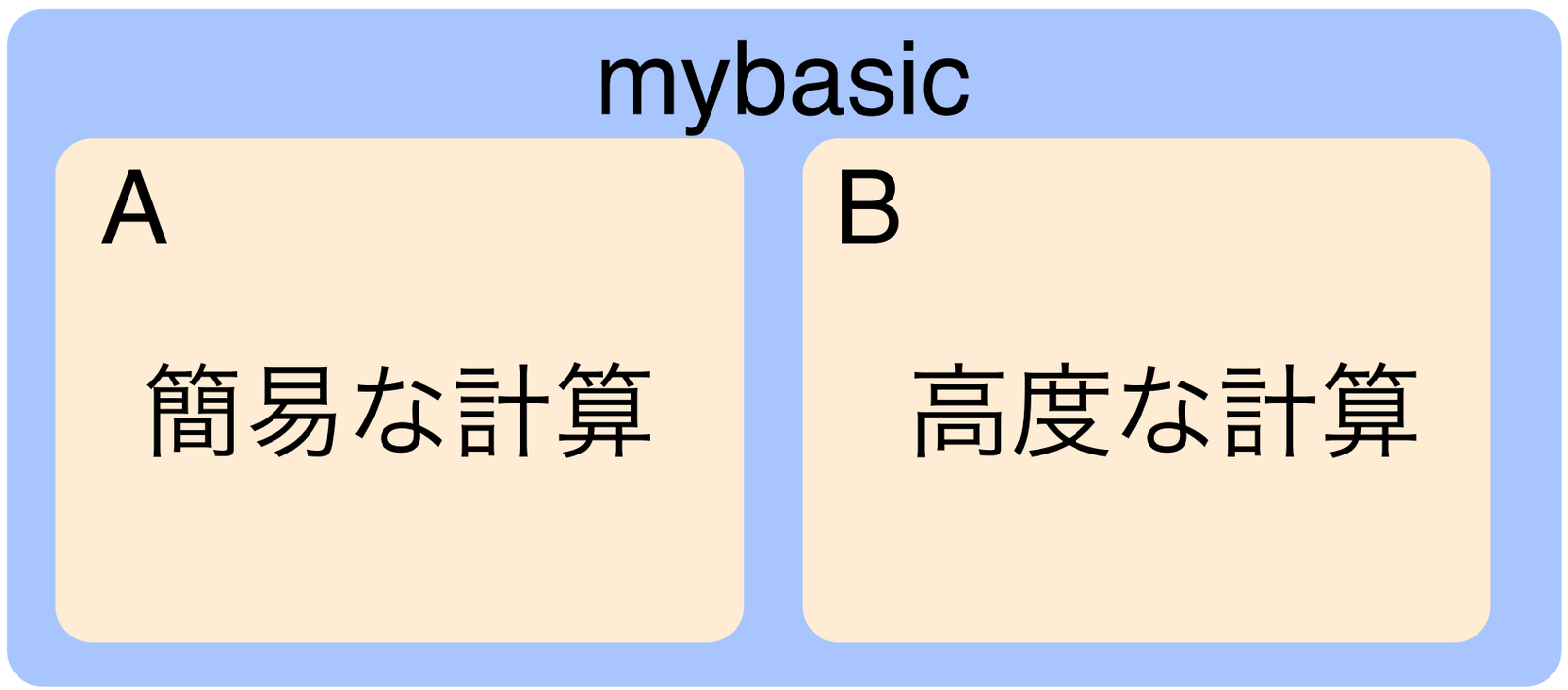
✔ メインプログラム名はmain.cもしくはmain.cppとする. mybasicはcでもc++でも使える.

✔ サブプログラムを追加する場合はMakefile.GENERICに記述する.

以上までを気をつければ単に「make ; ./test_program」とすれば実行できる.

サポートしている関数の説明

ここではmybasicがサポートする関数群を紹介する.
mybasicは 数値計算上で簡易な計算を行うための関数群Aと，やや高度なことを行うための関数群Bに分けることができる.以降ではそれぞれのグループごとに関数を紹介する.



