

Cプログラミング

第2回 C言語の復習(2)

青柳 滋己
aoyagi@aoni.waseda.jp

本日の講義内容

■ C言語の復習(2)

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

■ 演習

前回の演習の解答例(1)

kadai1_1.c

```
#include <stdio.h>

int main(void)
{
    int i,j;

    printf("Please input number 1: ");
    scanf("%d", &i);
    printf("Please input number 2: ");
    scanf("%d", &j);

    printf("%d + %d =%d\n", i, j, i+j);
    printf("%d - %d =%d\n", i, j, i-j);
    printf("%d * %d =%d\n", i, j, i*j);
    if (j == 0) {
        printf("devide by 0 error\n");
    } else printf("%d / %d =%f\n", i, j, (float)i/j);

    return 0;
}
```

前回の演習の解答例(2)

kadai1_2.c

```
.....  
void multiple(int array1[][DIM], int array2[][DIM], int ans[][DIM])  
{  
    int i,j,k,tmp;  
  
    for (i=0;i<DIM; i++) {  
        for (j=0;j<DIM;j++) {  
            tmp=0;  
            for (k=0;k<DIM;k++) {  
                tmp += array1[i][k]*array2[k][j];  
            }  
            ans[i][j]=tmp;  
        }  
    }  
}  
.....
```

関数

- Cプログラムは関数の寄せ集めで構成される.
 - ◆ mainも関数である. Cプログラムは1つ以上の関数で構成される.
- printfやscanfなども関数である.
- 自分で関数を作ることができる.
- 同じような処理をする部分を関数として独立させる
 - ◆ プログラムが見やすくなる.
 - ◆ 複数人で開発する際, 内容に応じて分割できる.

関数の定義の仕方

- 関数は次のように定義する.

```
戻り値の型 関数名(仮引数の型 仮引数名, ....)
{
    関数内でのみ使用する変数の宣言

    関数での処理の内容
    return 戻り値;
}
```

```
double calc(double x, double y)
{
    double answer;

    answer= x * x + y * y;
    return answer;
}
```

- 関数名に続く()内に書く変数を引数という. 関数を呼び出すときの値がこの引数の変数にコピーされる. この変数はこの関数内でのみ使える.
- main関数と同じく, 定義の先頭で, この関数内で使える変数が引数とは別に定義できる.
- 引数は複数持てるが, 戻り値は一つだけ.
- 戻り値が不要な関数のときは, 関数をvoidで定義する. 引数が不要な場合は関数名に続く()内にvoidと書く.
- return文は関数の処理内容のどこにでもいくつでも書け, 実行したら関数の処理が終了する. voidで宣言した関数の実行途中で終了したい場合はreturn; だけ書けばよい.

プロトタイプ宣言

- 関数呼び出しより後に関数の定義が来る場合、呼び出しする場所より前に、関数をプロトタイプ宣言しておく必要がある。
- プロトタイプ宣言では関数の型、関数名、引数の型を宣言する。
- プロトタイプ宣言では引数の変数名は省略可。(書いても良い)

```
#include <stdio.h>

/* プロトタイプ宣言 */
double calc(double, double);

int main(void)
{
    double num;

    num = calc(10.3, 8.4);
    ....
}

/* 関数の定義 */
double calc(double x, double y)
{
    ....
}
```

メモリとアドレス

- メモリにはbyte単位にアドレスが割り当てられている.
- 変数を宣言することにより, どこかに必要な領域が確保され, そこに値が保存される.
 - ◆ char型 1byte
 - ◆ int型 4byte
 - ◆ double型 8byte
- メモリ内容はアドレスを使うことで参照できる.

