C 言語 018 構造体・列挙体

【構造体】

構造体は、複数の変数をひとつにまとめて、1 個の変数にする物です。
構造体を使用するには「構造体の定義」と「構造体の変数宣言」の 2 つが必要です。

＜構造体の定義＞

```
struct 構造体名 {
    変数宣言
};
```

構造体の中で宣言された変数を「メンバ変数」と称します。

構造体の定義はグローバル領域でも関数内（ローカル）でも記述することができますが、関数内に記述すると記述した関数内でしか使用できず、他の関数からは使用できません。

＜構造体の変数宣言＞

```
struct 構造体名 変数名;
```

定義した構造体を変数にする宣言です。
グローバル変数にすることや、配列やポインタにも使用できます。

＜構造体の変数宣言と初期化＞

```
struct 構造体名 変数名 = {初期値, 初期値…};
```

＜メンバ変数への代入＞

```
変数名.メンバ変数名 = 値;
```

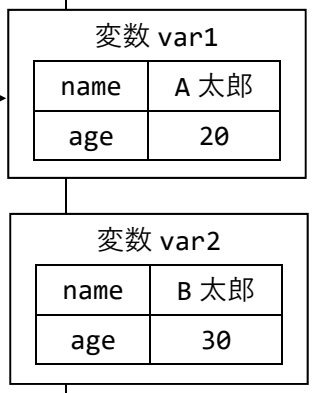
＜例文＞

```
/* 構造体の定義 */
struct personal {
    char name[10];    /* メンバ変数 name */
    int  age;         /* メンバ変数 age */
};                    /* セミコロンの付け忘れに注意*/

int main(void) {

    /* 構造体の変数宣言 */
    struct personal var1 = {"A 太郎", 20};
    struct personal var2;

    /* メンバ変数への代入 */
    strcpy(var2.name, "B 太郎");
    var2.age = 30;
```



＜構造体の定義と変数宣言を同時に行う＞

```
struct 構造体名 {
    メンバ変数の記述
} 変数名;
```

この記述を関数内に宣言すればローカル変数、グローバル領域に宣言すればグローバル変数になります。

＜構造体の定義と変数宣言と初期化＞

```
struct 構造体名 {
    メンバ変数の記述
} 変数名 = {初期値, 初期値…};
```

＜例文＞

```
struct personal {
    char name[10];
    int  age;
} var = {"A 太郎", 20};
```

【構造体を配列にする】

＜構造体の配列宣言＞

```
struct 構造体名 変数名[ 要素数 ];
```

＜例文＞

```
/* 構造体の定義 */
struct personal {
    char name[10];
    int  age;
};

int main(void) {

    /* 構造体の配列宣言 */
    struct personal array[3];

    /* メンバ変数への代入 */
    strcpy(array[0].name, "A 太郎");
    array[0].age = 20;
```

配列 array の値

添字	0	1	2
値	name: A 太郎 / age: 20	name: / age:	name: / age:

【構造体を関数の引数にする】

```
struct personal { char name[10]; int age; };    /* 構造体の定義 */

int main(void) {
    void func(struct personal);                  /* 関数プロトタイプ宣言 */
    struct personal var = {"A 太郎", 20};        /* 構造体の変数宣言と初期化 */
    func(var);                                   /* 引数に構造体 */
    return 0;
}

void func(struct personal arg) {                 /* 引数のデータ型が構造体 */
    printf("%s さん%d 歳です。¥n", arg.name, arg.age);
}
```

【構造体を関数の戻り値にする】

```
struct personal { char name[10]; int age; };    /* 構造体の定義 */

int main(void) {
    struct personal func(void);                  /* 関数プロトタイプ宣言 */
    struct personal var = func();                /* 戻り値に構造体 */
    printf("%s さん%d 歳です。¥n", var.name, var.age);
    return 0;
}

struct personal func(void) {                     /* 戻り値のデータ型が構造体 */
    struct personal ret = {"A 太郎", 20};        /* 構造体の変数宣言と初期化 */
    return ret;                                  /* 戻り値に構造体を返す */
}
```

【構造体のアドレスを関数の引数にする】

```
struct personal { char name[10]; int age; };    /* 構造体の定義 */

int main(void) {
    void func(struct personal*);                /* 関数プロトタイプ宣言 */
    struct personal var;                        /* 構造体の変数宣言 */
    func(&var);                                 /* 参照渡しにする */
    printf("%s さん%d 歳です。¥n", var.name, var.age);
    return 0;
}

/* 関数 */
void func(struct personal* arg) {               /* 引数が構造体のポインタ変数 */
    strcpy(arg->name, "A 太郎");                /* アロー演算子を使用する */
    arg->age = 20;
}
```