A群

```
Dtpr(int n, double *u, double *v, double *norm)
```

サイズnの２つの配列u,vの内積を返す．結果はnormに入る.

使用例　Test3/

main.c

main.c

main.c 〉 No Selection

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

int main(void)
{
    double u[3],v[3];
    double  naiseki;

    u[0] = -1;
    u[1] = 0;
    u[2] = 1;
    v[0] = -1;
    v[1] = 0;
    v[2] = 1;

    Dtpr(3,u,v,&naiseki);
    printf("naiseki = %f\n",naiseki);

    return 0;
}
```

A群

```
InitializeReal(int n, double *u)
```

サイズnの配列uを0.0で初期化する． InitializeIntは0で初期化する.

main.c

main.c

main.c 〉 No Selection

```
1  #include <stdio.h>
2  #include <basic.h>
3
4  int main(void)
5  {
6      int i;
7      double u[3];
8
9      InitializeReal(3,u);
10
11     for(i = 0;i < 3;i++)
12         printf("%f\n",u[i]);
13     return 0;
14 }
15
```

使用例　Test4/

A群

```
Mean(int n, double *u, double *heikin)
```

サイズnの配列uの平均値をもとめる.

使用例　Test5/

main.c

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

int main(void)
{
    int i;
    double u[3],heikin;

    for(i = 0;i < 3;i++)
        u[i] = i;

    Mean(3,u,&heikin);
    printf("heikin = %f\n",heikin);

    return 0;
}
```

A群

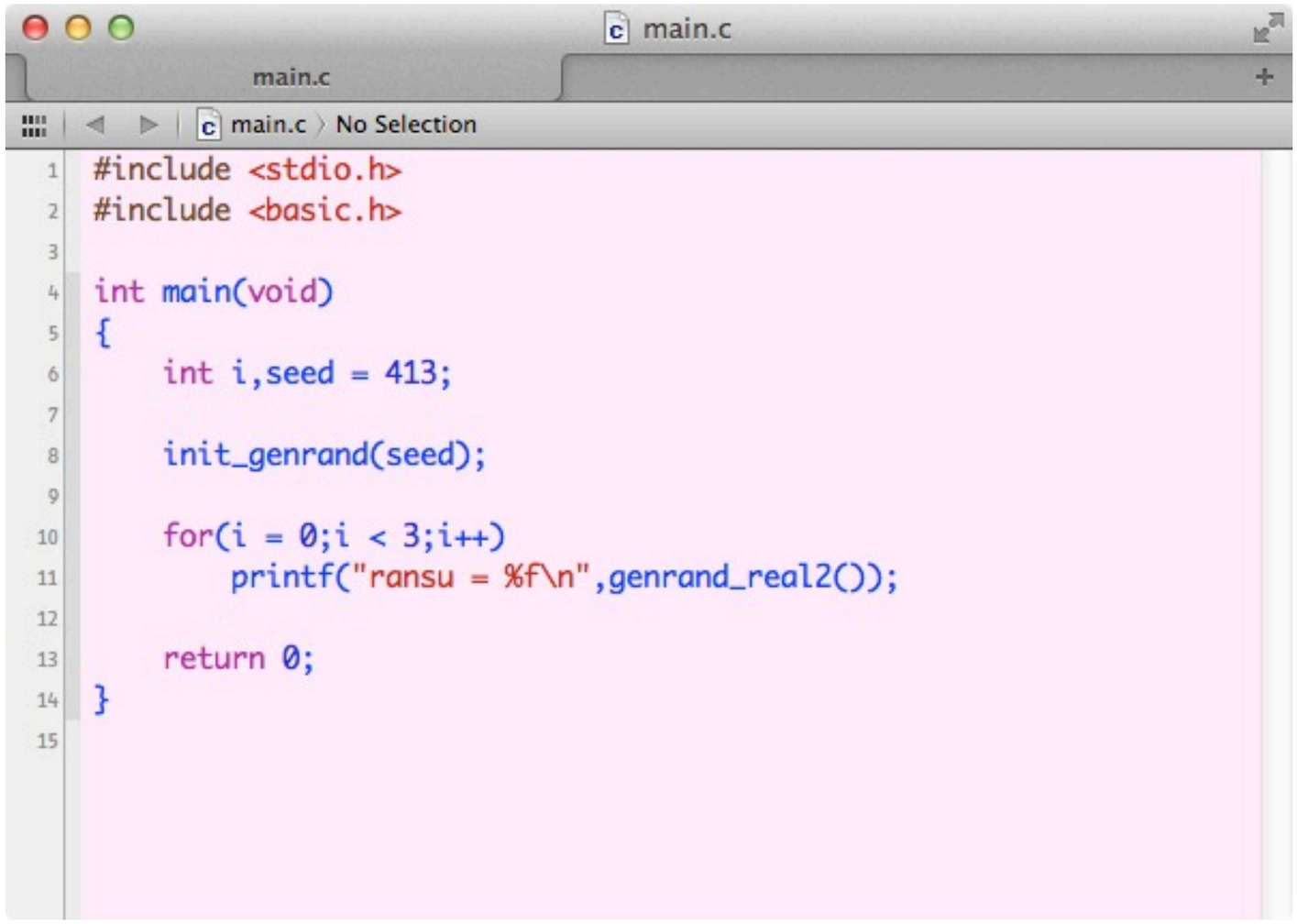
genrand_real2(void)

