

Cプログラミング

基幹7クラス
第1回

講義概要・C言語の復習(1)

青柳滋己

aoyagi@aoni.waseda.jp

講義目的

実践的なプログラミング演習を多く取り入れ、プログラミング技術習得を目指す上でのアドバンス科目と位置づける。本科目は、以下2項目の習得を目標として授業を進める。

1. 実践的プログラミング技術の習得

「Cプログラミング入門」で学んだ内容を踏まえて、科学技術計算処理、データの加工等、実践的事例を取り入れ、プログラミング技術の向上を目標とする。

2. プログラミングにおける総合的知識・技術の習得

本科目では、作品を完成させる演習を行う。演習を通して、設計・開発・評価というソフトウェア開発における一連の工程を実践することで、プログラミングにおける総合的知識・技術を習得する。

講義内容(予定)

- 第1回 講義概要・C言語の復習(1)
- 第2回 C言語の復習(2)
- 第3回 連立一次方程式の解法(1)
- 第4回 連立一次方程式の解法(2)
- 第5回 非線形方程式の解法(1)
- 第6回 非線形方程式の解法(2)
- 第7回 常微分方程式の解法
- 第8回 再帰
- 第9回 プログラミング演習
- 第10回 ソート
- 第11回 探索
- 第12回 データ構造(1)
- 第13回 データ構造(2)
- 第14回 プログラミング演習
- 第15回 レポート解説(Course N@viで行う)

授業の前提

- 以下は習得済みを前提とする.
 - ◆ Cプログラミング入門の知識
 - ◆ PCの基本操作, C言語の基本
 - ◆ ファイル作成やコンパイル
 - Linuxの基礎, gccの使い方

授業形態

■ 授業

授業前半は講義
授業後半は演習

■ 教科書・参考書

教科書は指定しない. 参考書として以下の本を推薦する.

- ・林 晴比古:「新C言語入門 シニア編」, ソフトバンククリエイティブ
- ・B.W.カーニハン, D.M.リッチー:「プログラミング言語C 第2版」, 共立出版

成績評価

■ 出欠

- ◆ 遅刻は授業開始後10分まで認める. それ以上遅れた場合は欠席とする.
- ◆ 全14回の講義のうち, **2/3以上の出席**が必須.
- ◆ 電車などが遅れた場合は遅延証明書等を提示すること.
- ◆ 出席登録は各自がCourseN@viで行う
- ◆ 登録方法は

<http://www-it.sci.waseda.ac.jp/links/attendance/> 参照

■ レポート

- ◆ レポートで評価. 試験はしない.
- ◆ レポートは2回実施予定. (第9回, 第14回を予定)
- ◆ 授業に欠席してもレポートは受け付ける.
- ◆ レポート提出にはCourseN@viを使う.

サポート

■ TAによるサポート

授業中はTA(Teaching Assistant)の方がサポートしてくれます。

河辺 夏希さん

若山 馨太さん

桑島 昂平さん

■ 講義資料等

適宜更新するので、参照すること。

<http://www-it.sci.waseda.ac.jp/>

■ 情報Q&A

<http://www-it.sci.waseda.ac.jp/itqa/>

■ 情報関連科目FAQも目を通しておくこと

<http://www-it.sci.waseda.ac.jp/links/faq.html>

授業への質問

■ プログラミングや授業内容に関する質問等

◆ 授業時間中にTAまたは講師へ

■ 出欠席に関する質問等

◆ 理工メディアセンタ窓口へ

site@list.waseda.jp

◆ 授業開始前・終了後にTAまたは講師へ

■ コンピュータルーム環境に関する質問等

helpdesk@mse.waseda.ac.jp

■ その他, 個人的事情による問い合わせや授業への要望など

aoyagi@aoni.waseda.jp 宛にmailで

その他

- 作成するCプログラムは理工UNIXシステム (Linux)の gcc でコンパイルし動作させること
 - ◆ Linuxシステムの基本コマンド等の習熟は必須
 - ◆ コンパイルエラーや警告が出ないようにすること

本日の講義概要

- UNIXシステムへのログイン・ログアウト

- C言語の復習

 - ◆ 変数

 - ◆ 制御構造



 - ◆ 配列

 - ◆ 乱数

UNIXへのログイン・ログアウト

PCを起動し、WindowsとLinuxの選択画面が出たらLinuxを選択する.

