ポインタのインクリメントとかの話

知っていればそんなの当たりまえって話だけど、知らなければなんだこれってなるのが世の常である。

この資料には普通のインクリメントとポインタのインクリメントの違いをさっとまとめる。

※`int型`は4byte、`char型`は1byte、ポインタは4byteという前提とする。

普通のインクリメント

普通のインクリメントは値が1増える。

コード

```
int main() {
    int a = 0;
    return 0;
}
```

メモリ

値	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	a																					

intが4byteとすると、メモリ上には変数`a`のためにメモリが4byte確保されて、値は0になっている状態。

コード インクリメントを追加

```
int main() {
    int a = 0;
+   a++;
    return 0;
}
```

メモリ

値	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	a																					

変数`a`の値が`1`増えて、メモリは👆このような状態になる。

これが一般的というか、よくあるインクリメントである。

ポインタのインクリメント その1

まずは`char`のポインタを用意してインクリメントしたらどうなるかを確認してみよう。

コード

```
int main()
{
    char* p = 0;
    return 0;
}
```

メモリ

値	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	p																					

ポインタ変数`p`のために4byteのメモリを確保し、値は0(いわゆるヌルポ)の状態になる。

(※ポインタ変数を4byteとしたが64bit環境だと8byteになる、32bit環境なら4byte)

コード インクリメントを追加

```
int main()
{
    char* p = 0;
+   p++;
    return 0;
}
```

メモリ

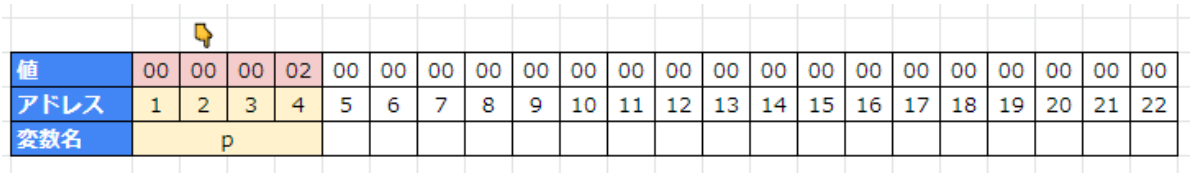
値	00	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	p																					

ポインタをインクリメントすると値が1増える。またポインタの意味的にはアドレス1番を指している状態になる(指のとこ)

コード さらにインクリメントを追加

```
int main()
{
    char* p = 0;
    p++;
+   p++;
    return 0;
}
```

メモリ



値	00	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	p																				

ポインタの値がまた1つ増え`2`になる。またポインタの意味的にはアドレス2番を指している状態だ。

この時点では`int`型の変数をインクリメントしたときと何も変わらない。

ポインタのインクリメント その2

今度は`int`のポインタを用意してインクリメントしたらどうなるかを確認してみよう。

コード

```
int main()
{
    int* p = 0;
    return 0;
}
```

メモリ

値	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	p																					

`int`型だろうと`char`型だろうとポインタは4byteなので、メモリの状態はこうなる。

コード インクリメントを追加

```
int main()
{
    int* p = 0;
+   p++;
    return 0;
}
```

メモリ

値	00	00	00	04	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
アドレス	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
変数名	p																					

`char`型のポインタの時は値が1増えただけだったが、`int`型のポインタをインクリメントすると値は`4`増加した。

このようにポインタのインクリメントは**必ずしも1増えるわけではない**のである。

ポインタをインクリメントした時の値の増え方

ポインタをインクリメントしたとき、`char`の時は1増え、`int`の時は4増えた。

同じインクリメントでも値の増え方が違うが、何が決め手になるのか。

答えを言ってしまえばそれは、**ポインタの型**によって決まる、もう少し言えばポインタの型のサイズで決まる。

`char`は1byteなので、`char`のポインタをインクリメントすれば**1増える**

`int`は4byteなので、`int`のポインタをインクリメントすれば**4増える**

そういうカラクリである。