C言語講座 第5回

ポインタ

はじめに

- 第5回の講座でこれから話すポインタはC言語の中でとても重要な機能です。（何故かはあとで話します。）

- しかし、C言語を始めたばかりの人の大半が、ポインタを理解できず、つまずくことが多いです。
（そしてそのままにします。）

- なので、無理に今回だけでわかろうとせずに、わからないことがあったらいつでも気軽に聞きに来てください。

ポインタとは

- ずばりはっきりというと、
  「変数のアドレス」を記憶する変数です。

- これだけではなんなのかわかりませんね。
  （知っている人もいるかもしれませんけど）

- なので、順に説明していきます。

アドレス

変数を記憶しておく場所を示すもの（住所）

変数とアドレス

- 今まで使ってきた「int～」や「char～」などの変数は全てメモリ上に一時的に記憶（保存）されている。

```
int a;
char b;
```

こんな感じで使うよね。

- その変数が、メモリ上のどこに保存されているか、つまり、変数を記憶しておく場所を示すもの（住所）が、アドレスである。

アドレスを表示させる

- アドレスを実際に使うには変数の前に「&」をつける。
（printf()でアドレスを表示させるときは「%p」を使う。）

```
#include<stdio.h>

int main(void){
        int a;

        printf("aのアドレスは%p¥n",&a);

        return 0;
}
```

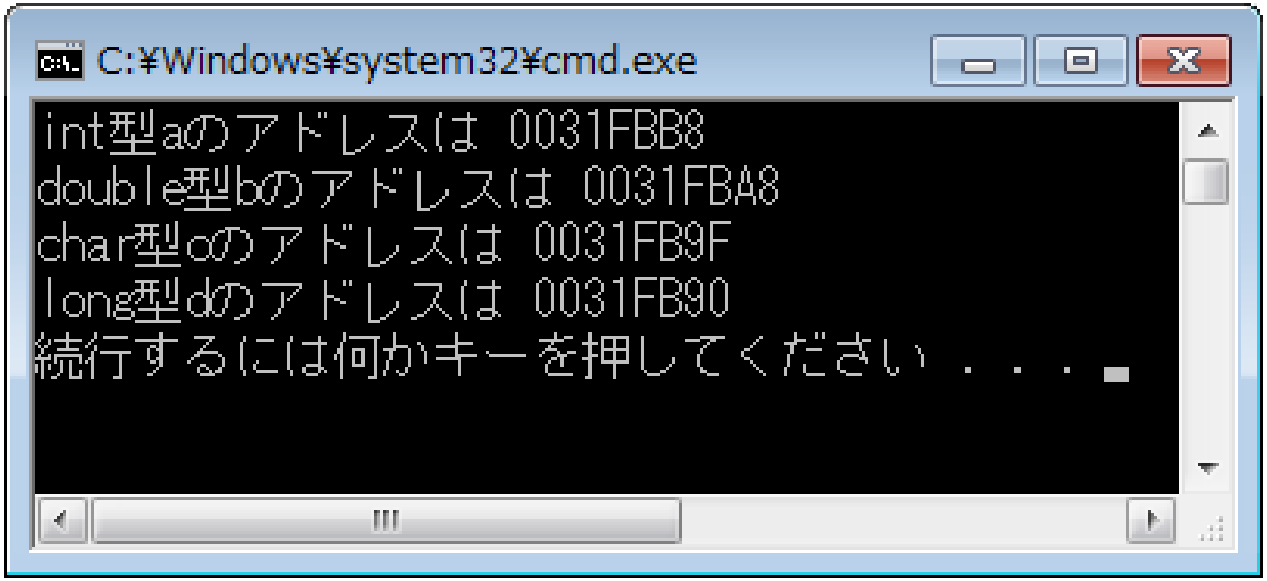
こんな感じ

実行してみる

- こんな感じでいろいろな型の変数をいくつか書いてみます。
- どんな型の変数も変数名の前に「&」を付ければアドレスを示します。

```
#include<stdio.h>

int main(void){
        int a;
        double b;
        char c;
        long d;

        printf("int型aのアドレスは %p¥n", &a);
        printf("double型bのアドレスは %p¥n", &b);
        printf("char型cのアドレスは %p¥n", &c);
        printf("long型dのアドレスは %p¥n", &d);

        return 0;
}
```