メルセンヌ・ツイスタ版の乱数発生関数である．使用例のようにすれば[0,1)内の乱数を発生させることができる.

関数名に64を付加すると64bit版のMTを使うことができる.

（init_genrand>init64_genrand,genrand_real2->genrand64_real2）．また，使用例だけではなくMT,MT64のすべての関数を使うことができる.

詳細は松本眞氏のHomePageを確認してほしい.



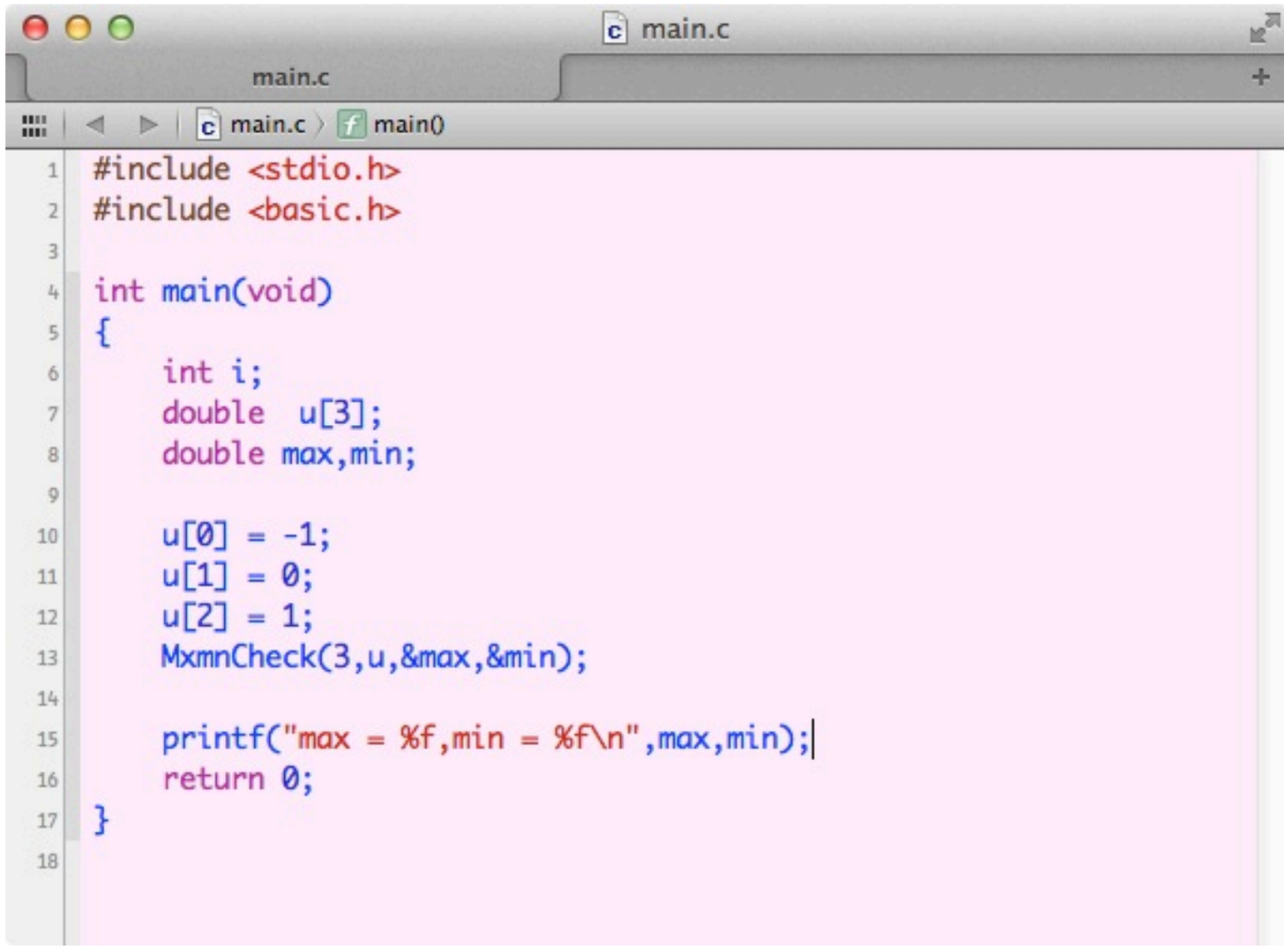
使用例　Test6/

http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html

A群

```
MxmnCheck(int n, double *u, double *max, double *min)
```

サイズnの配列uの最大値と最小値を返す. また両値が非常に近い時, 警告文を出す.



使用例　Test7/

A群

```
QuickSort(double *u, int start, int end)
```

配列uのstart番目からend番目までを小さいもの順に並べ替える．使用例では配列の全要素を並べ替えている．

使用例　Test8/

A群

Sign(double x)

引数の符号を調べて，＋なら１

ーなら-1を返す.

main.c

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

int main(void)
{
    double fugo;

    fugo = Sign(-1.234);
    printf("%f\n",fugo);

    return 0;
}
```

使用例　Test9/

A群

timer.h

timer.hはプログラムの計算時間を調べるためのツールをまとめたものである.

