物理学情報処理演習
7. C言語③

2015年5月29日
ver2015529

本日の推奨作業directory

lesson07

7.1 プログラムの書き方
7.2 配列
7.3 関数・ライブラリ

参考文献

・やさしいC++ 第4版 高橋 麻奈 (著)
ソフトバンククリエイティブ
・プログラミング言語C++第4版
ビャーネ・ストラウストラップ, Bjarne Stroustrup, 柴田 望洋
・Numerical Recipes: The Art of Scientific Computing, Third Edition in C++

身内賢太朗

レポート提出：fsci-phys-jouhou@edu.kobe-u.ac.jp
課題提出期限 2015年6月12日13:00

7.1 プログラムの書き方

7.1.1 プログラムの書き方：変数の名前、型

数名、型をルールをもって付けよう！

変数名

- 全て英小文字(+ 数字 + _)で書く。
  - 一文字目には数字や_を避ける。
- 重要な変数は意味の分かる名前にする。

型

- Cの組み込み型には、
  - 整数: long, int, short, unsigned etc…
  - 実数: double, float
  - 文字: char
- 単精度実数”float”は使用しない。
  - 現在ほとんどマシンの内部計算は倍精度で行われている。
- 倍精度整数”long”は使用しない。
  - 現在ほとんどのマシンでintとlongは同精度

7.1.2 プログラムの書き方：定数

- よく使用する定数、あるいは重要な定数には名前をつけて使用する。
  - マクロ
    - #define MAX 1000000
    - #define BELL ‘¥x7’

    を使用する。
  - 定数名は
    - 全て英大文字(+数字+_)で書く。
    - 一文字目には、数字や__を避ける。
    - 重要な変数は、意味の分かる名前にする。
  - マクロ#defineは、プリプロセッサで処理され、単純な文字列の置き換えのみ行われる

7.1.2 プログラムの書き方：実行文

- 実行文を書く際には、制御の流れが見えるようにする。
  - 括弧
  - 改行
  - 字下げ

  を正しく使う、
    - emacsエディターのCモードでは、tabを使うことで字下げをそろえることができる。

  - 一行に複数の文を入れない。

    良い例:　i = 5*j;
    　　j = j*j;
    悪い例:　i = 5*j; j = j*j;

  - コメントは1行毎に行う方が良い。

    良い例:　// a comment
    　　// next comment
    悪い例:　/* a comment
    　　next comment */

7.1.2 プログラムの書き方：実行文

- 実行文を書く際には、制御の流れが見えるようにする。
  - 括弧
  - 改行
  - 字下げ

  を正しく使う、
    - emacsエディターのCモードでは、tabを使うことで字下げをそろえることができる。

  - 一行に複数の文を入れない。

    良い例:　i = 5*j;
    　　j = j*j;
    悪い例:　i = 5*j; j = j*j;

  - コメントは1行毎に行う方が良い。

    良い例:　// a comment
    　　// next comment
    悪い例:　/* a comment
    　　next comment */

- 字下げ・括弧を使ってループの深さを明らかにする
  - 推奨例

```
int func( double a )
{
    …
    return j;
}
```

7.1.2 プログラムの書き方：実行文

- まとまった処理は、関数として定義することでプログラムの流れを明確にする。
  - ループの深さは、せいぜい4～5段
  - 1つの関数の定義は、どんなに大きくても200～300行に押さえる。
  - コメントをいれることで、
    - 処理の内容
    - 事前条件
    - 事後条件

    等を明らかにする。

7.2 配列

7.2.1 配列の定義

- （同じ型の）変数の集合は配列として定義できる。
  箱を並べておくイメージ。
- 配列の宣言
  - 配列名と型、及び配列の大きさを指定する。
    - 例

      int array[10];　　//10個の整数からなる配列

      char moji[128];　/* 128個の文字からなる配列
  - 配列の大きさは定数でなくてはならない。
    - 良い例

      #define SIZE 100

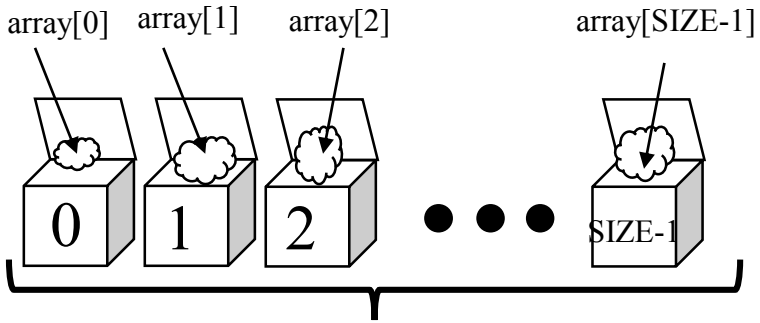
      …

      int array[SIZE];　// 100個の整数からなる配列
    - 悪い例

      int n = 100;