実行結果は？

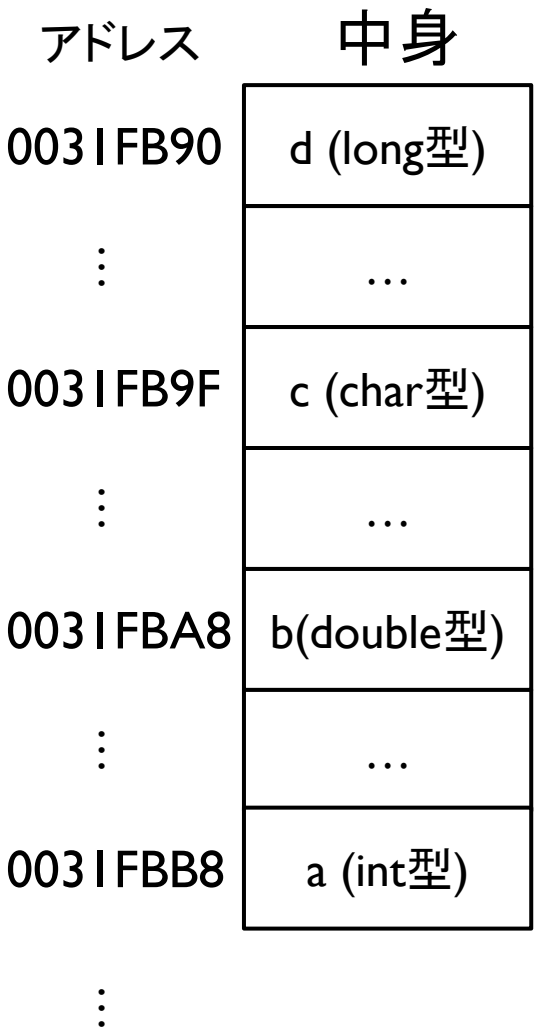


- 「～型～のアドレスは ・・・」と表示されます。
- この「・・・」が、その変数のメモリ上に保存されている場所、つまり「アドレス」です。
- （このアドレスは環境によって違うので同じ必要はないですとりあえずこんな感じのものが出れば問題ないです。）

図にしてみると

- メモリの中身（一部）を
  図にしてみるとこんな感じ。

- 変数を宣言すると、
  使っていないメモリの中から
  適当に選ばれたアドレスに
  変数が記憶されます。



scanf関数

- 今までユーザーから値を入力する際に使ってきたscanf関数ですが、以下のように記述してきました。

```
scanf("%d", &a);
```

- この二つ目の引数「&a」、第一回でとりあえず変数に「&」をつけるように説明されてました。

- scanf関数は「アドレスが示すメモリを入力された値に書き換える」といったことをします。

ポインタ(pointer)

「変数のアドレス」を記憶する変数

ポインタとは

- この講座のはじめの方に言ったこと、
  ポインタとは「変数のアドレスを入れる変数」です。

- ある変数が、メモリ上のどこに保存されているかを示す
  アドレスを保存しておけます。

- ではこれからポインタの使い方について説明します。

ポインタの宣言

- ポインタも変数なので、はじめに宣言します。
  ポインタの宣言の仕方は決まっているので覚えてください。

  int a; ⇒ int *p;

- こんな感じで、変数の前に「*」を付けます。
  型は入れたいアドレスの変数の型です。

- 今回の例ではint型の変数aのアドレスを入れたいので、
  ポインタの型はint型にします。

ポインタの使い方

- ポインタを宣言したら、まずアドレスを入れます。

```
int main(void){
        int a;
        int *p;

        p=&a;

        return 0;
}
```

pにaのアドレスを代入します。

- アドレスを使う時は変数の前に「&」をつけるとさっき言ったよね。

補足

- 最初にポインタを使うには「int *p」と書くと説明しましたが、別に「*p」という変数ではないです。
- 正確にいうと「int p」という変数に「*」をつけることによって他の変数のアドレスを変数「p」に入れて使用できるようになるといった考え方をしてください。

- 「p」という変数に「a」のアドレスを代入したいので「p=&a」となるわけです。(*p=&aという書き方はできない。)
- ただし、宣言するときに初期化する場合のみ、
以下のように書くことができます。

```
int *p=&a;
```

