

C言語講座第二回 2018

分岐とループと乱数

前回のお詫び

- ▶ 環境構築に手間取ってしまい説明ができなかったことをお詫び申し上げます。
- ▶ 時間の都合上前回の資料の内容は理解していることを前提に進めさせていただきまます。わからない場合は個別に聞きに来て下さい。

前回の補足(1.入出力型指定子早見表)

入出力型指定子	printf()	scanf()
int	%d	%d
long	%d	%ld
float	%f	%f
double	%f	%lf
char	%c	%c

代入・単項演算子

代入演算子	記法	意味
+=	A += B	A = A + B
-=	A -= B	A = A - B
*=	A *= B	A = A * B
/=	A /= B	A = A / B
%=	A %= B	A = A % B

単項演算子	記法	意味
++	++AまたはA++	A = A + 1
--	--BまたはB--	B = B - 1

前置と後置の違いは扱いません。

比較・論理演算子

条件式：真ならtrue(=1)を、偽ならfalse(=0)を返す

比較演算子	記法	意味
==	A == B	AとBは 等しい か
<	A < B	AはB 未満 か
>	A > B	AはB より大きい か
<=	A <= B	AはB 以下 か
>=	A >= B	AはB 以上 か
!=	A != B	AとBは 等しくない か
論理演算子	記法	意味
&&	(A >= 100) && (A <= 200)	Aは100以上 かつ 200以下か
	(A <= 100) (A >= 200)	Aは100以下 または 200以上か
!	!(A >= 100)	Aは100以上 ではない か

200 >= A >= 100みたいに書くと予想外の動きをするので注意

if else

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

```
int main() {  
    int a;  
    a = 50;
```

```
    if (a <= 100) {  
        printf("aは100以下です。¥n");  
    }  
    else {  
        printf("aは100以下ではありません。¥n");  
    }
```

```
    return 0;
```

```
}
```



{ }を使わなかった場合はその行のみ適用

else if

else{if}と同じ

```
#include <stdio.h>

int main() {
    int a;
    a = 50;

    if (a <= 100) printf("aは100以下です。¥n");
    else if(a >= 80) {
        printf("aは80以上100未満です。¥n");
    }
    else{
        printf("aは80未満です。¥n");
    }
    return 0;
}
```

switch

switch(変数)の変数と同じ
caseの{以下を実行
break;でswitch{}から脱出

どのcaseにも該当しない場合は
defaultの{以下を実行する。
defaultは書かないことも可能。

switchおよびcaseの
変数・定数は式でもよい。

```
#include <stdio.h>
int main(){
    int a = 0;
    switch (a){
        case 0:{
            printf("aは0です。 \n");
            break;}
        case 1:{
            printf("aは1です。 \n");
            break;}
        default:{
            printf("エラーが発生しました。 \n");
            break;}
    }
    return 0;
}
```


while と for

#include <stdio.h>されたmain()の中で

```
int i = 0;
while(i < 10){
    printf("i=%d\n", i);
    i++;
}
```

//while(条件式)の
//条件式が偽になるまで繰り返す

```
int i = 0;
for(i=0; i<10; i++){
    printf("i=%d\n", i);
}
```

//for(初期化式;条件式;更新式)
//初期化式を行い、}まで行ったら更新
//条件式が偽になったら脱出
//初期化式は省略可能である
for(; i < 10 ; i ++)

演習1

変数*i*,*num*,*sum*を用意し、

while文かfor文を使って1,1,2,3,5,8,13,21 . . . の規則性を求め任意(45以下)の項(*num*番目)までの和を出力するプログラムを書け。

余裕のある人は*num*=45とした時正常に動いているか確認せよ。もし正常に動いていなかった場合修正せよ。

疑似乱数の取得

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

int main(){
    int x;
    x = rand();
    printf("x=%d\n", x);

    return 0;
}
```

疑似乱数は

$x_{n+1}=f(x_n)$ という漸化式になっており、
`rand();`をすると x_{n+1} が返ってくる。

そのため x_0 (シード値)が決まると
`rand();`を何回実行した時の値が
何になるか決まる。

任意の i に対する x_i は

$0 \leq x_i \leq \text{RAND_MAX}(=2^{15} - 1)$ であり
一様整数乱数(int)

シード値の変更

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

int main(){
    srand(1);

    int x;
    x = rand();
    printf("x=%d\n", x);

    return 0;
}
```

シード値をnにする場合はsrand(n);
プログラムの途中で書くことも出来る。
(シード値のリセット)

