

物理学情報処理演習

8. C言語⑤ 文字列・ポインタ

2016年6月7日

ver20160607_2

本日の推奨作業directory
lesson08

8.1 文字列
8.2 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++

身内賢太郎

レポート提出: fsci-phys-jouhou@edu.kobe-u.ac.jp

課題提出期限 2016年6月14日13:00

8.1 文字列

• 8.1.1 文字列処理

- ある変数に対するアドレス(メモリ上の位置)をもつ変数
- **N個の文字**からなる文字列は、**N+1個の大きさ**の文字配列
- 末尾の文字は、**NULL文字** ('**¥0**')
 - 例 `char c[16];`

```
#include <iostream>
using namespace std;
int main(){
// comment can be written starting with "///"
char c[16]="Hello World!";
cout <<c << endl;
return 0;
}
```

hello_2.cxx

配列cの中身

‘H’ ‘e’ ‘l’ ‘l’ ‘o’ ‘ ’ ‘W’ ‘o’ ‘r’ ‘l’ ‘d’ ‘!’ ‘¥0’

演習8.1.1(提出不要) hello_2e.cxx、hello_2ee.cxx、hello_2eee.cxx をdebugしてみよう。エラーメッセージをよく確認すること。

• 8.1.2 文字列処理

- 標準ライブラリの<string.h>に定義してある文字列処理ルーチン

<code>char* strcpy(char* s, const char* ct)</code>	NULL文字を含めて文字列ctを文字列sにコピーし、sを返す。
<code>char* strncpy(char* s, const char* ct, int n)</code>	文字列ct内からn文字を文字列sにコピーし、sを返す。ctがn文字より少ないときはNULL文字をつめる。
<code>char* strcat(char* s, const char* ct)</code>	文字列ctを文字列sの終わりに連結し、sを返す。
<code>char* strncat(char* s, const char* ct, int n)</code>	文字列ct内から最大n文字を文字列sの終わりに連結し、sを返す。
<code>int strcmp(const char* cs, const char* ct)</code>	文字列csと文字列ctを比較する。一致していれば0を返す。一致していなければ、相違が発見された文字どうしの数値上の差を返す。
<code>size_t strlen(const char* cs)</code>	文字列csの長さを返す。

演習8.1.2 (提出不要) hello_3.cxxを実行してみよう。内容を確認して、上記機能を試してみよう。

8.1.2 文字列処理

- 標準ライブラリの<stdio.h>の関数printfも使える。

<pre>int printf(const char *format, . . .);</pre>	第一引数に書式を書き、第二引数以降に実際に出力される変数を書く。 出力は標準出力。
<pre>int sprintf(char *str, const char *format, . . .);</pre>	第一引数に代入されるべき文字、第二引数に書式を書き、第三引数以降に実際に出力される変数を書く。

```
#include <iostream>
#include <string.h>
#include <stdio.h>
using namespace std;
int main(){
// comment can be written starting with "/"
int i;
char c1[16]="Hello ";
char c2[16]="World!";
char c3[16];
char c4[64];
cout <<c1 << endl;
strcat(c1,c2);
cout <<c1 << endl;
for(i=0;i<10;i++){
    sprintf(c3,"%d",i);
    //substitute i to c3 as a character
    sprintf(c4,"%s%s_%d.dat",c1,c2,i);
    //sprintf can have more than one paramters
    printf("%d\t%s",i,c3);
    //same as cout << i << "\t" << c3 << endl;
}
return 0;
}
```

hello_4.cxx

<書式の例>

☆☆☆ %d 10進数
☆☆☆ %lf double
☆☆☆ %e e 指数表示
☆☆☆ %s 文字列
☆☆☆ フィールド幅 %とdなどの間に数字を入れる。
%3dで3桁で整数をs桁で表示
%2.1fで全体で2桁、小数点以下1桁
%2.1eで全体で2桁、小数点以下1桁
☆☆ %c 1文字