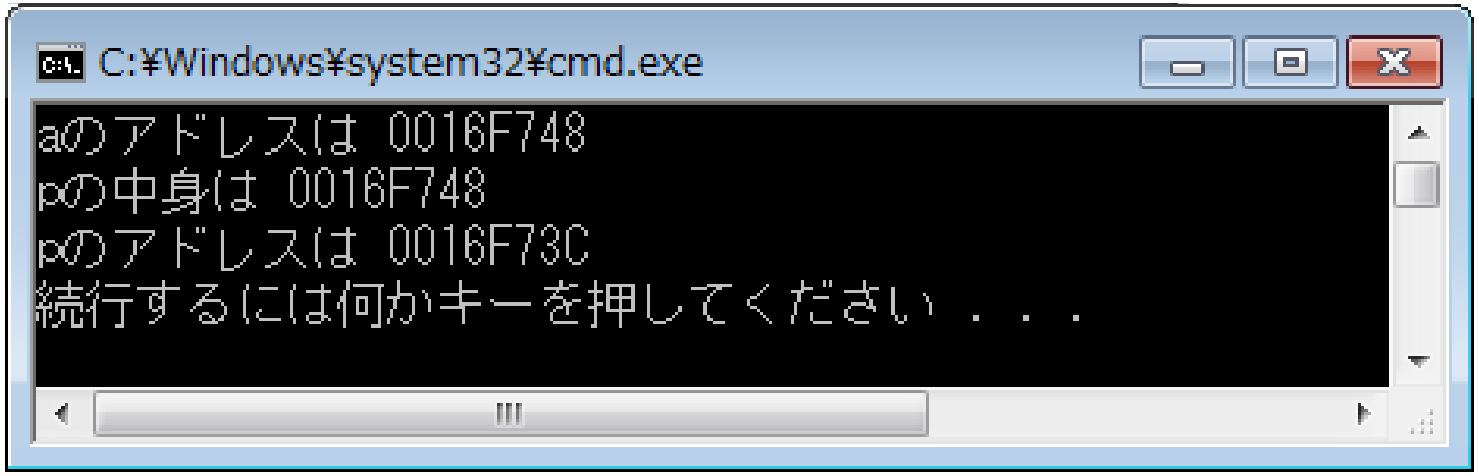
実行してみる

- さっきのプログラムはアドレスを代入するだけだったので、表示するようにprintf()を書き加えて実行してみます。

```
#include<stdio.h>

int main(void){
        int a;
        int *p;

        p=&a;

        printf("aのアドレスは %p¥n", &a);
        printf("pの中身は %p¥n", p);
        printf("pのアドレスは %p¥n", &p);

        return 0;
}
```

「*p」ではなく「p」と書きます。

実行結果



- aのアドレスとpの中身が同じになります。
- pも変数ですので、メモリに保存されています。
  よって、しっかりとアドレスがあります。
  （aとpは違う変数なのでアドレスは違います。）