~/include/にtimer.hをコピーして使用するか, 使用したいプログラムのフォルダに入れincludeしてつかう.

右のように使用すると

StartTimer関数とStopTimer関数で囲まれた箇所の実行時間をt1はもつ.

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

#include "timer.h"

int main(void)
{
    int i,j = 1;
    unsigned int t0,t1;

    StartTimer(&t0);
    for(i = 1;i < 1000000000;i++)
    {
        j *= i;
        j = j % 100 + 1;
    }
    printf("j = %d\n",j);
    t1 = StopTimer(&t0);
    print_timer(t1);

    return 0;
}
```

使用例　Test10/

A群

```
ExtendNeumann1Dim(int imax, double *u1, double *u1_ext)
```

サイズnの配列uに対して，ノイマン型拡張*された配列u_extを計算する．拡張された配列のサイズ数はn+2であることに注意．ノイマン型拡張以外にもディリクレ型拡張，周期境界型拡張もある．また配列の次元も1~3まで使うことができる．ただし，2,3次元配列の格納方式は後述の行主導の格納方式であることに注意する．

ノイマン型拡張

u_ext[0] = u[0], u_ext[n + 1] = u[n - 1]

ディリクレ型拡張

u_ext[0] = 0, u_ext[n + 1] = 0

周期境界型拡張

u_ext[0] = u[n-1], u_ext[n + 1] = u[0]

使用例　Test11/

main.c

main.c › main()