      …

      int array[n];　　// 100個の整数からなる配列



SIZE個の箱にそれぞれ変数を格納

factorial_6.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;

int main(int argc, char *argv[] )
{
 //calculated the factorial
 int n,i,j,ans;
 int factorial[11];
  if(argc>1){
   n=(int)atof(argv[1]);
  }
  else{
   n = 1;
  }
  ans=1;
  for(j=0;j<11;j++){
   ans=1;
   for(i=j;i>0;i--){
                ans=ans*i;
                //              cout << i << "¥t" << ans << endl;
   }
   factorial[j]=ans;
  }
  for(j=0;j<11;j++){
   cout << j << "¥t" << factorial[j] << endl;
  }
return 0;
}
```

factorial_6.cxx　:factorial_5.cxxを配列を用いて計算部と出力部を分離した例

7.2.2 配列への代入、参照

- 配列の各要素を参照（代入）するには、”[ ]”を使い、添字で指定する。
  - 添字は0から始まる。
  - よって、N個の要素の配列は、0, …, N−1 で指定される。

例

```
#define SIZE 10
int array[SIZE];
    // 10個の整数からなる配列
int index;   // 要素を指定する添字
for (index=0; index<SIZE; ++index) {
        array[index] = index * 2 - 1;
}
```

- 添字が配列の大きさを超えてもコンパイルエラーとはならない。
  - 例

```
int a[10];          //10個の整数からなる配列
a[100] = 100;       コンパイルエラーとはならない
```

実行時にセグメンテーション違反やバスエラーがでたり、おかしな振る舞いをしたりする。もちろん計算結果も正しくない場合がある。

factorial_6.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;

int main(int argc, char *argv[] )
{
 //calculated the factorial
 int n,i,j,ans;
 int factorial[11];
  if(argc>1){
   n=(int)atof(argv[1]);
  }
  else{
   n = 1;
  }
  ans=1;
  for(j=0;j<11;j++){
   ans=1;
   for(i=j;i>0;i--){
            ans=ans*i;
            //          cout << i << "¥t" << ans << endl;
   }
   factorial[j]=ans;
  }
  for(j=0;j<11;j++){
   cout << j << "¥t" << factorial[j] << endl;
  }
return 0;
}
```

factorial_6.cxx　:factorial_5.cxxを配列を用いて計算部と出力部を分離した例

7.2.3 配列のコピー

- 配列のコピーは、要素毎に参照し代入する。
  - 例

```
#define SIZE 100
int a[SIZE], b[SIZE];// 10個の整数からなる配列
int index;              //要素を指定する添字
for (index=0; index<SIZE; index++){
        a[index] = b[index];
}
```

- 配列全体の代入はできない。

悪い例

```
int a[100], b[100];          // 100個の整数からなる配列
…
a = b;                       // 配列全体の代入はできない
```