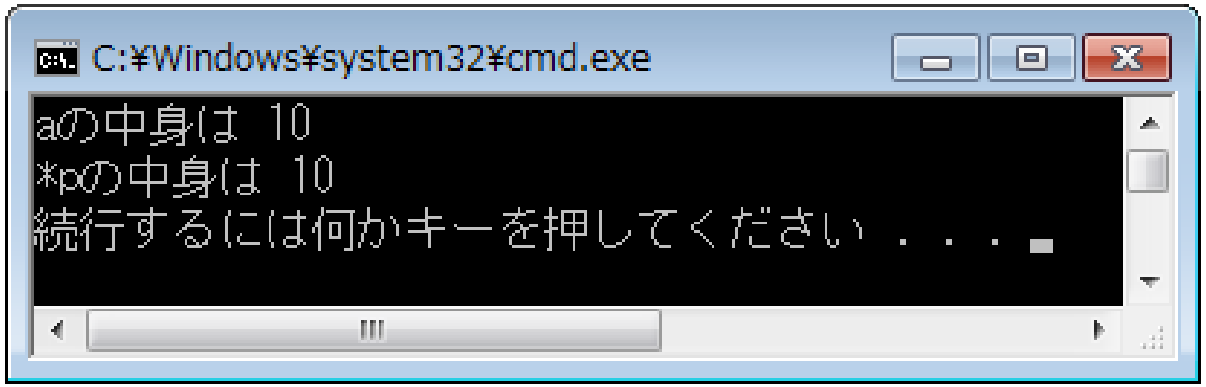
＊の使い方

- ポインタに「*」のつけた時、*のついている変数に入っているアドレスの中身を参照します。
- 次のプログラムを用いて説明します。

```
#include<stdio.h>

int main(void){
        int a=10;
        int *p=&a;

        printf("aの中身は %d¥n", a);
        printf("*pの中身は %d¥n", *p);

        return 0;
}
```