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

#define     N   (10)

int main(void)
{
    int i;
    double  u[N],u_ext[N + 2];

    for(i = 0;i < N;i++)
    {
        u[i] = i;
    }
    ExtendNeumann1Dim(N,u,u_ext);
    for(i = 0;i < N + 2;i++)
    {
        printf("u[%2d] = %f\n",i,u_ext[i]);
    }
    return 0;
}
```

A群

GetParam(int argc, char **argv, int max_va_num,...)

数値計算プログラムを作成する際，パラメタ付きの問題を解く場合がある．そのような時，コマンドライン上でパラメータを指定出来れば便利である．本関数はそれを簡単に実装できる．

右のようにプログラム内で呼ぶことで実行ファイル名に続いて-a * -b * という風にパラメタを手打ちすることができる．パラメタの最大数は27であることに注意してほしい．

```
Test12 — Terminal — bash — 79×26
Test12;make
c++ -c -I/Users/masakazu/include -W -Wall -O   main.c
Linking test_program ...
done
Test12;./test_program
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
Test12;./test_program -a 5.0 -c 1.2
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 5.000000
param2 = 3.500000
param3 = 1.200000
Test12;./test_program -b 2.3
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 1.200000
param2 = 2.300000
param3 = -3.100000
Test12;
```

main.c

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

int main(int argc, char *argv[])
{
    double param1 = 1.2;
    double param2 = 3.5;
    double param3 = -3.1;

    printf("param1 = %f\n",param1);
    printf("param2 = %f\n",param2);
    printf("param3 = %f\n",param3);

    GetParam(argc,argv,27,
             &param1,
             &param2,
             &param3,
             NULL);

    printf("param1 = %f\n",param1);
    printf("param2 = %f\n",param2);
    printf("param3 = %f\n",param3);

    return 0;
}
```

使用例　Test12/

A群

GetParam2(int argc, char **argv, int max_va_num,...)

GetParamの最新版である．パラメタの数の上限（２７）が撤廃された.

右のようにプログラム内で呼ぶことで実行ファイル名に続いて-p0 * -p2 * という風にパラメタを手打ちすることができる．パラメタの最大数は指定した数となる.

```
Terminal — zsh — 41×26
done
Test16;./test_program
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
Test16;./test_program -p0 5.0 -p2 1.2
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 5.000000
param2 = 3.500000
param3 = 1.200000
Test16;./test_program -p1 2.3
param1 = 1.200000
param2 = 3.500000
param3 = -3.100000
param1 = 1.200000
param2 = 2.300000
param3 = -3.100000
Test16;
Test16;
Test16;
Test16;
```

main.c

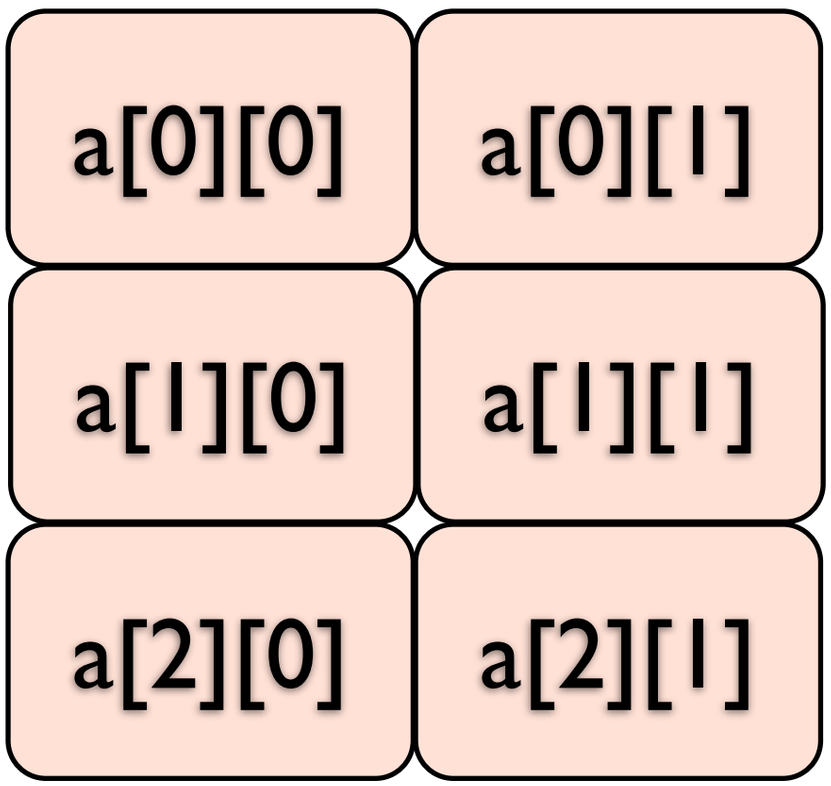
main.c 〉 No Selection

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

int main(int argc, char *argv[])
{
    double param1 = 1.2;
    double param2 = 3.5;
    double param3 = -3.1;

    printf("param1 = %f\n",param1);
    printf("param2 = %f\n",param2);
    printf("param3 = %f\n",param3);