＜アロー演算子＞

「->」は弓矢の矢の形から「アロー演算子」と称し、メンバ変数のポインタ（参照先）を指す演算子。

＜構造体とポインタ＞

```
struct personal var = {"A 太郎", 20};    /* 構造体の変数宣言と初期化 */
struct personal* p = &var;                /* 変数 var のアドレスを記録するポインタ変数 p */
printf("%s さん¥n", var.name);            /* 変数は.（ドット）を使用する */
printf("%s さん¥n", p->name);             /* ポインタ変数は->を使用する */
```

ポインタ変数を使用して参照先の値の代入、参照を行うには「*変数名」と記述します。
p はポインタ変数なので、参照先を表示するために「*p.name」と記述するのですが、
（アスタリスク）と.（ドット）では.（ドット）の優先順位が高いため「(p.name)」（p のメンバ変数 name の参照先）と解釈されてしまいます。
そこで括弧を付けて優先順位を変え「(*p).name」（p の参照先のメンバ変数 name）と記述するか、アロー演算子を使って「p->name」と記述します。
(*p) と p-> は同義になります。

【構造体名に typedef を使用する】

```
typedef struct 構造体名 {
    メンバ変数の記述
} 新しいデータ型名;
```

＜例文＞

```
typedef struct personal {
    char name[10];
    int  age;
} data;

int main(void) {
    data var;    /* 新しいデータ型名で宣言できる */
```

【共用体】

基本的な使い方は構造体と同じです。

しかし、共用体はメンバ変数が複数あっても、どれかひとつのメンバ変数にしか値を記録することができません。

メンバ変数 A に値を代入した後にメンバ変数 B に値を代入すると、メンバ変数 A の値は上書きされ消去されます。

定義や宣言に「struct」でなく「union」を使用します。

＜共用体の定義＞

```
union 共用体名 {
    メンバ変数の記述
};
```

＜共用体の変数宣言＞

```
union 共用体名 変数名;
```

＜例文＞

```
/* 共用体の定義 */
union personal {
    char name[10];
    int  age;
};

int main(void) {
    union personal var;              /* 共用体の変数宣言 */
    strcpy(var.name, "A 太郎");      /* メンバ変数「name」に代入 */
    printf("%s さん¥n", var.name);   /* メンバ変数「name」を参照 */
    var.age = 20;                    /* メンバ変数「age」に代入 */
    printf("%d 歳¥n", var.age);      /* メンバ変数「age」を参照 */
    printf("%s さん¥n", var.name);   /* メンバ変数「name」の値は消去されている */
    return 0;
}
```

構造体と比べるとサイズが小さく済むぐらいしかメリットが無いため、近年では使用されることが無くなってきました。

【列挙体】

列挙体は複数の定数をまとめる物です。

列挙体を使用するには「列挙体の定義」と「列挙体の変数宣言」の 2 つが必要です。

＜列挙体の定義＞

```
enum 列挙体名 {
    定数名,
    定数名,
    定数名
};
```

定数名は,（カンマ）で区切って複数定義することができます。

列挙体の定義はグローバル領域でも関数内（ローカル）でも記述することができますが、関数内に記述すると記述した関数内でしか使用することができず、他の関数は使用することができません。

＜列挙体の変数宣言＞

```
enum 列挙体名 変数名;
```

＜変数への代入＞

```
変数名 = 定数名;
```

列挙体の変数には、定義で記述した定数を代入します。

＜例文＞

```
/* 列挙体の定義。定数は整数型で、宣言順に RED=0、GREEN=1、BLUE=2 と定義される */
enum collar {
    RED,
    GREEN,
    BLUE
};

int main(void) {

    /* 列挙体の変数宣言。実際は整数を格納する型 */
    enum collar var;

    /* 定数を代入。実際は整数 0 を代入している */
    var = RED;

    /* 変数 var の値は RED か？*/
    if (var == RED) {    /* 真になる */
```

＜定数は整数＞

列挙体の定数は整数なので「enum collar var;」を「int var;」、「var = RED;」を「var = 0;」と記述することは可能なのでしょうか？

答えはコンパイラにより異なります。

MinGW では可能ですが、可読性を保つためにも enum による宣言と定数を使用することを推奨します。