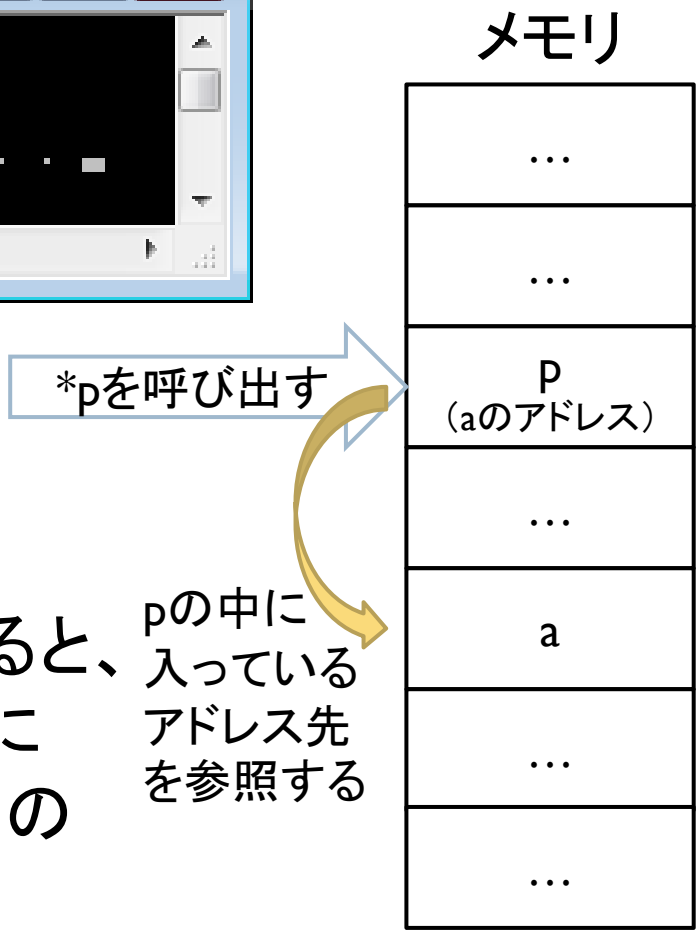
はじめに、
aに10を代入して、
そのあとにpにaのアドレスを入れています。

＊の仕組み



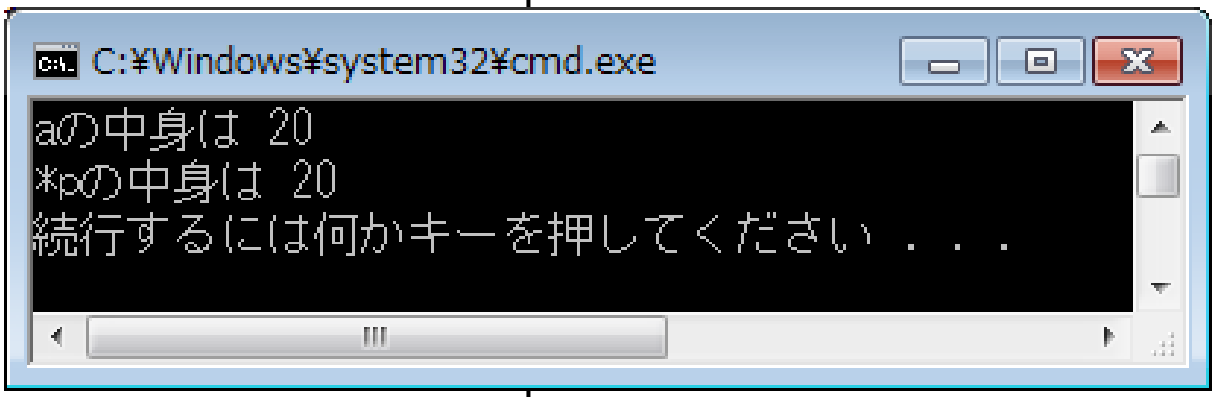
- pにaのアドレスを入れると、
  *pの値がaと同じになりました。

- *のついたポインタにアクセスすると、
  *のついている変数（ここではp）に
  入っているアドレス（ここでは&a）の
  中身（ここではa）を参照します。



- では次のようにpにaのアドレスを入れたあとに、「a」の値を変えてみます。(「*p」の値は変更しません。)

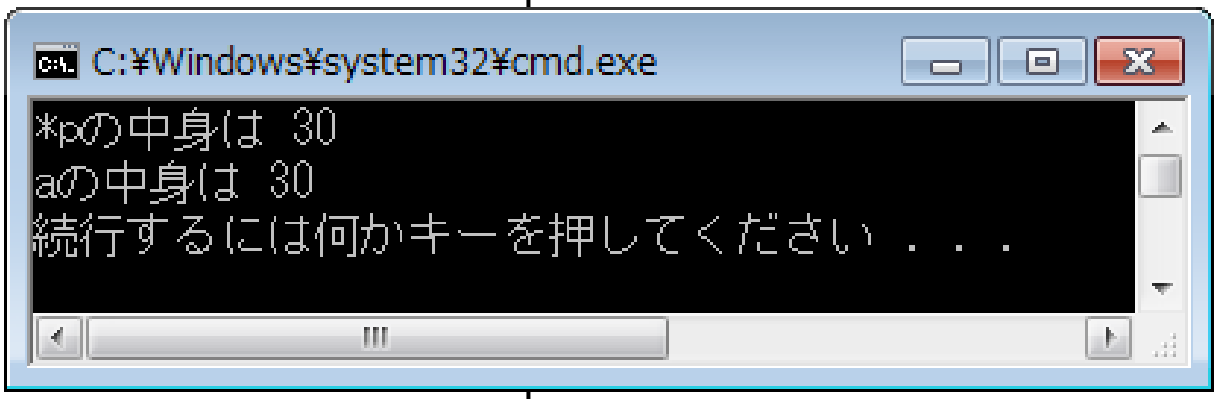
```
#include<stdio.h>

int main(void){
        int a=10;
        int *p=&a;

        a=20;

        printf("aの中身は %d¥n", a);
        printf("*pの中身は %d¥n", *p);

        return 0;
}
```



*pの値は変更していませんが、*pは、pに入っているアドレスの中身を参照するので、aの値を変えると、*pの値もaと同じになります。

- では今度は逆にpにaのアドレスを入れたあとに、「*p」の値を変えてみます。(「a」の値は変更しません。)

```
#include<stdio.h>

int main(void){
        int a=10;
        int *p=&a;

        *p=30;

        printf("*pの中身は %d¥n", *p);
        printf("aの中身は %d¥n", a);

        return 0;
}
```



*pは、pに入っているアドレスの中身を参照するので、*pの値を変えことで、aの値に変更することもできます。

- *のついたポインタにアクセスすると、その都度そのポインタに入っているアドレス先を参照します。
- つまり、「 *のついたポインタ」と「そのポインタに入っているアドレス先」は常に連動し、同じ値を示します。
- この機能こそが、ポインタがC言語の中で重要な機能であるということの理由です。

※注意 WARNING

- アドレスを渡さずにポインタを使用するのは
  絶対にしないでください。

- ポインタはアドレスを得てその中身を間接的に書き換えたり参照する物です。
- ポインタにアドレスを渡さないと、そのポインタの中には宣言した際の適当な値が入っています。
- その状態で「*p」を使うと「p」に入っているどこのものかもわからないアドレスを参照してしまいます。
- もし、その参照したアドレスが他のシステムで使っている場所なら、そこを書き換えたりするとシステムがクラッシュする可能性があります。（大事な課題,OS…etc.）