■ ログイン

1. ユーザIDを入力して  (Enter)キーを押す
2. パスワードを入力して  キーを押す

※ ユーザIDは学籍番号(CD付きハイフンなし9桁)
パスワードはWaseda-netパスワード

■ ログアウト

1. 上部パネルのアクションをクリック
2. メニューからログアウトかシャットダウンを選択
3. OKをクリック

※ Windowsを起動後、TeraTermを使ってログインしても良い

Cプログラムの基本形

プログラムを実行すると、main関数が呼ばれ、main関数内に書かれた処理を実行し、終わるとプログラムも終了する。

```
#include <stdio.h> /* 入出力がなければ不要 */

int main(void)
{
    /* 最初に変数の宣言。変数の解説は次回。 */

    /* 処理内容をここに書く */
    .....

    return 0; /* プログラム終了 */
}
```

- Cのソースファイル名は、○○○.cと拡張子に.cをつけること。
- 0は数字。英字の0との区別に注意。

プログラムの例(hello.c)

hello.c

```
#include <stdio.h>

int main(void)
{
    printf("Hello, world.¥n");
    return 0;
}
```

- 見やすいように字下げ(indent)を行うこと.
- printf("...");は“...”の部分画面に出力する関数. “¥n”は“\”
のことで、改行を意味する.

gccによるコンパイル方法

- コンパイルには gcc を使う.

```
$ gcc hello.c
```

- コンパイルに成功すると a.out というファイルがカレントディレクトリに作成される.

```
$ ls  
a.out  hello.c  
$
```

- 実行するには, ./a.out  とする.

```
$ ./a.out  
Hello, world  
$
```

gccのオプション

- gccに "-o ファイル名" というオプションをつけると, a.outではなくファイル名の実行ファイルが作られる.

```
$ gcc -o hello hello.c
$ ls
hello hello.c
$ ./hello
Hello, world.
$
```

変数

- プログラム中、数や文字などのデータを一定期間記憶する場合には、記憶する場所に名前をつけて区別する。それを**変数**という。
cf. 電卓のメモリ
- 名前の異なる変数は違うものとして区別される。この名前のことを識別子という。
- どのような名前の変数を使うのかプログラム中で明示しなければならない。それを**変数の宣言**という。
- 変数の宣言は次の形式。
変数の型 変数名(=初期値);
- 変数の宣言はプログラム(関数ブロック)の最初で行う。

```
#include <stdio.h>

int main(void)
{
    int num;
    int max=1000;

    .....
}
```


変数への代入, 参照

- 変数へ実際のデータを関連づけることを代入という.
- 代入には = を使う.
変数名 = 式;
右辺の式を計算した値が変数に代入される.
- 変数への最初の代入を初期化という.
- 変数に代入したデータを使うことを参照という.
- 変数を参照する前に, その変数は初期化されていないなければならない.

```
#include <stdio.h>

int main(void)
{
    int num;
    int max=1000;
    .....

    num=4;
    height1=100; width1=30;
    height2=height1;
    width2=width1;

    .....
}
```

if文

■ if文の基本形

```
if (条件式) { /* 条件式が真の時に実行 */  
    文1;  
    文2;  
    ..  
    文m;  
} else { /* 条件式が偽の時に実行 */  
    文1;  
    ..  
    文n;  
}
```

■ else以降はなくても良い

```
if (条件式) { /* 条件式が真の時に実行 */  
    文1; 文2; ...  
}
```

if文(続き)

■ else ifを使う

```
if (条件式1) {          /* 条件式1が真の時に実行 */
    .....
} else if (条件式2) { /* 条件式1が偽で条件式2が真の時に実行 */
    .....
} else if (条件式3) { /* 条件式1,2が偽で条件式3が真の時に実行 */
    .....
} else {                /* 条件式1,2,3が偽の時に実行 */
    .....
}
```

■ 文が1つの場合, {}を省略できる.

```
if (条件式) 文1; /* 条件式が真の時, 文1 を実行 */
else 文2;      /* 条件式が偽の時, 文2 を実行 */
```

if文の条件式

■ 関係・等価演算子

◆ 例

if (x > 30) ... xが30より大きいとき

if (x <= a) ... xがa以下のとき

if (x == 1) ... xが1のとき

注意: 数学的に「=」は "**==**" を使う. "="だと代入になる.