    GetParam2(argc,argv,3,
            &param1,
            &param2,
            &param3,
            NULL);

    printf("param1 = %f\n",param1);
    printf("param2 = %f\n",param2);
    printf("param3 = %f\n",param3);

    return 0;
}
```

使用例　Test16/

ここでは配列の格納方法について述べる.

double a[3][2];



Cでは左のように宣言すると図のような６つの２次元配列を使うことができる. この配列は

a[行][列]

となっているので, 直感的でわかりやすい.

しかしながら, 関数を設計する際は問題が生じる. 例えばこの配列の和を計算するプログラムを作成した場合大抵以下のようなプログラムになる.

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

void Wa2(int Nx,int Ny,double (*u)[2],double *wa)
{
    *wa = 0;
    int i,j;

    for(i = 0;i < Nx;i++)
    {
        for(j = 0;j < Ny;j++)
        {
            *wa += u[i][j];
        }
    }
}

int main(void)
```

この中のdouble(*u)[2]という表現が問題である. すなわちCは配列の列方向の大きさを前もって指定してやらないと正しく計算してくれないのである. つづく

使用例　Test13/

ここでは配列の格納方法について述べる.

一般に配列の列サイズは可変である．またここではNyなのだからdouble (*u)[Ny]とすべきである．しかしながら，この表現はCではエラーとなるのである.
(Cの設計者がダメだと作者は思う)一般に多次元配列uが存在した場合，Cではその関数化は困難と見難い形となることが知られている.
double u[A][B][C][D]..[Z];と宣言した時，その関数の設計は
double (*u)[B][C][D]..[Z]としなければならない.
作者はこの自体を決してスマートであるとは考えない．もっと言えば，こんな表現をするプログラムを書きたくない．そこで，これらを一挙に解決する技が存在する.
それが

「1次元配列の多次元化」である.

１次元配列の多次元化技法について

例えば２次元配列 u[Imax][Jmax]を考えよう．これは同じ大きさの配列 v[Imax*Jmax]とうまくマッピングさせれば両者は同じ事である．すなわち v[i * Jmax + j] = u[i][j]とすればよいのである．しかしながら，プログラム中でいちいち v[i * Jmax + j]とかくのは格好が悪い．そこで右のようなマクロ置き換えを使うことでプログラムをスマートに書くことができる.

このプログラムのようにu(i,j)なる表現はマクロ置き換えによって直ちに内部的にu[j * 3 + i]となる．これが味噌である.

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

#define     Imax            (3)
#define     Jmax            (2)
#define     u(i,j)          u[(j) * (Imax) + (i)]

int main(void)
{
    int i,j;
    double u[Imax * Jmax];
    double  wa;

    for(i = 0;i < Imax;i++)
        for(j = 0;j < Jmax;j++)
            u(i,j) = i * Jmax + j;