ポインタの使用例

実際にどうやって使うのか

ポインタを使用する？

- 今までポインタの仕様について説明しましたが、正直、「aの値を変えたいなら、わざわざ*pなんて使わないで、直接aに値を代入すればいいのでは？」と思う人もいるかもしれません。

- 確かにこれまでの例のような使用方法は、実際にはほとんど行いません。（上記の通り、直接値をいじればいいのですから。）

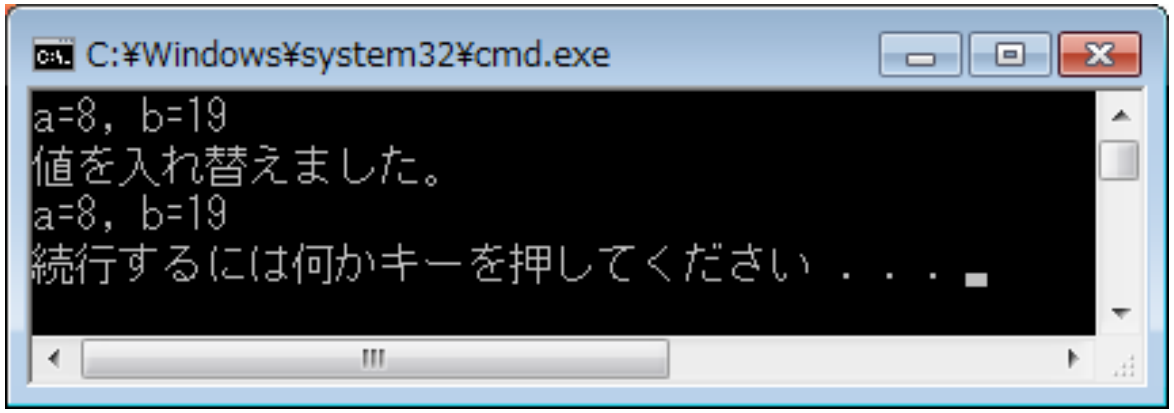
- では、なぜポインタというものがC言語においてとても重要なのでしょうか。

使用例

- こちらの例を使って説明します。
- このswap関数は、自作関数で、二つの引数を渡して、それらを入れ替えるという関数です。
- 一見正しそうに見えます。

```
#include<stdio.h>

void swap(int x, int y){
        int tmp;
        tmp=x;
        x=y;
        y=tmp;
        printf("値を入れ替えました。¥n");
}

int main(void){
        int a=8, b=19;
        printf("a=%d, b=%d¥n", a, b);

        swap(a, b);
        printf("a=%d, b=%d¥n", a, b);
        return 0;
}
```

実行結果



- さらっと「入れ替えました。」なんて表示してますが、結果は見ての通りまったく入れ替わっていません。

- 先ほどのプログラムでは、機能しない理由を図を使って説明します。

main()関数

```
int a=8, b=19;

printf("a=%d, b=%d¥n", a, b);

swap(a, b);

printf("a=%d, b=%d¥n", a, b);

return 0;
```

a,bの値をswap関数に渡します

```
int x=a;
int y=b;
```

swap()関数

```
int tmp;

tmp=x;
x=y;
y=tmp;

printf("値を入れ替えました。¥n");
```

- main関数からswap関数が呼び出されたときに、swap関数にint x, int yという変数が宣言され、それぞれ渡された値を代入します。
- しかし、このx,yはmain関数のa,bの値を代入しただけの別の変数なのです。そのため、このx,yを入れ替えても、main関数の変数が入れ替わらないのです。（違う変数だからね。）

使用例（改訂）

- では、ポインタを使ってこの問題を解決します。
- swap関数の引数をint*型に変えて、swap関数を呼び出す際に、それぞれの変数のアドレスを渡してあげるようにします。

```
#include<stdio.h>

void swap(int *x, int *y){
        int tmp;
        tmp=*x;
        *x=*y;
        *y=tmp;
        printf("値を入れ替えました。¥n");
}

int main(void){
        int a=8, b=19;
        printf("a=%d, b=%d¥n", a, b);

        swap(&a, &b);
        printf("a=%d, b=%d¥n", a, b);
        return 0;
}
```

main()関数

```
int a=8, b=19;

printf("a=%d, b=%d¥n", a, b);

swap(&a, &b);

printf("a=%d, b=%d¥n", a, b);

return 0;
```

