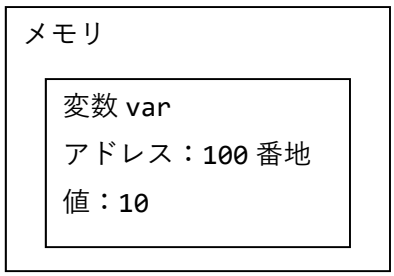
C 言語 015 ポインタ

【ポインタ】

```
int var = 10;
```

変数を宣言すると、メモリ上に領域が確保されます。
作られた変数には「アドレス」が割り当てられ、この「アドレス」を使用してデータのやり取りを行うことを「ポインタ」と称します。



【アドレス】

アドレスは 16 進数の数値で表現しますが、テキストでは分かりやすく「100 番地」と表現します。
アドレスは 1 バイトごと割り振られており、メモリの状況によって変動するので、毎回同じアドレスが割り当てられるとは限りません。

```
int var = 10;
```

アドレスの先頭を仮に 100 番地、int 型を 4 バイトとするとメモリの中は下記の通りになり、プログラムではアドレス 100 番地～104 番地を変数 var として扱います。

変数 var

アドレス	100	101	102	103
値	10			

【ポインタ変数】

アドレスを記録する変数をポインタ変数と称します。

＜ポインタ変数の宣言＞

```
データ型* ポインタ変数名;
もしくは
データ型 *ポインタ変数名;
```

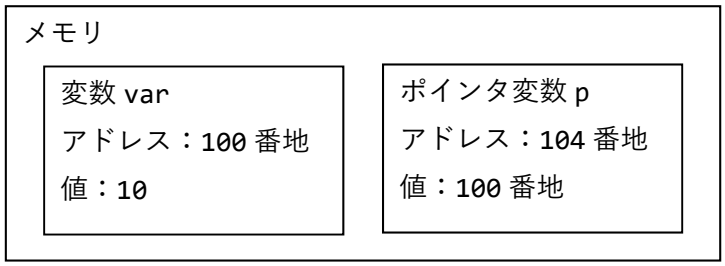
*はデータ型の後ろか、変数名の前に付けます。

＜アドレスの代入＞

```
ポインタ変数名 = &変数名;     /* 変数名に&を付ける */
```

＜例文＞

```
int  var;    /* 変数 var */
int* p;      /* ポインタ変数 p */
var = 10;    /* var に 10 を代入 */
p = &var;    /* p に var のアドレスを代入 */
```



ポインタ変数 p は変数 var のアドレスを記録しています。

【ポインタで値を書き換える】

ポインタ変数の参照先の変数に値を代入します。

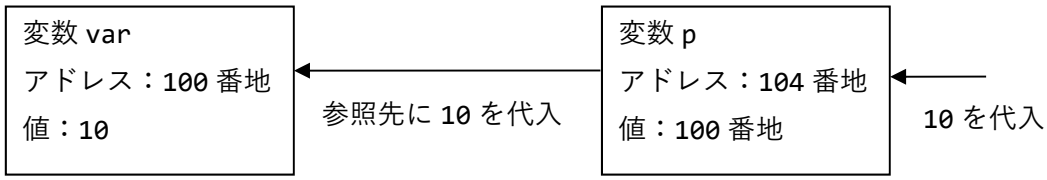
＜参照先に代入＞

```
*ポインタ変数名 = 値・変数・式
```

＜例文＞

```
int  var;                /* 変数 var */
int* p;                  /* ポインタ変数 p */
p = &var;                /* p に var のアドレスを代入 */
*p = 10;                 /* p の参照先に 10 を代入 */
printf("%d¥n", var);     /* var を参照。「10」と出力 */
printf("%p¥n", p);       /* p を参照。アドレスが出力 */
printf("%d¥n", *p);      /* p の参照先を参照。「10」と出力 */
```

最終的に変数 var に 10 が代入されます。



変数 p は変数 var を「指し示す」ので「ポインタ」なのです。

【ポインタを関数の引数にする】

関数の引数の渡し方には「値渡し」と「参照渡し」があります。

実引数にポインタを使用することで「参照渡し」にすることができます。

＜値渡し＞

```
int main(void) {
    void func(int);
    int var = 10;
    func(var);
    printf("%d¥n", var);    /* 10 と出力 */
    return 0;
}

void func(int arg) {
    arg = 20;
}
```

値渡しは、実引数 var の値を仮引数 arg に渡しています。

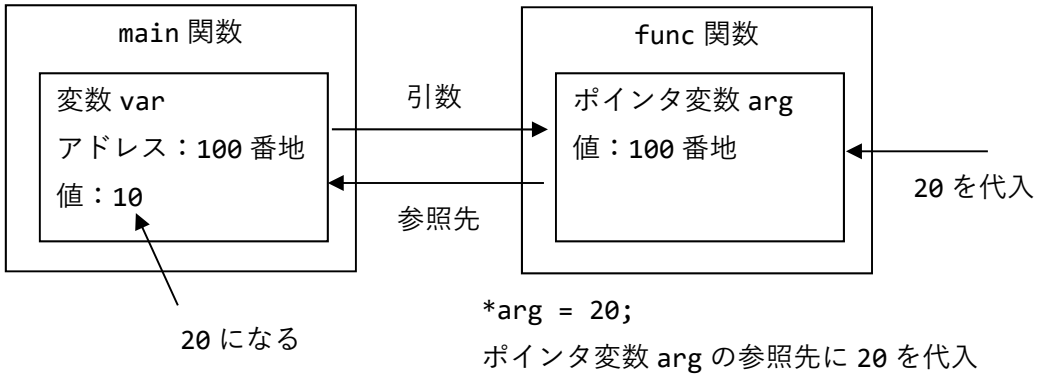
arg に 20 を代入しても、var には関係ありません。

＜参照渡し＞

```
int main(void) {
    void func(int*);
    int var = 10;
    func(&var);             /* アドレスを渡す */
    printf("%d¥n", var);    /* 20 と出力 */
    return 0;
}

void func(int* arg) {       /* 引数がポインタ変数 */
    *arg = 20;              /* 参照先に代入 */
}
```