factorial_7.cxx

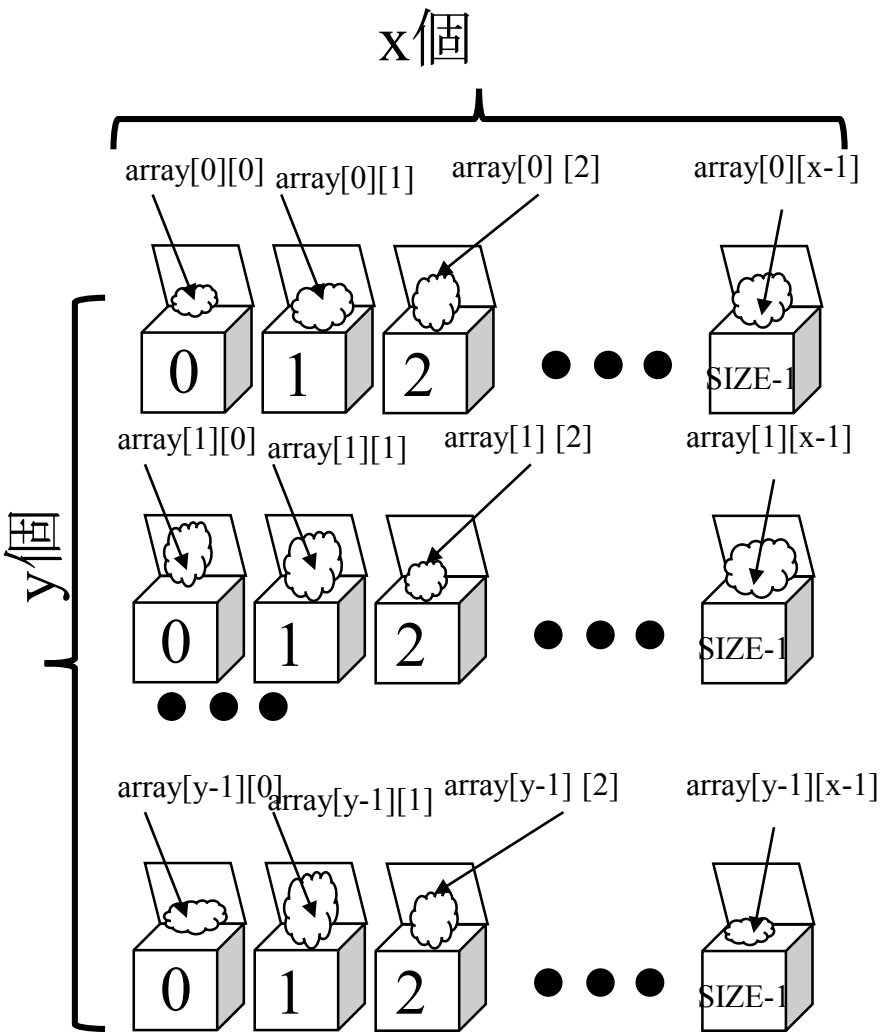
```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;

int main(int argc, char *argv[] )
{
  //calculated the factorial
  int n,i,j,ans;
  int factorial[11];
  int factorial_cp[11];
    if(argc>1){
      n=(int)atof(argv[1]);
    }
    else{
      n = 1;
    }
    ans=1;
    for(j=0;j<11;j++){
      ans=1;
      for(i=j;i>0;i--){
                ans=ans*i;
                //              cout << i << "¥t" << ans << endl;
      }
      factorial[j]=ans;
    }
    for(j=0;j<11;j++){
      factorial_cp[j]=factorial_[j];
    }
    for(j=0;j<11;j++){
      cout << j << "¥t" << factorial_cp[j] << endl;
    }

    return 0;
}
```

factorial_7.cxx　:factorial_6.cxxに配列のコピーを用いた例。

7.2.4 多次元配列

- []を複数書くことで多次元の配列を作ることができる。
  2次元配列は箱を2次元に並べておくイメージ。

- a[x][y] は x×y個の変数を保持できる



factorial_8.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;

int main(int argc, char *argv[] )
{
  //calculated the factorial
  int n,i,j,ans;
  int factorial[2][11];
    if(argc>1){
      n=(int)atof(argv[1]);
    }
    else{
      n = 1;
    }
    ans=1;
    for(j=0;j<11;j++){
      ans=1;
      for(i=j;i>0;i--){
                    ans=ans*i;
                    //                cout << i << "¥t" << ans << endl;
      }
      factorial[0][j]=j;
      factorial[1][j]=ans;
    }
    for(j=0;j<11;j++){
      cout << factorial[0][j] << "¥t" << factorial[1][j] << endl;
    }