あるアドレスのメモリ

アドレス	内容
0x10000000	0x38
0x10000001	0x9f
0x10000002	0x03
0x10000003	0x00
0x10000004	0x38
0x10000005	0x38
0x10000006	0x38
0x10000007	0x38
0x10000008	0x38
0x10000009	0x38
0x1000000a	0x38
0x1000000b	0x38
0x1000000c	0x38
0x1000000d	0x38
0x1000000e	0x38

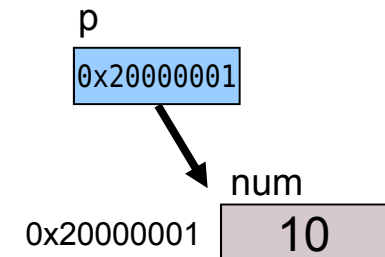
ポインタ

- 変数のアドレスを知るには変数名の前に & をつける

```
int num;  
  
printf("num=%d, address = %p¥n", num, &num); /* %pでアドレス表示 */
```

- アドレスを記憶する変数も存在 ⇒ **ポインタ変数**
- ポインタ変数の宣言は、変数名の前に * をつける

```
int num=10;  
int *p, *q; /* pとqはint型変数を指すポインタ変数 */  
p=&num;     /* ポインタ変数pにnumのアドレスを代入 */  
.....  
int func(int *x, int *y) /* ポインタ変数の引数 */  
{.....
```



- ポインタ変数の型は、ポインタが指す変数の型を指定する.

ポインタ変数の利点

- C言語では、関数呼び出し時の引数の受け渡しの際、値はコピーされ(call by value), 関数の戻り値も一つしかない.
- 引数に変数へのポインタを渡すことで、呼び出された関数内で変数の値を直接書き換えることができる.

cf. scanfで引数変数に&をつけていたのはアドレスを渡すため

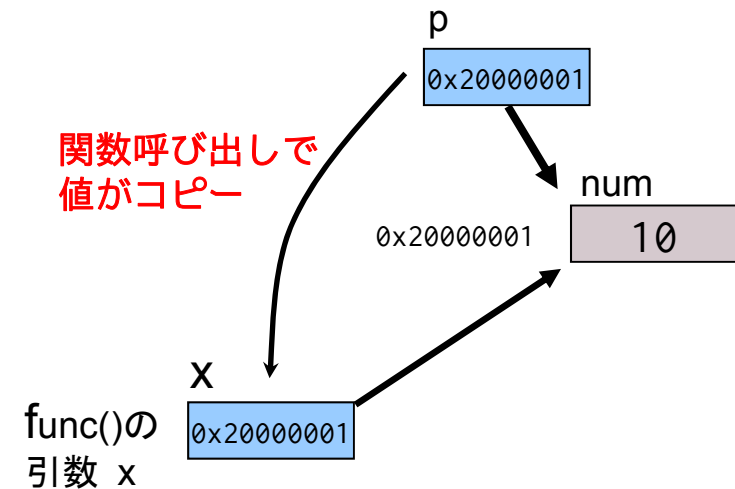
- ポインタ変数にアドレスの値(例えば0x20000001など)を直接代入してはいけない.

```
int num=10;
int *p;

p=&num;
func(p);
/* func(&num); */

.....
```

```
void func(int *x)
{
    *x = 20;
}
```



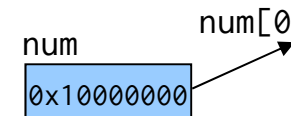
配列とポインタ

- 配列名は実はポインタと考えることができる.

例: 右は `int num[10];`

で確保されたメモリの様子の一部

- 配列名 `num` は配列の先頭の要素 `num[0]` を指すポインタと同等. ただし, 変数ではないので代入等はできない.
- 配列はメモリ上に連続して確保される.
- 配列 `num` の `i` 番目の要素のアドレスは `&num[i]` である.



アドレス	内容
0x10000000	0x38
0x10000001	0x9f
0x10000002	0x03
0x10000003	0x00
0x10000004	0x38
0x10000005	0x38
0x10000006	0x38
0x10000007	0x38
0x10000008	0x38
0x10000009	0x38
0x1000000a	0x38
0x1000000b	0x38
0x1000000c	0x38
0x1000000d	0x38
0x1000000e	0x38

文字

- 文字を扱うには, char型(1byte)を使う.
- char型の変数一つに, 文字一つを記憶できる.
- 文字は'A', 'B', 'c'のように' 'で囲む.
- 文字は実際にどのような数値になっているかは, man ascii で見ることができる.
- scanf, printfでの文字の入出力では, %c を使用.
- この講義では半角文字のみを扱う

```
char ch;
```

```
ch='A';      /* char型変数chにAという文字を代入 */  
scanf("%c", &ch); /* 一文字入力 */  
printf("%c", ch); /* 一文字出力 */
```