参照渡しは、実引数 var のアドレスを仮引数 arg に渡しています。

arg の参照先に 20 を代入すると、var の値が変わってしまいます。



「値渡し」は、仮引数を修正しても実引数は修正されません。
「参照渡し」は、仮引数を修正すると実引数も修正される場合があります。

【const 付きポインタ変数】

const が付いたポインタ変数は 2 通りの書き方があります。

```
A)  const データ型* 変数名 = 初期値;
B)  データ型* const 変数名 = 初期値;
```

A の場合はポインタ変数の「参照先の値」を変更できません。

B の場合はポインタ変数の「アドレス」を変更できません。

＜例文 1＞

```
int var = 10;
const int* p = &var;    /* const 変数は宣言時に初期化すること */
*p = 20;                /* p の参照先の値を変更できない。エラーになる */
```

＜例文 2＞

```
int var1 = 10;
int var2 = 20;
int* const p = &var1;    /* var1 のアドレスを代入 */
p = &var2;               /* p のアドレスを変更できない。エラーになる */
```

＜関数の引数に const＞

引数がポインタ変数の場合は「参照渡し」になります。

しかし、引数のポインタ変数に「const」を付けた場合、その引数の参照先を変更できないため、実引数の値も変更できません。

仮引数 arg の参照先の値は変更されないことを保障しています。

```
void func(const int* arg){
```

【ポインタを関数の戻り値にする】

戻り値のデータ型の後ろか、関数名の頭に*を付けて、return でアドレスを返します。

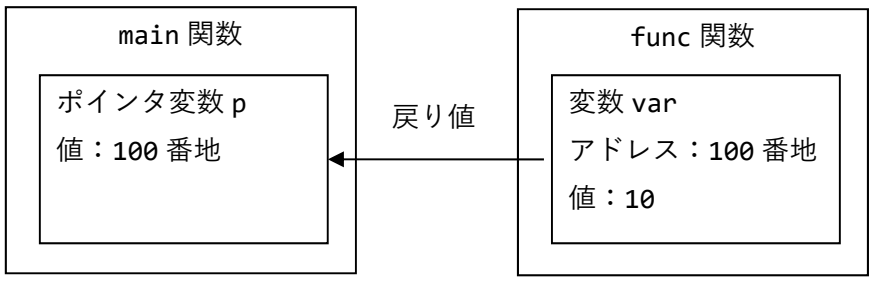
＜例文＞

```
int main(void) {
    int* func(int*, int*);          /* 関数プロトタイプ宣言 */
    int  var1 = 10;
    int  var2 = 20;
    int* p = func(&var1, &var2);    /* 戻り値にアドレスが返ってくる */
    printf("%d の方が大きい", *p);
    return 0;
}

int* func(int* arg1, int* arg2) {    /* 戻り値にポインタを指定する */
    int* ret;                        /* ポインタ変数 */
    if(*arg1 > *arg2){
        ret = arg1;
    }
    else{
        ret = arg2;
    }
    return ret;                      /* アドレスを返す。*/
}
```

＜ポインタを関数の戻り値にする際の注意＞
下記のコードをコンパイルすると警告が発せられます。

```
int main(void) {
    int* func(void);
    int* p = func();     /* 100 番地を受け取る */
    printf("%d", *p);    /* 100 番地の値を出力する */
    return 0;
}

int* func(void) {
    int var = 10;        /* var のアドレスを仮に 100 番地とする */
    return &var;         /* 100 番地を返す */
}
```

予想通り、実行結果は「10」と出力されますが、なぜ警告が出るのでしょうか？
それは「変数 var」が「ローカル変数」だからです。
ローカル変数は関数が終了された時点で消去されてしまいます。100 番地には値 10 が残っているため、実行結果は「10」となりますが、100 番地は「空き番地」になるため、他に上書きされる可能性があり、いつまでも値 10 である保障が無いからです。



関数終了後、変数 var は消去。
100 番地は「空き番地」になってしまう。

変数 var を「static 変数」にすれば、プログラムが終了するまで変数は消去されない、つまり、アドレスは空き番地にならないため、警告は出ません。

【NULL】

NULL とは「未参照」、ポインタ変数にアドレスが入ってないことを表す特殊な値です。
ポインタ変数は宣言しただけでは中身が不定値になるため、NULL を代入し初期化するのに使用します。

＜例文＞

```
int* p = NULL;       /* NULL を代入して初期化する */
if ( p == NULL) {    /* 変数 p は NULL か？ */
```

＜注意＞

「NULL」と「NULL 文字」は別物です。
NULL は「アドレス未参照」のこと。ポインタ変数で使用します。
NULL 文字は「文字列の終わり」のこと。文字配列で使用します。