【クラスと構造体】

C++の「クラス」は C 言語の「構造体」と似ています。

下記の 2 つのソースコードの実行結果は同じです。

＜C 言語の構造体＞

```
#include <stdio.h>

/* 構造体の定義 */
struct rectangle {

    /* メンバ変数 */
    int vertical;    /* 縦の長さ */
    int side;        /* 横の長さ */
};

/* 面積を出力する関数 */
void display_area(struct rectangle arg) {
    printf("面積は%d¥n", arg.vertical * arg.side);
}

/* メイン関数 */
int main(void) {

    struct rectangle var1;    /* 構造体変数「var1」の宣言*/
    struct rectangle var2;    /* 構造体変数「var2」の宣言 */

    var1.vertical = 10;       /* var1 のメンバ変数「vertical」に代入 */
    var1.side = 20;           /* var1 のメンバ変数「side」に代入 */
    display_area(var1);       /* var1 の面積を出力する */

    var2.vertical = 30;       /* var2 のメンバ変数「vertical」に代入 */
    var2.side = 40;           /* var2 のメンバ変数「side」に代入 */
    display_area(var2);       /* var2 の面積を出力する */

    return 0;
}
```

＜C++のクラス＞

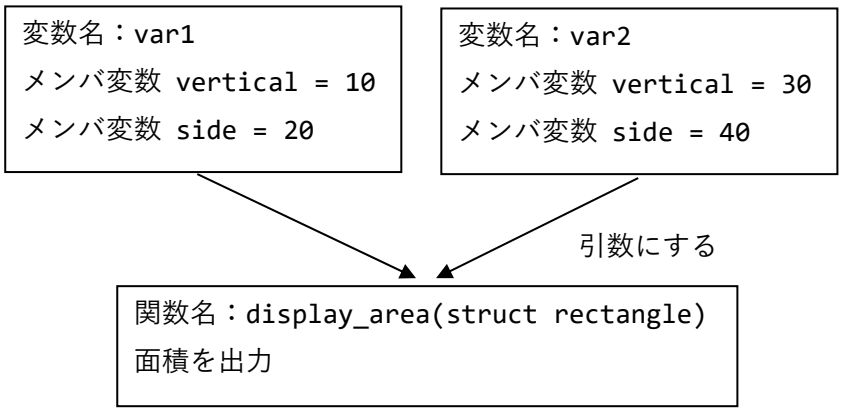
```
#include <iostream>

//クラスの定義
class rectangle {
public:

    //メンバ変数
    int vertical;    //縦の長さ
    int side;        //横の長さ

    //面積を出力するメンバ関数の宣言
    void display_area();
};

//面積を出力するメンバ関数の実装
void rectangle::display_area() {
    std::cout << "面積は" << vertical * side << std::endl;
}

//メイン関数
int main() {

    rectangle var1;          //インスタンス「var1」の宣言
    var1.vertical = 10;      //var1 のメンバ変数「vertical」に代入
    var1.side = 20;          //var1 のメンバ変数「side」に代入
    var1.display_area();     //var1 の面積を表示する。

    rectangle var2;          //インスタンス「var2」の宣言
    var2.vertical = 30;      //var2 のメンバ変数「vertical」に代入
    var2.side = 40;          //var2 のメンバ変数「side」に代入
    var2.display_area();     //var2 の面積を表示する。

    return 0;
}
```