ASCIIコード表

■ 文字は下表の数字として扱う

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
0x0	¥0 (NUL)	(制御文字)	空白文字	0	@	P	`	p
0x1	(制御文字)	(制御文字)	!	1	A	Q	a	q
0x2	(制御文字)	(制御文字)	"	2	B	R	b	r
0x3	(制御文字)	(制御文字)	#	3	C	S	c	s
0x4	(制御文字)	(制御文字)	\$	4	D	T	d	t
0x5	(制御文字)	(制御文字)	%	5	E	U	e	u
0x6	(制御文字)	(制御文字)	&	6	F	V	f	v
0x7	(制御文字)	(制御文字)	'	7	G	W	g	w
0x8	(制御文字)	(制御文字)	(8	H	X	h	x
0x9	¥t (HT)	(制御文字))	9	I	Y	i	y
0xa	¥n (LF)	(制御文字)	*	:	J	Z	j	z
0xb	(制御文字)	ESC	+	;	K	[k	{
0xc	(制御文字)	(制御文字)	,	<	L	\	l	
0xd	CR	(制御文字)	-	=	M]	m	}
0xe	(制御文字)	(制御文字)	.	>	N	^	n	~
0xf	(制御文字)	(制御文字)	/	?	O	_	o	(制御文字)

文字リテラル

- 文字列を"で囲んだものを文字リテラルと呼ぶ。基本的に定数扱い。

例

```
"abcdef"  
"9876543210"
```

- 文字リテラルの内部表現

例

```
"abcdef" → 'a' 'b' 'c' 'd' 'e' 'f' '¥0'  
            0x61 0x62 0x63 0x64 0x65 0x66 0x0
```

最後に終端を意味する'¥0'が付加される

文字列

- 文字列はchar型の配列として扱う.
- どこまでが文字列なのか, 判別するために終端文字'¥0'を使う.
- 表示にはprintfを使う際には, "%s"で指定する.
- 文字列読み込みにはscanfではなく, fgetsで1行全部読み込むのが良い.
- 関数引数にする場合はポインタを使う.

```
char str[50], str1[100]:
```

```
str[0]='A';  
str[1]='b';  
str[2]='c';  
str[3]='¥0';  
printf("%s",str);  
scanf("%s", str1);  
fgets(str1, 100, stdin);  
printf("%s",str1);
```

文字列の操作

```
char ss[5]={ 'a', 'b', 'c', 'd', '¥0'};  
ss[]="opq"; /* これは初期化時以外は駄目 */  
ss[0]='o', ss[1]='p'; ss[2]='q'; ss[3]='¥0';  
/* あるいは strcpy を使う */  
strcpy(ss, "opq");
```

- 文字列のコピーは配列のコピーになるので、各要素を一つずつ代入するか、あるいはstrcpyを使う必要がある。
- strcpyを使うときは終端に'¥0'が付加されるのも考慮し、十分大きな配列を用意すること。

文字や文字列を扱う関数

■ 文字

#include <ctype.h>

isalpha, isdigit, islower, isupperなど.

■ 文字列

#include <string.h>

strlen, strcpy, strcat, strcmpなど.

■ 文字と入出力

getchar, ~~gets~~, putchar, puts,
fgetc, fgets, fputc, fputsなど.

■ その他

#include <stdlib.h>

atoi, atofなど

注: getsは使用してはいけない. 代わりにfgetsを使用すること. 同様に文字列読み込み時にscanfを使用してはいけない.

※ manコマンドで使い方を確認すること.

ファイル

- ファイルにはテキスト形式とバイナリ形式の2種類が存在
 - ◆ テキスト...文字のみが書かれているファイル. cat等のプログラムで画面に表示しても問題ない.
 - ◆ バイナリ...a.outファイルなど, 文字以外の情報が入ったファイル. cat等で画面表示させようとすると端末がおかしくなる.
- C言語におけるファイルの扱いは上のテキストとバイナリで異なる.