■ 論理演算子

◆ 複数の条件が書ける. &&の方が||より優先順位が上.

if (0 <= X && x <= 100) ... xが0以上100以下のとき

if (x < 0 || x > 100) ... xが0未満又は100より大きいとき

if (a > b || c == d && d < a) は
if (a > b || ((c == d) && (d < a))) と同じ

| 数学 | 演算子 |
|--------|-----|
| > | > |
| < | < |
| \geq | >= |
| \leq | <= |
| = | == |
| ≠ | != |

| | 演算子 |
|-----|-----|
| not | ! |
| and | && |
| or | |

switch文

```
switch (整数式) {  
    case 定数式1:  
        .....  
        break;  
    case 定数式2:  
        .....  
        break;  
    case 定数式3:  
        .....  
        break;  
    default:  
        .....  
}
```

- switchの後の整数式の値によって制御を分岐させることができる.
- 整数式の値に一致するcase文があればそこから実行する. どれもあてはまらなかった場合, default文が実行される.
- break; を実行すると, このswitch文を抜ける. break; がないと, 次のcase文も実行されるので注意.

for文

```
for (初期化式; 継続条件式; 再設定式) {  
    繰り返し実行する文  
}
```

- (1) ループに入る前に**初期化式**を一回だけ実行
- (2) **継続条件式**が偽なら, for文実行終了
- (3) 繰り返し実行する文を一回実行
- (4) **再設定式**を実行
- (5) (2)の条件式チェックに戻る.

※ セミicolon ; の位置に注意.

```
#include <stdio.h>  
  
int main(void)  
{  
    int i, sum;  
  
    sum = 0;  
    for (i = 1; i <= 100; i++) {  
        sum += i;  
    }  
    printf("Answer : %d\n", sum);  
    return 0;  
}
```

while文

```
while (条件式) {  
    繰り返し実行する文  
}
```

- (1) 条件式が偽ならwhile文を終了
- (2) 繰り返し実行する文を
 一回実行
- (3) (1)に戻る

```
#include <stdio.h>  
  
int main(void)  
{  
    int i, sum;  
  
    i = 1; sum = 0;  
    while (i <= 100) {  
        sum += i;  
        i++;  
    }  
    printf("Answer : %d\n", sum);  
    return 0;  
}
```

do~while文

```
do {  
    繰り返し実行する文  
} while (条件式);
```

- (1) 繰り返し実行する文を一回実行
- (2) 条件式が偽ならdo~whileを終了
- (3) (1)に戻る

注意: 最後の while (条件式) の
後に ; が必要

```
#include <stdio.h>  
  
int main(void)  
{  
    int i, sum;  
  
    i = 1; sum = 0;  
    do {  
        sum += i;  
        i++;  
    } while (i <= 100);  
    printf("Answer : %d\n", sum);  
    return 0;  
}
```


配列

- 同じ型のデータを複数作りたいときに使う.
- 配列の中はint型, double型, char型などが使える.
- 変数と同じく, 宣言が必要.

例: `double num[10];`

- ◆ double型の変数が10個確保される.
- ◆ []の中の数字を添字(そえじ)という.
- ◆ `num[0], num[1], ..., num[9]`はdouble型変数と同じ.
- ◆ 添字は 0 から始まることに注意.上の例ではnum[10]は作られない.
- ◆ 宣言のとき, []内に変数を使うことはできない.
- ◆ 宣言された配列に対しては, []内に変数を使って参照することが可能.

多次元配列

- 多次元配列の場合は [] を複数並べて宣言

例: `int data[3][5];`

int型二次元配列の宣言.

配列長5の配列を3つ確保したことになる.

- 初期値

`int b[9]={1,2,3,4,5,6,7,8,9};`

などと初期値を与えることもできる.