    for(i = 0;i < Imax;i++)
    {
        printf("\n");
        for(j = 0;j < Jmax;j++)
        {
            printf("%f ",u(i,j));
        }
    }
    printf("\n");
    Sum(Imax * Jmax,u,&wa);
    printf("wa = %f\n",wa);
    return 0;
}

```

使用例　Test14/

1 次元配列の多次元化技法について

u(i,j)のマクロ置き換えは２つの自由度が生まれる.

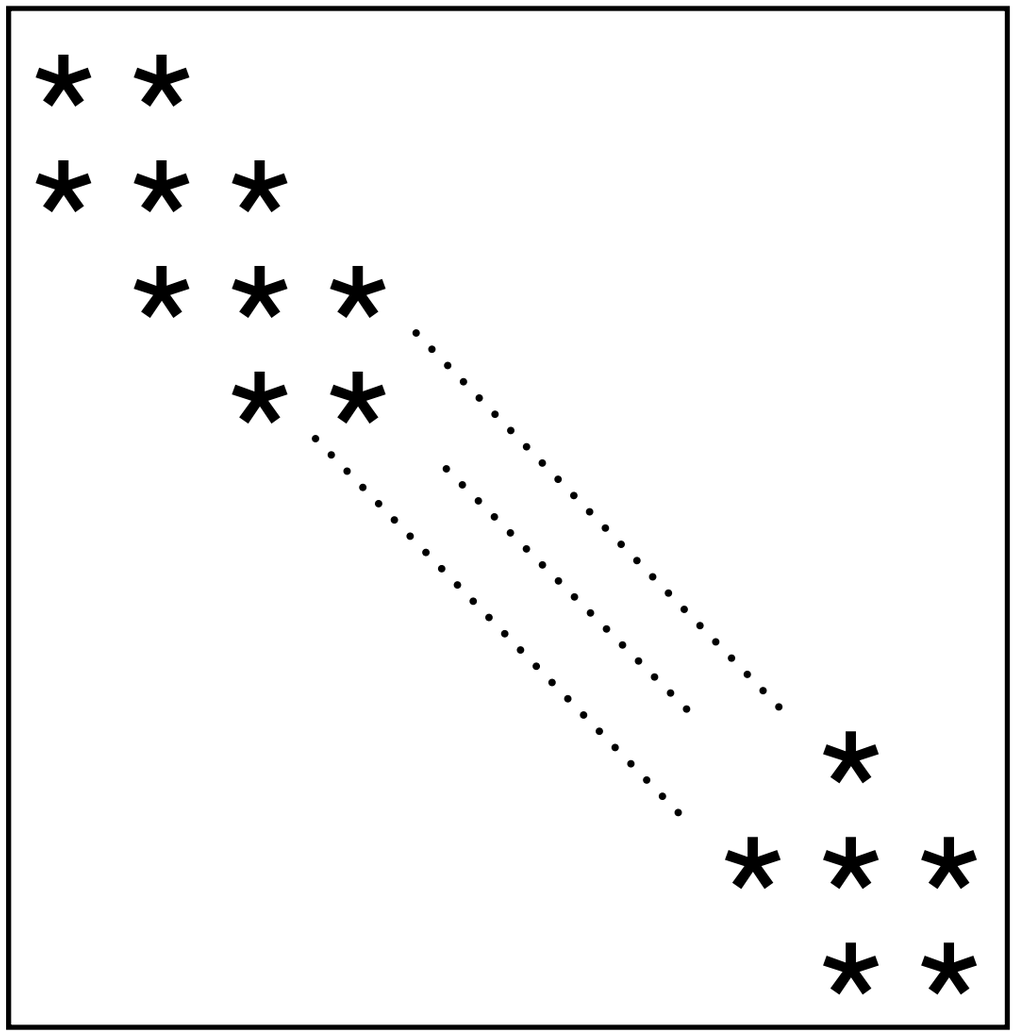
(A-Type)　　#define u(i,j) u[(j) * Imax + (i)]
(B-Type)　　#define u(i,j) u[(i) * Jmax + (j)]

ここでAは行主導型の行配列方式と呼ばれる．またBは列主導の配列方式とよばれる．これらはどちらも同じようにみえるが，混同してつかうとImax≠Jmaxのとき問題が起こることがわかるだろう.
そこで，mybasicでは
「すべての多次元配列っぽくつかえるマクロ置き換え配列はA-Typeの方式でデータを格納している」
