物理学情報処理演習
9. C言語⑤

2015年6月19日

本日の推奨作業directory

lesson09

9.1 乱数
9.2 ポインタ

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

9.1 乱数

- 乱数はシミュレーションなどを行う際に有用である。
- `default`では再現する乱数が発生する。
  →挙動の確認のため。
- 「真の」乱数を得るためには適切な初期値(乱数の種)を与える必要がある。通常`time(NULL)`を用いて、現在時刻を種とする。

- 算術ライブラリ `math.h`

  `int rand()` 0からRAND_MAXまでの整数を発生する。

  `int RAND_MAX`

  `int srand();`

rand_1.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 10
int main(int argc, char *argv[] )
{
 int j;
  for(j=0;j<MAX;j++){
    cout << j << "¥t" << rand() << "¥t" << RAND_MAX<< endl;
 }
 return 0;
}
```

rand_3.cxx

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define MAX 10
int main(int argc, char *argv[] )
{
 int j;
 srand(time(NULL));
  for(j=0;j<MAX;j++){
    cout << j << "¥t" << rand() << "¥t" << RAND_MAX<< endl;
 }
 return 0;
}
```

演習9.1(提出不要) rand_1を何回か走らせてみて、乱数発生が再現していることを確認してみよう。rand2.cxxを変更してsrandで与える種を変更して実行してみよう。rand_1では乱数の種に何が与えられているだろうか。

rand_3を何回か実行してみよう。再現性はあるだろうか。rand_3を変更して、rand_4として1列目にj、2列目に0～1までの間の乱数を表示するプログラムとしておこう。(double)rand()としてint型をdouble型に変更する必要がある。

9.2 ポインタ

9.2.1 ポインタ

・ある変数に対するアドレス(メモリ上の位置)をもつ変数

・＊ を付けて定義する。

例 int *p_int; //*p_intというパラメータがint p_intがポインタ変数

・ポインタ演算子 ＊ は、ポインタの指し示している変数の値を得る演算子

・アドレス演算子&は、変数のアドレスを得る演算子

pointer_1.cxx

```
#include <iostream>
#include<string.h>
#include<stdio.h>
using namespace std;
int main(){
  int *p_int;
  int i=1;
  *p_int=2;
  cout << "i: "<<i<<endl; //integer
  cout << "*p_int: "<<*p_int<<endl; // integer
  cout << "p_int: "<<p_int<<endl; // pointer of integer
  cout << "&(*p_int): "<<&(*p_int)<<endl; // pointer of integer
  return 0;
}
}
```

演習9.1 (提出不要) pointer_1を実行、何が起きているか理解しよう。double型のポインタ変数を入れてみよう。

9.2.2 配列とポインタ

- 配列を示す変数には、配列の先頭アドレスが入っている。
  - 配列を示す変数は、ポインタとしても使用できる。
    - 例1: 次のようにすると、moji1, moji2 は同じ文字となる。

```
char array[128], moji1, moji2;
moji1 = array[0]; /* 0番目の文字 */
moji2 = *array;  /* 配列の先頭の文字 */
```

    - 例2: 次のようにすると、moji1, moji2 は同じ文字となる。

```
char array[128], moji1, moji2;
int i;
moji1 = array[i]; /* i番目の文字 */
moji2 = *(array+i); /* 配列の先頭からi番目の文字
*/
```

配列とポインタ

- 配列とポインタの違い
  - ポインタを宣言したときは、
    - アドレスの値を入れる「メモリ領域」が確保される。
  - 配列を示す変数は、ポインタとしても使用できる。
    - 配列全体を格納する「メモリ領域」が確保されると共に、
    - アドレスの値を入れる「メモリ領域」が確保され、
    - 配列の「先頭アドレス」がセットされる。