＜実行結果＞

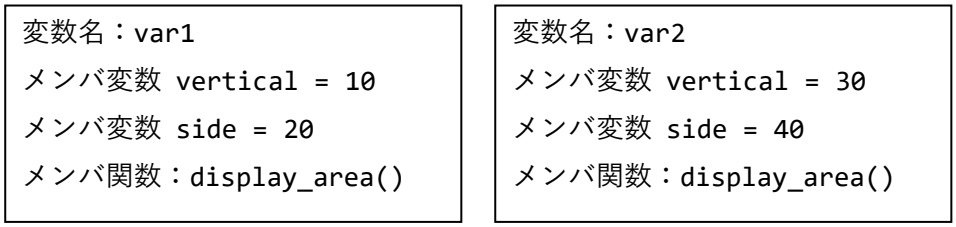
面積は 200

面積は 1200

＜C 言語の構造体のイメージ＞




＜C++のクラスのイメージ＞




C 言語の構造体が「メンバ変数」しか使用できないのに対して、C++のクラスは「メンバ変数」に加えて「メンバ関数」が使用できます。

メンバ関数はメンバ変数を使用できるので、インスタンスごとに「値の記録」と「処理（関数）」が行えます。

【クラスの作り方、使い方】

＜クラスの定義＞

```
//クラスの定義
class クラス名{
public:                                  //アクセス指定子
    データ型 変数名;                     //メンバ変数は変数宣言
    戻り値のデータ型 関数名( 引数 );     //メンバ関数はプロトタイプ宣言
};                                       //セミコロンの付け忘れに注意

//メンバ関数の実装
戻り値のデータ型 クラス名::メンバ関数名( 引数 ) {
    関数の処理を記述
}
```

＜インスタンスの宣言＞

クラス型の変数を「インスタンス」や「オブジェクト」と称する。

```
クラス名 インスタンス名;
```

クラス型の変数を宣言する事を「インスタンス化」や「インスタンスの生成」と称する。

＜メンバ変数の使用＞

```
インスタンス名.メンバ変数名
```

＜メンバ関数の使用＞

```
戻り値 = インスタンス名.メンバ関数( 実引数 )
戻り値 = インスタンス名.メンバ関数()     //引数が無い場合
インスタンス名.メンバ関数( 実引数 )      //戻り値が無い場合
```

メンバ関数もオーバーロードする事が可能。

＜インスタンスの配列化＞

```
クラス名 インスタンス名[ 要素数 ];
インスタンス名[ 添字 ].メンバ変数名
戻り値 = インスタンス名[ 添字 ].メンバ関数( 実引数 )
```

＜例文＞

```
rectangle array[3];
array[0].vertical = 10;
array[0].side = 20;
array[0].display_area();
```

array の中身

| 添字 | 0 | 1 | 2 |
|---|---|---|---|
| メンバ変数 | vertical = 10<br>side = 20 | vertical =<br>side = | vertical =<br>side = |
| メンバ関数 | display_area | display_area | display_area |

＜インスタンスのポインタ＞

```
クラス名 インスタンス名;
クラス名* ポインタ変数名 = &インスタンス名;
```

＜ポインタ変数によるメンバの使用＞

```
ポインタ変数名->メンバ変数名
ポインタ変数名->メンバ関数名( 実引数 )
```

構造体と同じく、ポインタ変数からメンバを使用する場合はアロー演算子を使用します。

＜クラスを関数の引数、戻り値にする＞

```
戻り値のデータ型 関数名( クラス名 仮引数名 ){     //クラスを引数にする。
クラス名 関数名 ( 引数 ){                        //クラスを戻り値にする。
```

これも構造体と同じです。
ポインタ、リファレンスを使用して参照渡しも行えます。

【メンバ関数のもうひとつの書き方】

```
//クラスの定義
class クラス名 {
public:
    データ型 メンバ変数;

    //クラス内にメンバ関数の実装を記述する
    戻り値のデータ型 メンバ関数名( 引数 ) {
        関数の処理を記述
    }
};
```

＜例文＞

```
class rectangle {
public:

    //メンバ変数
    int vertical;    //縦の長さ
    int side;        //横の長さ

    //面積を表示するメンバ関数
    void rectangle::display_area() {
        std::cout << "面積は" << vertical * side << std::endl;
    }
};
```

この書き方の場合、メンバ関数「display_area」は「インライン関数」になります。
インライン関数は通常の関数に比べると処理速度は速くなりますが、プログラムサイズが大きくなってしまいます。

【const 付きメンバ関数】

```
class sample_class {
public:
    int var;
    void func() const;
};

void sample_class::func() const {
    var = 100;    //メンバ変数は変更できない。この行はエラーになる。
}
```

const が付いているメンバ関数内ではメンバ変数の値は変更できません。

const を使用する事により「値の変更をしない、値の変更はできない」事を保障し、コンパイルを最適化するメリットが得られます。

【C++の構造体】

C++には「構造体」は存在しません。C++では「構造体」は「クラス」になります。（構造体の場合、メンバのアクセス指定子がデフォルトで public 属性、クラスは private 属性になる）

「struct」キーワードを使用する事ができますが、それでも「構造体」ではなく「クラス」になります。

よって構造体でもメンバ関数を使用する事ができますが、C++ではメンバ変数のみを格納する場合は構造体、メンバ関数を組み込む場合はクラスを使用するのが一般的です。