ものとして，プログラムを作成していることに注意せねばならない.
このルールによって何次元配列であろうとも結局は 1 次元配列を参照することになるため，次元に応じた関数を作成するような馬鹿なことをしなくても良くなるのである.

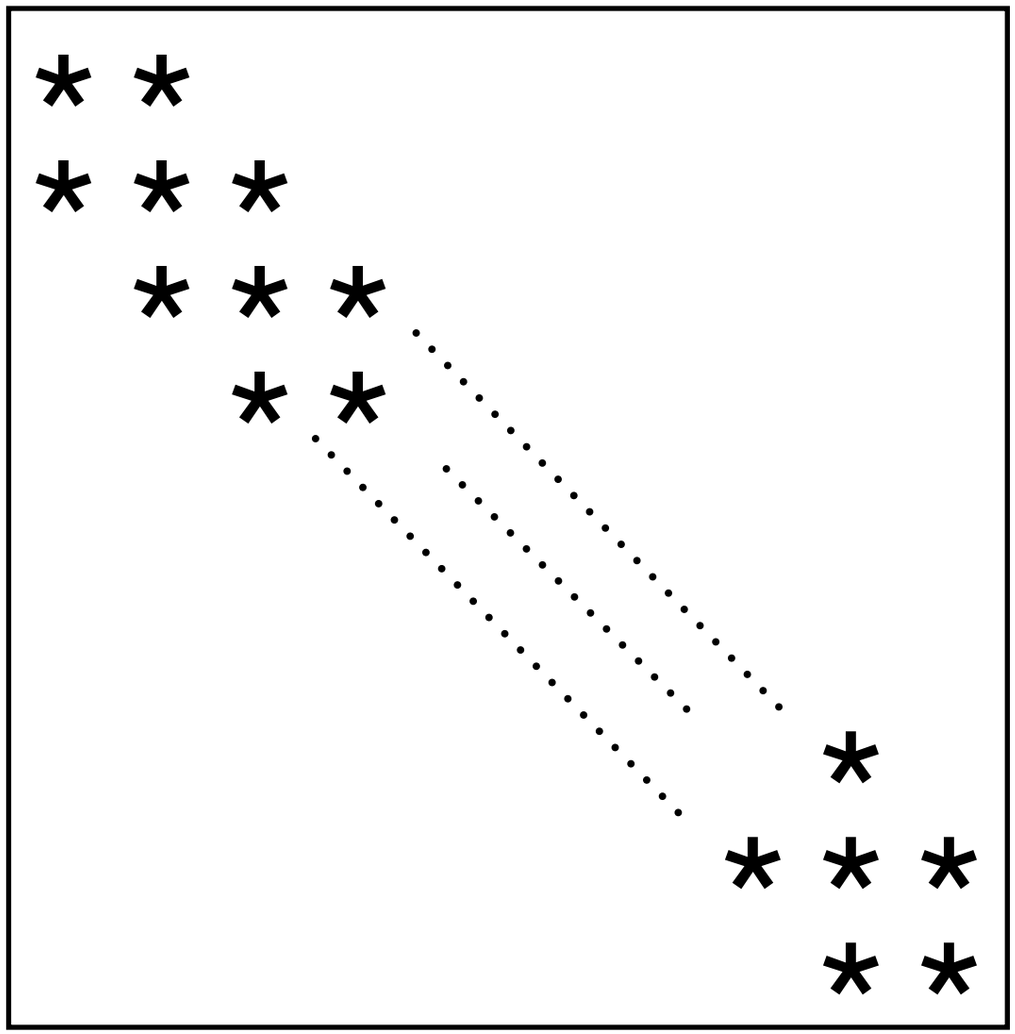
B群　LU分解について

mybasic内のLU分解ソルバは１次元の拡散方程式を解くために設計されている*.

差分方程式は境界条件によって以下の2つのタイプの行列が構成される.



ノイマン型境界条件 or ディリクレ型境界条件



周期境界条件

*ただし行列の構造が同じなら，拡散方程式にかかわらず解くことができる.

B群　LU分解について

mybasicではLU分解ソルバとして下の３つの関数を設計した.

（１）解くべき行列の生成（GenMatrix関数）

（２）行列のLU分解（LuDcmp関数）

（３）Ax=bを解く（LuSolve関数）

（１）解くべき行列の生成（GenMatrix関数）

行列は境界条件(mode=0,1,2はそれぞれNeumann,Dirichlet,Periodic)とc,Dをインプットしなければならない.　ここでｃ,Dは以下の意味である.