a,bのアドレスをswap関数に渡します

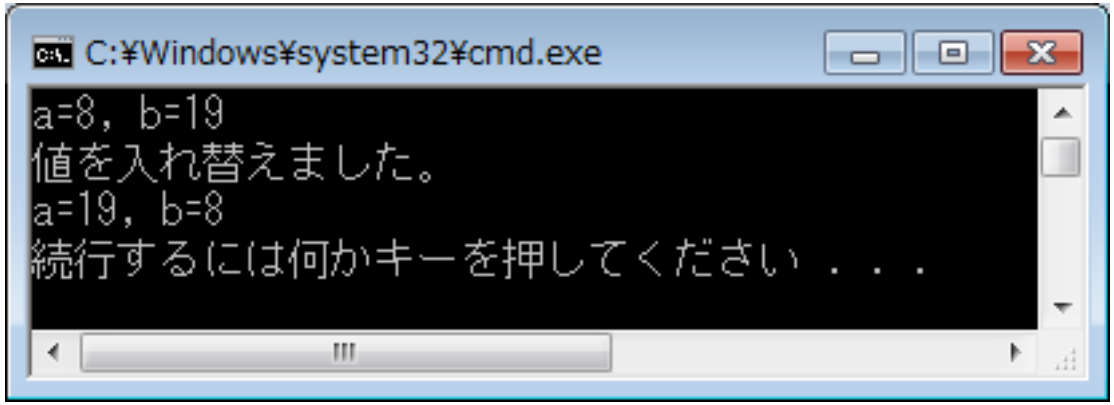
```
int *x=&a;
int *y=&b;
```

swap()関数

```
int tmp;

tmp=*x;
*x=*y;
*y=tmp;

printf("値を入れ替えました。¥n");
```

- 今回は、main関数から
  swap関数が呼び出されたときに、swap関数にint *x, int *yというポインタが宣言され、渡されたアドレスを代入します。
- x,yにはa,bのアドレスが入っているので、*x,*yはメモリ上のa,bを参照します。よって、*x,*yを変更すればa,bも変更されます。

実行結果



- 実行結果もしっかりと入れ替わるようになりました。
- これで、swap関数が正常に使えるようになりました。

練習問題

- main関数でx, yを宣言して、main関数の外からx, yの値を変更する関数を作成してください。
- set関数で値を入力するようにして、x, yの値が変わるように作ってください。
- 関数の作り方はわかるよね。

```
戻り値の型 関数名(引数){
        文;
}
```

- とりあえず、前回までの範囲で、できるところまではこんな感じで書けると思います。
- とりあえずここまでは書いて問題を進めよう。

```
#include<stdio.h>

void set(〇〇);

int main(void){
        int x, y;

        printf("xの値を入力してください:");
        set(〇〇);
        printf("yの値を入力してください:");
        set(〇〇);

        printf("x=%d,y=%d¥n", x, y);
        return 0;
}

void set(〇〇){
        〇〇
}
```

まとめ

- このように直接いじることのできないところを、ポインタを利用することで、扱うことができるようになり、より自由度の高いプログラミングが可能になります。

- しかし、ポインタを使えば、処理はより複雑になりますのでプログラムが読みにくくなったり、本来アクセスできないところからのアクセスが可能になることで意図しない値の変更が起こったりする可能性もあります。

- ですので、慣れないうちは無理にポインタを使わないで、本当に使用する必要があるところだけにポインタを使い、なるべくわかりやすく書くのがいいかもしれません。

- これでC言語講座 第5回 ポインタについての説明は以上になります。今回の講座では、ポインタについての基礎的なことをやりました。
- 実際にはポインタにはまだまだいろいろな機能があります。
- 今回の講座は、まずポインタがどういうものであるかを理解してもらうための講座ですので、説明はしません。（今日のことだけでも十分にプログラムを書くことができます。）
- これらの機能は必要になってから勉強するのでも遅くないです。

・配列のポインタ
・ポインタの配列
・ポインタに対するポインタ
（ダブルポインタ）
・関数ポインタ
（関数を参照するポインタ）

長い講座でしたが、お疲れ様でした。

わからないことがあったらいつでも質問に来てください。

部室に来てくれてもいいですし、メールをくれてもいいです。