      return 0;
}
```

factorial_8.cxx　:factorial_6.cxxを多次元配列を用いて書いた例。

（補） #defineの使用

- #define 指令を使って定数を名前で置き換える（マクロ定義）。
  - #define （文字列1）（文字列2） とすると、文字列1が文字列2に置換される。文字列1としては、関数呼び出しと区別するため通常大文字を使う。

    例

    #define NUMBER 5　　//NUMBER という名を5で置き換える

    #define GREETING “Hello”　　// GREETING を”Hello”で置き換える

- 定数の意味をはっきりさせるため、名前をつける。
  - 例

    #define SIZE 100　　// 100個の配列の大きさ

    …

    int array [SIZE];

    …

    for (i=0; i<SIZE; i++){

    …

    }

    マクロで定義しておけば配列の大きさを変えるときに一か所で済む。 → 間違いが少ない。

- #define 指令は、プリプロセッサによってマクロ名をその後の文字列で置き換えるよう処理される。
  - 文ではないので セミコロン “ ; ” は付けてはいけない。
  - マクロ名は変数ではないので代入できない。

    例

    #define NUMBER 100　プリプロセッサ後

    …　　…

    NUMBER = 10;　　100 = 10;

    コンパイルエラーとなる

factorial_9.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 11
int main(int argc, char *argv[] )
{
 //calculated the factorial
 int n,i,j,ans;
 int factorial[MAX];
  if(argc>1){
   n=(int)atof(argv[1]);
  }
  else{
   n = 1;
  }
  ans=1;
  for(j=0;j<MAX;j++){
   ans=1;
   for(i=j;i>0;i--){
                ans=ans*i;
                //              cout << i << "¥t" << ans << endl;
   }
   factorial[j]=ans;
  }
  for(j=0;j<MAX;j++){
   cout << j << "¥t" << factorial[j] << endl;
  }
  return 0;
}
```

factorial_9.cxx　:factorial_6.cxxの変数を定数にしたもの

演習7.1.3 次のようなプログラムを書いてみよう(提出は不要)

fact_6　を基にして、fact_7,8,9　　すべての修正を加え、0から13までの階乗を計算する fact_10を作ってみよう。

fact_10　の出力をgnuplotを用いて
　横軸n　(linear scale)
　縦軸n!　(log scale)　　というグラフを書いてみよう。

7.2.5 配列の初期化

- 配列を初期化するためには、次のようにする。
  - 例 int array[3] = { 0, 10, 20 };
- 配列の代入には"{ }"は使えない(各要素毎に代入する)。

  例

  int array[3];

  …

  array = { 0, 10, 20, 40 };　　コンパイルエラーとなる
- 初期化子が足りない配列要素は0に初期化
- 全く初期化していない配列の全ての要素は不定
- 大きさが与えられていない配列に初期化が行われると、その配列の大きさは初期化時の要素数と一致する。

  例

  int array[ ] = { 0, 10, 20, 40 };　　arrayの大きさ

grade_1.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define NUMBER 10
int main(int argc, char *argv[] ){
  int i;
  int mark[NUMBER]={5,5,5,5,5,5,5,5,4,6};
  for (i=0;i<NUMBER; i++){
    cout << mark[i]<<endl;
  }
  return 0;
}
```

grade_1.cxx　:配列初期化の例

(補) 配列への代入(cinの使用)

- 変数への代入でcinを用いることができる。
  - cin >> 変数
    標準入力から読み込んだ入力を変数に代入できる。

  - gradeA.txtを用いて
  - ./grade_2 < gradeA.txt
    とすることでgrade_1.txtと同じ結果を得ることができる。

grade_2.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define NUMBER 10
int main(int argc, char *argv[] ){
  int i;
  int mark[NUMBER];
  for (i=0;i<NUMBER; i++){
    cin >> mark[i];
  }
  for (i=0;i<NUMBER; i++){
    cout << mark[i]<<endl;
  }
  return 0;
}
```

grade_2.cxx　:cinを用いた変数への代入例

演習7.1.4 次のようなプログラムを書いてみよう(提出は不要)

grade_2　を基にして、平均点、標準偏差を求めるプログラム　grade_3を作る。

結果を第2講でエクセルで求めたものと比較する。

7.3 関数

7.3.1 関数

- 関数は
  - 名前
  - 返り値:型
  - 引数:型 ＋ 仮引数
    をもつ
  - void型 の使い方
    返り値がないときは、返り値の型をvoidにする
    引数がないときは、引数の型をvoidにする