多次元配列・ポインタの配列

複数の文字列をつくるには、

- （文字の）多次元配列

```
char array[A_SIZE][STR_SIZE]
```

- （文字への）ポインタからなる配列

```
char *array[A_SIZE]
```

- （文字への）ポインタへのポインタ

```
char **array
```

が使用できる

9.2.3 ポインタと関数

- これまで:関数への値渡し
  - 関数へ値を渡す。渡した変数は変更されない。

演習9.2 (提出不要) hist_1 < grade.dat として実行、上記を確認しよう。また、結果をパイプリダイレクションで grade_hist.datに書き出してgnuplotで描画してみよう。

hist_1.cxx

```
##include <iostream>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
#define HIST_MIN 0
#define HIST_MAX 50
#define HIST_BIN 10
int output_hist(int value){
  cout <<value<<"¥t";
  value=0;
  cout <<value<<“¥t”;          // 関数内の変数valueは変更される。
}
int main(int argc, char *argv[] )
{
  int i,num;
  double hist[3][HIST_BIN];
  double hist_step=(double)(HIST_MAX-HIST_MIN)/HIST_BIN;
  double grade;
  for(i=0;i<HIST_BIN;i++){
    hist[0][i]=hist_step*i;//hist[0] for lower bound
    hist[1][i]=hist_step*(i+1);// hist[1] for upper bound
    hist[2][i]=0; //hist[2] for contents
  }
  num=0;
  while(cin>>grade){
    hist[2][(int)(grade/hist_step)]++; // filling the histgram
    //    cout << grade<<“¥t”<<hist[0][(int)(grade/hist_step)]<<“¥t”<<
hist[1][(int)(grade/hist_step)]<<"¥t"<<hist[2][(int)(grade/hist_step)]<<endl;
    // check for histgram filling
    num++;
  }
  for(i=0;i<HIST_BIN;i++){
    cout<<hist[0][i]<<"¥t"<<hist[1][i]<<"¥t"<<hist[2][i]<<"¥t";
    output_hist(hist[2][i]);
    cout<<hist[2][i]<<endl;        // mainの変数hist[2][i]は変更されない。

  }
  return 0;
```

hist_2.cxx

```
using namespace std;
#define HIST_MIN 0
#define HIST_MAX 50
#define HIST_BIN 10
int hist_init(double hist[3][HIST_BIN],double *hist_step){
  int i;
  *hist_step=(double)(HIST_MAX-HIST_MIN)/HIST_BIN;
  for(i=0;i<HIST_BIN;i++){
    hist[0][i]=(*hist_step)*i;
    hist[1][i]=(*hist_step)*(i+1);
    hist[2][i]=0;
  }
}
int hist_fill(double value,double hist[3][HIST_BIN],double
hist_step){
  hist_step=(double)(HIST_MAX-HIST_MIN)/HIST_BIN;
  hist[2][(int)(value/hist_step)]++;
  return 0;
}
int hist_output(double hist[3][HIST_BIN]){
  int i;
  for(i=0;i<HIST_BIN;i++){
    cout<<hist[0][i]<<"¥t"<<hist[1][i]<<"¥t"<<hist[2][i]<<endl;
  }
  return 0;
}
int main(int argc, char *argv[] )
{
  int i,num;
  double hist[3][HIST_BIN];
  double hist_step;
  double grade;
  hist_init(hist,&hist_step);
  num=0;
  while(cin>>grade){
    hist_fill(grade,hist,hist_step);
    num++;
  }
  hist_output(hist);

  return 0;
```

- 新:アドレス渡し
  - 呼び出した側の引数としてアドレスを渡す。
  - 関数の処理によって変数の値を変えることができる。

呼び出し側の引数のアドレス(`&hist_step`)が、呼び出された側の引数であるポインタ変数(`hist_step`)の値(`*hist_step`)となる。

呼び出された側で、`hist_step`の指し示す変数の値を変更しているので、呼び出し側の変数`hist_step`も変化する。

演習9.2 (提出不要) hist_2 < grade.datとして実行、上記を確認しよう。また、結果をパイプリダイレクションでgrade_hist.datに書き出してgnuplotで描画してみよう。(必要に応じてshow_hist.pltを使うこと。)

レポート提出：fsci-phys-jouhou@edu.kobe-u.ac.jp
提出期限　2015年6月26日13:00

ソースコードファイル名：2015_jouhou_09_学籍番号の下4桁.cxx
出力画像ファイル名：2015_jouhou_09_学籍番号の下4桁.pdf

課題9：以下の様なコードを作成せよ。
提出はソースコード、gnuplotの出力画像ファイル、の2点とする。
(プログラム仕様)
① 0から1までの乱数を100回発生させ、そのうちで0.01以下になる回数Xを数える。
② ①を100回繰り返し、横軸X、縦軸頻度確率のグラフを出力する。
③ ②の出力と期待値1のポアソン分布をgnuplotで重ね書きする。