ファイル入出力をするための手続き

- Cでのファイル入出力関数には高レベル入出力関数と低レベル入出力関数の2種類が存在.
- 本講義では高レベル入出力関数しか扱わない
- Cでのファイル入出力は次の手順で行う.
 1. ファイル名を指定してファイルをオープンする.
fopen 関数を用いる
 2. ファイルの読み書きを行う.
fscanf, fprintf, fgetc, fputc など
fread, fwrite(バイナリ)
 3. ファイルをクローズする.
fclose関数
- fopen(),fclose()は必ず行うこと.

ファイルのオープン・クローズ

- ファイル入出力はFILE構造体へのポインタであるファイルポインタを使う.
- ファイルオープンするにはfopenを使う

```
fopen(ファイル名, モード);
```

- 使い方の例

```
#include <stdio.h>

FILE *fp;

fp=fopen("testdata.txt", "r");
if (fp == NULL) { /* オープン失敗時の処理 */
    .....
}
```

- ファイル名やモードは"で囲むこと.
- 一つのファイルポインタで同時に複数のファイルをオープンすることはできない.

ファイルのオープン・クローズ

■ fopenの主なモードは右の通り.

■ ファイルのクローズは, fcloseの引数にファイルポインタを指定する.

```
fclose(fp);
```

モード	意味
"r"	読み出し専用
"w"	書き込み専用. ファイルが存在しなければ作成される. ファイルが存在すると前のファイルの中身はなくなる
"a"	追加書き込み専用. ファイルが存在しなければ作成される. ファイルが存在すると, 前の中身の後に追加される.

fopenが失敗したとき

- 存在しないファイルを読み込もうとした時や、ファイルが他プロセスによって既に開いている時は、fopenが失敗する。
- fopenが失敗すると、NULLポインタが返されるので、エラーチェックを行う。exit();はそこでプログラムの実行が終了する。引数の値がプログラムの実行値になる。

```
#include <stdio.h>
#include <stdlib.h>

FILE *fp;
.....
fp=fopen("testdata.txt", "r");
if (fp == NULL) {
    printf("fopen error\n");
    exit(1);
}
```

```
/* 左のプログラムと同じ */
#include <stdio.h>
#include <stdlib.h>

FILE *fp;
.....
if ((fp=fopen("testdata.txt", "r")) == NULL) {
    printf("fopen error\n");
    exit(1);
}
```

構造体

- 複数の変数をまとめて組にして扱える.
- 複数の属性を持つものを扱うときに使うと良い.
- まず最初に構造体の宣言を行う. 宣言はその構造体のテンプレートを定義する. 変数を続けて宣言してもよい.

```
struct タグ名
{
    第1メンバの宣言;
    第2メンバの宣言;
    第3メンバの宣言;
    .....
} 変数名;
```

```
.....
struct book
{
    char title[256];
    char author[256];
    int pages;
};

int main(void)
{
    struct book a1;

    strcpy(a1.title, "C nyuumon");
    strcpy(a1.author, "H. Hayashi");
    a1.pages = 322;

    .....
}
```

構造体変数

- 構造体変数を使うには、使用宣言する.

```
struct タグ名 変数名;
```

- 構造体のメンバにアクセスするには、"."を使う.
- タグ名を使わずに変数宣言することもできる.

```
struct  
{  
    char title[256];  
    char author[256];  
    int pages;  
} a1;
```

- =で代入が可能. 各メンバがそのままコピーされる.
- メンバに他の構造体を持つことも可能.

```
.....  
struct book  
{  
    char title[256];  
    char author[256];  
    int pages;  
};  
  
int main(void)  
{  
    struct book a1, b1;  
  
    .....  
    printf("title=%s\n", a1.title);  
    printf("author=%s\n", a1.author);  
    printf("pages=%d\n", a1.pages);  
    b1 = a1;  
}
```


構造体ポインタ

- 構造体変数を指す構造体ポインタは他のポインタ変数と同様に宣言する.
- 構造体ポインタの指す構造体のメンバを参照するには, アロー演算子" \rightarrow "を使う.