- 関数を使用するには
  - 宣言: 関数名、返り値の型、引数の数と型
  - 定義(実装):処理の内容が必要

- 関数(定義部)の構造
  - 宣言文
    - 変数の宣言
  - 実行文
    - 実際の処理を行う

factorial_11.cxx

関数の宣言

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 11
int fact(int n){
  int i,ans;
   ans=1;
  for(i=1;i<n+1;i++){
   ans=ans*i;
  }
  return ans;
}

int main(int argc, char *argv[] )
{
  //calculated the factorial
  int n,i,j,ans;
  int factorial[MAX];
   if(argc>1){
    n=(int)atof(argv[1]);
   }
   else{
    n = 1;
   }
   for(j=0;j<MAX;j++){
    cout << j << "¥t" << fact(j) << endl;
   }
   return 0;
}
```

関数の定義

関数の呼び出し

factorial_11.cxx:factorial_6を関数を用いて書き換えたもの

7.3.2 関数への値渡し

- 関数へ”値渡し“を行える
  - 値は一致している必要がある。
  - 呼び出した側の引数に入っているjが、呼び出された側の引数nに渡される。

呼び出された側の変数 n の値を変更しても、呼び出した側の変数 j は変化しない

factorial_11.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 11
int fact(int n){
  int i,ans;
    ans=1;
  for(i=1;i<n+1;i++){
    ans=ans*i;
  }
  return ans;
}

int main(int argc, char *argv[] )
{
  //calculated the factorial
  int n,i,j,ans;
  int factorial[MAX];
    if(argc>1){
      n=(int)atof(argv[1]);
    }
    else{
      n = 1;
    }
    for(j=0;j<MAX;j++){
      cout << j << "¥t" << fact(j) << endl;
    }
    return 0;
}
```

関数の呼び出し

factorial_11.cxx:factorial_6を関数を用いて書き換えたもの

7.3.4　関数への値渡し(配列)

- 配列を引数としてとる関数
  - 配列の宣言は、
    - 呼び出す側では大きさを指定
    - 関数側では大きさ不定の配列
    - 型は一致していなければならない。
    - 例

```
int array[10];          // 10個の整数からなる配列

func(array);// 整数配列を引数にとる関数の 呼び出し
void func(int a[])   // 整数配列を引数に
                     // とる関数の定義
{
…
}
```

    - 関数内で、配列の要素が変更された場合、関数から戻ってもその変更は有効

7.3.5 ライブラリ

- ライブラリ：関数群

- ヘッダーファイル（xxxx.h）に、これらのライブラリ関数の宣言が書かれている
- 関数の中身は、ライブラリ（libxxx.a）にある
  - 入出力　　iostream
    例：cout,cin
  - 文字操作　　ctype.h, string.h
    例：　　int tolower(int *c*)
  - ユーティリティ　stdlib.h
    例：　　atoi(const char **s*)
  - 日付　　time.h
    例：　　time_t time(time_t **tp*)
  - 算術　　math.h
    例：　　double sqrt(double a)
    例：　　int rand()
    例：　　int RAND_MAX

factorial_11.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 11
int fact(int n){
  int i,ans;
    ans=1;
  for(i=1;i<n+1;i++){
    ans=ans*i;
  }
  return ans;
}

int main(int argc, char *argv[] )
{
  //calculated the factorial
  int n,i,j,ans;
  int factorial[MAX];
    if(argc>1){
      n=(int)atof(argv[1]);
    }
    else{
      n = 1;
    }
    for(j=0;j<MAX;j++){
      cout << j << "¥t" << fact(j) << endl;
    }
    return 0;
}
```

関数の呼び出し

factorial_11.cxx

レポート提出：fsci-phys-jouhou@edu.kobe-u.ac.jp
提出期限　2015年6月12日13:00

ソースコードファイル名：2015_jouhou_07_学籍番号の下4桁.cxx
出力ファイル名：2015_jouhou_07_学籍番号の下4桁.dat
出力画像ファイル名：2015_jouhou_07_学籍番号の下4桁.pdf

課題7：以下の仕様のプログラムを製作し、正規分布について計算せよ。ソースコード、出力ファイル、gnuplotで描画した出力結果の図を提出せよ。(必要であればgaus_sample.cxxを参考にすること。)

①　出力1列目：−10から10まで0.1刻みの実数

②　出力2列目：規格化された正規分布

$$N(x) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x^2}{2}\right)$$

③　出力3列目：出力2列目にステップ幅0.1をかけたものを−10から順に足し挙げていったもの(累積確率分布関数。−10で0　+10で1になるはず。)

グラフ①
横軸：リニアスケール　1列目（範囲は−10～10）
縦軸：リニアスケール　2列目（正規分布　範囲は0～1）
グラフ②（replotを用いてグラフ①と同じ絵に描くこと）
横軸：リニアスケール　1列目（範囲は−10～10）
縦軸：リニアスケール　3列目（累積分布関数　範囲は0～1）