`int b[]={1,2,3,4,5,6,7,8,9};`

と添字を省略することも可能. (多次元配列のときは, 一番左の [] の中の添字だけ省略可能)

- 注意

- ◆ 配列の添字の範囲に注意. 範囲外の添字でアクセスしてもコンパイル時にエラーは出ない.
- ◆ 配列はまとまって確保される.

| |
|------------|
| data[0][0] |
| data[0][1] |
| data[0][2] |
| data[0][3] |
| data[0][4] |
| data[1][0] |
| data[1][1] |
| data[1][2] |

⋮

乱数

- 擬似乱数を発生させるには関数 rand()を使う.
- rand()は0からRAND_MAXで指定される範囲の整数値乱数を発生する. RAND_MAXはstdlib.hで宣言されている.
- rand()を使うときにはstdlib.hをインクルードする.
- rand()を1回実行すると, int型の値が1個戻り値として得られるので, たとえば10個の乱数列を得るにはrand()を続けて10回実行する.

```
#include <stdio.h>
#include <stdlib.h> /* rand() を使うのに必要 */
#define N 10

int main(void)
{
    int i;
    double data[N];    /* N個の配列の宣言 */

    for (i=0; i<N; i++) data[i]=rand()/(RAND_MAX + 1.0);
    .....
}
```

乱数発生方法の例

■ 0以上1未満の乱数を得るには

```
#include <stdlib.h>  /* rand() を使うのに必要 */  
double dnum;  
  
dnum = (double) rand() / (RAND_MAX + 1);
```

または

```
dnum = rand() / (RAND_MAX + 1.0);
```

■ 0以上N未満の乱数を得るには

```
dnum = rand() / (RAND_MAX + 1.0) * N;
```

■ 0以上N以下の整数乱数を得るには

```
int inum;  
inum = (int)(rand() / (RAND_MAX + 1.0) * (N + 1));
```

乱数のシード

- srand()はシード(初期値)を与える関数
- srand()は引数の整数に応じて乱数の系列を変更する. 同じシードを与えると同じ乱数を生成できる. どの乱数表を選択するのかに相当するため, rand()を実行する前に一度実行すればよい.
- 毎回違う乱数を発生させるには現在時刻をシードに与えるのが定番. time()は1970年1月1日0時0分0秒からの秒数を返す. 引数に NULL と書くこと.

```
#include <stdlib.h>  /* rand() を使うのに必要 */
#include <time.h>

srand((unsigned)time(NULL));
.....
```

次回予定

■ C言語復習(2)

- ◆ 関数
- ◆ ポインタ
- ◆ 文字列
- ◆ ファイル入出力
- ◆ 構造体

本日の演習

- キーボードから整数を2個入力し, その2数の +, -, *, / を計算し表示するプログラム(kadai1_1.c)を作成せよ.
- 次ページのプログラムを参考に, 次の4x4の行列のかけ算を行うプログラムを作成せよ. (kadai1_2.c)

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \end{pmatrix} \times \begin{pmatrix} 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 \\ 25 & 26 & 27 & 28 \\ 29 & 30 & 31 & 32 \end{pmatrix}$$

※ Course N@viにてプログラムを提出(締め切り:本日中)
Cプログラミングを選択後, 第1回演習というのが一番下にある
ので, そこから提出すること.

plus.c

```
#include<stdio.h>
#define DIM 4
int mat1[DIM][DIM]={1,2,3,4,
                    5,6,7,8,
                    9,10,11,12,
                    13,14,15,16}};
int mat2[DIM][DIM]={17,18,19,20,
                    21,22,23,24,
                    25,26,27,28,
                    29,30,31,32}};

/*
  行列の和
*/
void plus(int array1[][DIM], int array2[][DIM],
          int ans[][DIM])
{
    int i,j;

    for (i=0; i<DIM; i++)
        for (j=0; j<DIM; j++)
            ans[i][j]=array1[i][j]+array2[i][j];
}

/* 右に続く */
```

plus.c

```
/*
  行列の表示
*/
void print_matrix(int array[][DIM])
{
    int i,j;

    for (i=0; i<DIM; i++) {
        for (j=0; j<DIM; j++) {
            printf("%d ", array[i][j]);
        }
        printf("\n");
    }
}

int main(void)
{
    int i,j,ans[DIM][DIM];

    plus(mat1,mat2,ans);
    print_matrix(ans);

    return 0;
}
```