ランダム性を持たせたい場合は
#include <time.h>された状態で
srand((unsigned int)time(NULL));とすると
現在時刻に応じてシード値が決まる
→起動するたびに違ったシード値になる

ただしデバッグが大変になるので
ここではデフォルトの1で行きます
(何も書かなかった場合も1です)

乱数の加工

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

int main(){
    srand(1);

    int x = rand() % 201;
    int y = 50 + rand() % 151;
    printf("x=%d,y=%d\n", x, y);

    return 0;
}
```

0以上 $2^{15} - 1$ 以下の乱数をそのままでは使えないので加工しましょう。

左の例では

$$0 \leq x \leq 200$$

$50 \leq y \leq 200$ となっています。

実数乱数、半開区間については今回のおまけに記載しています。

乱数の加工 番外編1

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

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

```
int main(){
```

```
    double a;
```

```
    a = rand()/RAND_MAX;
```

```
    printf("a=%f\n",a);
```

```
}
```

0以上1以下の実数乱数を作りたい！

と思っても左のプログラムでは
上手くいきません。

これはrand()とRAND_MAXが共に
int型であるため、計算結果も
int型になってしまうからです。

乱数の加工 番外編2

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

```
int main(){
    double a;
    a = (double)rand()/RAND_MAX;
    printf("a=%f\n",a);
}
```

そこで、
rand()をdouble型にキャストします。

こうすることで
計算結果も高精度であるdouble型になり
無事に0以上1以下の実数乱数に
加工できます。

乱数の加工 番外編3

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

```
int main(){
    double a;
    a = (double)rand()/(RAND_MAX+1);
    printf("a=%f\n",a);
}
```

また、0以上1未満の乱数に加工したい場合、

左のプログラムのようにRAND_MAX+1で割ることで1を除外することができます。

加工した0以上1以下/未満の実数に任意の最大値maxを掛ければ0以上max以下/未満の実数に加工できます。

演習2

「a~z,A~Z」の52文字を使って、任意の桁数のパスワードを作り出力するプログラムを作れ。

while と break

```
int main(){
    char a;
    while(1){
        scanf("%c", &a);
        if(a == 'A'){
            break;
        }
    }

    return 0;
}
```

while(1)と書くと()内が真なので無限ループします。が、break;を使うことで解消できます。

break;はswitch文の他にwhile,for,doからも脱出することができるからです。

左のプログラムではaに一文字代入しそれがAであった場合のみbreak;でwhile(1)から脱出するようになっています。

while と continue

```
int main(){
    char a;
    while(1){
        scanf("%c", &a);
        if(a != 'A'){
            continue;
        }
        break;
    }

    return 0;
}
```

break;とは逆に、continue;を使うと
continue;以下の命令を飛ばして
(更新式→)条件式の判定→再ループ/脱出
という動きをさせることができます。

と偉そうに書きましたが
書いている本人は使ったことはありません。

ポインタとは

変数のメモリ上のアドレスを格納するもの。

アドレスとは：

address【名詞】...住所、宛名、番地

メモリ上の場所のこと。

使用例1

int *pa;のように*を付けて宣言する。

名前は任意。

(int* paと書いてもよい。)

*をつけないpaに

変数aのアドレス(&a)を入れると

*paがaを指すようになり

aの中身を*paでも表せるようになる。

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

```
int main(){
```

```
    int a = 3;
```

```
    int *pa;
```

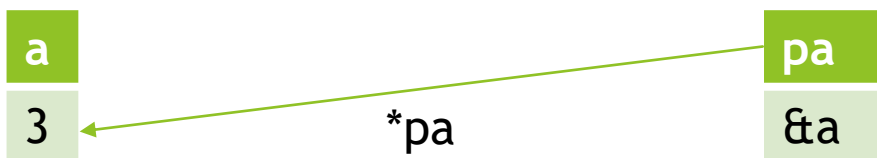
```
    pa = &a;
```

```
    printf("%d\n", *pa); //3
```

```
    printf("%d\n", pa); //aのアドレス
```

```
    return 0;
```

```
}
```



注意点

アドレスの代入は宣言時でもよい。

また、aと*paは
同じものを指しているため、
aを変えれば*paは変わり
*paを変えればaも変わる。