注: アロー演算子を使わない場合は演算子の優先順位に注意. 例えば

```
b->pages = 100;
```

は

✗

```
*b.pages = 100;
```

(\therefore `*(b.pages) = 100;` と解釈される)

○

```
(*b).pages = 100;
```

```
.....
struct book
{
    char title[256];
    char author[256];
    int pages;
};

int main(void)
{
    struct book a1, *b;

    b=&a1;
    printf("title=%s\n", b->title);
    printf("author=%s\n", b->author);
    printf("pages=%d\n", b->pages);
    ....
}
```

構造体と配列

- 構造体の配列は通常の変数の宣言と同じように宣言する.
- 構造体メンバへは"."でアクセス可能. ポインタを使って"->"を使うこともできる.

```
struct book
{
    char title[256];
    char author[256];
    int pages;
};

int main(void)
{
    struct book a[3], *b;
    int i;

    a[0].pages=200; /* 初期化 */
    .....
    for (i=0; i<3; i++) {
        b=&a[i];
        printf("title=%s\n", b->title);
        printf("author=%s\n", b->author);
        printf("pages=%d\n", b->pages);
    }
    .....
}
```

構造体を関数引数にする

■ 構造体を引数に渡すことができる.

- ◆ 構造体メンバの値は全部コピーされる.
- ◆ 構造体配列の場合は普通の配列と同じくポインタが渡される.
- ◆ 呼び出した関数内で値を変更したい場合は必ずポインタで渡すこと.

```
.....  
void viewstrct(struct book b)  
{  
    b.pages=150;  
    printf("title=%s¥n", b.title);  
    printf("author=%s¥n", b.author);  
    printf("pages=%d¥n", b.pages);  
}  
  
int main(void)  
{  
    struct book a1;  
    .....  
    strcpy(a1.title, "book1");  
    strcpy(a1.author, "author1");  
    a1.pages=200;  
    viewstrct(a1);  
  
    printf("pages=%d¥n", a1.pages);  
    .....  
}
```

次回予定

■ 連立一次方程式の解法(1)

本日の演習(1/2)

- 以下のkadai2_1.cのswap関数の引数と本体を作成してプログラム(kadai2_1.c)を完成させよ.

kadai2_1.c

```
#include <stdio.h>

void swap(    ) /* 引数を書くこと */
{
    /* 関数本体を書くこと */
}

int main(void)
{
    int x=100, y=200;

    printf("x=%d, y=%d\n", x, y);
    swap(&x, &y);
    printf("Swap: x=%d, y=%d\n", x, y);
    return 0;
}
```

実行すると下のようになる.

```
% gcc -o kadai2_1 kadai2_1.c
% ./kadai2_1
x=100, y=200
Swap: x=200, y=100
%
```

本日の演習(2/3)

4. 左のkadai2_2.cをコンパイルし実行した様子が右の実行結果である. 1番目と2番目に表示される数を答えよ.

kadai2_2.c

```
#include <stdio.h>

int main(void)
{
    int data[]={10,8,6,4,2};
    int *p=data, i;

    printf("%d¥n", (*p)++);
    printf("%d¥n", *p++);
    printf("%d¥n", --(*p));
    for (i=0; i<5; i++)
        printf("%d ", data[i]);
    printf("¥n");
    return 0;
}
```

実行結果

```
% gcc -o kadai2_2 kadai2_2.c
% ./kadai2_2
 ←①
 ←②
7
11 7 6 4 2
%
```

答えを kadai2_2.txt という名前のファイルに書いて提出.
どうしてそういう表示になるか理由も書くとよい.

本日の演習(3/3)

■ 文字列を逆転しファイル出力するプログラム

- ◆ キーボードから文字列を1行入力すると、入力した行を逆順にした文字列のファイル data.txt を作成するプログラムを作れ(kadai2_3.c).

```
% gcc -o kadai2_3 kadai2_3.c
% ls
kadai2_3  kadai2_3.c
% ./kadai2_3
Input?: Sample desu.
% ls
data.txt  kadai2_3  kadai2_3.c
% cat data.txt
.used elpmaS
%
```

※ Course N@viからプログラム, 答えを提出
(締め切り: 本日中)