<エスケープシーケンスの例>

☆☆☆ \n 改行
☆☆☆ \t タブ
☆☆ \r キャリッジリターン

演習8.1.3 (提出不要) hello_4.cxx, hello_5.cxxを実行してみよう。内容を確認して、上記機能を試してみよう。

• 8.1.3 多次元配列

任意の型の配列を多次元化することが可能。

```
#include <iostream>
#include <stdlib.h>
#include <string.h>
#include <math.h>
using namespace std;
int fact(int n){
    //calculated the factorial
    int i,ans;
    ans=1;
    for(i=1;i<=n;i++){
        ans=ans*i;
    }
    return ans;
}
int main(int argc, char *argv[] )
{
    int n,max,i;
    int ans[2][20];
    if(argc>1){
        max=(int)atof(argv[1]);
    }
    else{
        // for default
        max = 10;
    }
    for(i=0;i<max;i++){
        ans[0][i]=i;
        ans[1][i]=fact(i);
    }
    for(i=0;i<max;i++){
        cout <<ans[0][i]<<"!="<<ans[1][i] << endl;
    }
    return 0;
}
```

演習8.1.4 (提出不要) 階乗計算のプログラムを多次元配列を用いて書いたサンプルコード、fact_13.cxxを実行してみよう。出力3列目に1列目の数の2乗を出力するように変更を加えてみよう。

複数の文字列も多次元配列

`char array[A_SIZE][STR_SIZE]` が使える。

例

```
char array[4][8] = { "First", "Second", "Third", "Last" };  
                    value
```

`array[0]`



`array[0][0]`

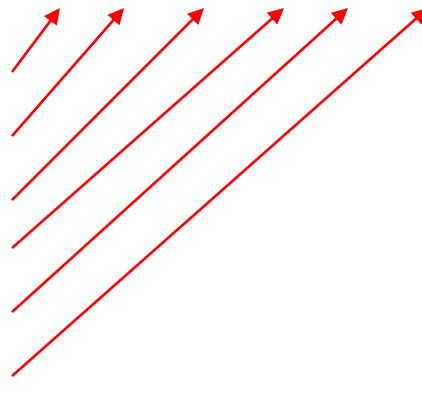
`array[0][1]`

`array[0][2]`

`array[0][3]`

`array[0][4]`

`array[0][5]`



`array[1]`



`array[2]`



`array[3]`



演習8.1.5 (提出不要) `hello_3.cxx`を二次元配列を用いて書いてみよう。

• 出力 ofstream

- #include <fstream>が必要
- これまでcout,cerrで出力していたが、出力先をファイルとすることも可能。

```
#include <iostream>
#include <fstream>
#include <string.h>
#include <stdio.h>
using namespace std;
int main(){
    int i;
    char fout[16]="test.out";
    char c1[16]="Hello World!";
    ofstream ofs(fout);
    ofs << c1<<endl;
    return 0;
}
```

ofstest_1.cxx

演習8.1.6 (提出不要) ofstest_1.cxxを書き換えて、出力ファイル名を変更してみよう。

8.2 ポインタ

8.2.1 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

```
#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;
}
```

pointer_1.cxx

演習8.2.1 (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。

8.2.2 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

8.2.3 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8.2.2 (提出不要) hist_1 < grade.dat
として実行、上記を確認しよう。また、結果をパイプラインで
grade_hist.datに書き出してgnuplotで描
画してみよう。

```
##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";
}
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 histogram
        // 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 histogram 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;
    }
}
return 0;
```

hist_1.cxx

// 関数内の変数valueは変更される。

// mainの変数hist[2][i]は変更されない。

```

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8.2.3 (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。)

課題8: 課題7のソースなどを基にして、以下の様なコードを作成せよ。
提出はソースコード、gnuplotの出力ファイル、作業directoryでlsした出力をtxtとしたファイルの3点とする。

(プログラム仕様)

① 平均値 $\mu=0$ 標準偏差 σ の規格化された正規分布を計算、出力1列目に X (-10から10)、2列目に確率をファイルに出力する。

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

② プログラム内で標準偏差 σ を0から5まで0.1刻みで変更し、それぞれの σ に対応する出力ファイルをgaus_?.?.dat(例: $\sigma=2.5$ の出力はgaus_2.5.dat)として、プログラムから自動的に生成されるようにする。(hello_5.cxx, oufstest_1.cxx参照。全部で50ファイルできるはずである。lsの出力をテキストファイルに落として提出のこと。)

③ ②で作成したデータのうちで $\sigma=1.0, 3.0, 5.0$ について同じplotに描画せよ。(必要であれば、show.pltを使用せよ。)