```
#include <stdio.h>

int main(){
    int a = 3;
    int *pa = &a;
    printf("%d\n", a); //3

    *pa = 5;
    printf("%d\n", a); //5

    return 0;
}
```

&と*について

&:アドレス演算子

&aは、aのアドレスを表す。

scanf("%d", &a)も、aのアドレスが指し示す場所(=a)を書き換える関数といえる。

*:脱参照演算子

こちらは逆にアドレスの指す先の値を表す。

ポインタ変数を使うことで、元の変数の値を間接的に変更することができる。

間接変更

「*paを使うとaに触れずにaを変更できる」という特徴は
次回の講座「関数」で利用します。

(仮引数を渡す関数ではaに直接触れられないため)

なので今「なぜこうするのか」は
あまり気にしないでください。

ポインタ要点

```
int *pa;
```

int型ポインタ変数paの宣言

```
pa = &a;
```

ポインタ変数に参照させたい変数のアドレスを代入

```
*pa = 3;
```

ポインタ変数の指す値(=a)に3を代入

配列

ポインタから少し離れて、配列の話。

int a[N];と書くと、
一気にN個の変数(a[0]~a[N-1])を
定義することができる。

```
#include <stdio.h>

int main(){
    int a[3];
    a[0] = 2;
    a[1] = 4;
    a[2] = 6;

    printf("%d\n", a[1]); //4

    return 0;
}
```

定義と初期化

```
int a[] = {p, q, r,...,z};
```

と書くと定義と

同時に初期化を行うことができる。

[]内のNは省略可能。

(N=初期化する要素の数になる)

また、各要素をa[n]で表せる。

右の例ではscanfで入力したiに対し
a[i]を表示するようになっている。

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

```
int main(){  
    int a[] = { 2, 4, 6 };  
    int i;  
  
    scanf("%d", &i);  
    printf("%d\n", a[i]);  
  
    return 0;  
}
```

内部の話

配列変数はアドレスが連番になる

別々の変数

p
0

q
1

r
2

配列変数

a[0]	a[1]	a[2]	a[3]	a[4]
0	1	2	3	4

配列と脱参照演算子

$a[N]$ において、 a は $a[0]$ のアドレス $\&a[0]$ を意味する
従って $*a$ は $\&a[0]$ の値であり $a[0]$ となる

また、 $a[0], a[1], \dots, a[N-1]$ のアドレスが連番であることを利用すると
例えば $*(a+1)$ は $a[1]$ となる

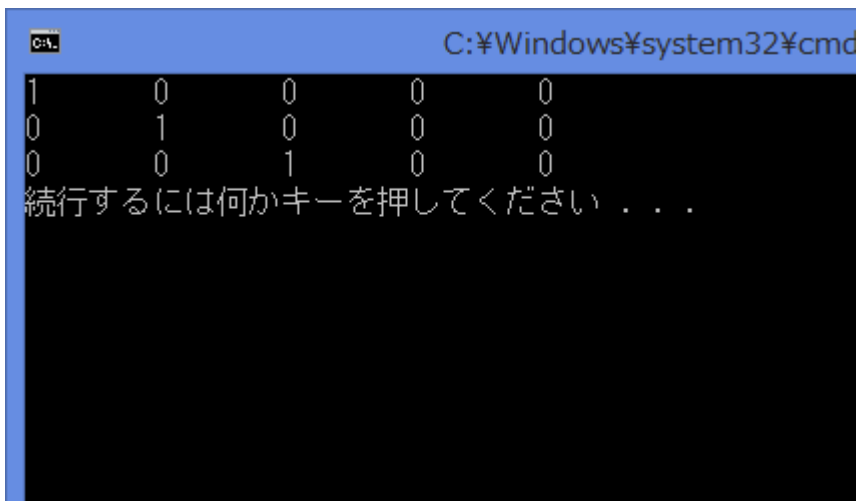
したがって $a[x]$ と $*(a+x)$ は同じものである

二次元配列変数

a[N][M]と書くと

N行M列の行列の形で配列変数を
定義できる。

右のプログラムを実行すると
下のようなになる。



```
C:\Windows\system32\cmd
1      0      0      0      0
0      1      0      0      0
0      0      1      0      0
続行するには何かキーを押してください . . .
```

```
#include <stdio.h>
int main(){
    int a[3][5];
    int i, j;
    for (i = 0; i < 3; i++){
        for (j = 0; j < 5; j++){
            if (i == j){
                a[i][j] = 1;
            }
            else{
                a[i][j] = 0;
            }
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

表で説明すると

	a[][0]	a[][1]	a[][2]	a[][3]	a[][4]
a[0][]	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][]	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][]	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]

こんな感じ。

演習3

使用例を参考にして、`a[7][20]`に7人の名前(英語)を格納し、それを利用して名前の若い順に表示するプログラムを作成せよ。