C++ 005 リファレンス

【ポインタの弱点】

関数の引数を「参照渡し」にしたい場合、C 言語では「ポインタ」を使用します。
しかし、ポインタは、ポインタ変数の値が「NULL」かもしれない事を想定する必要や、間違ったアドレスが渡される可能性を想定する必要があります。

```
void func(int* p) {
    if (p == NULL) {    /* ポインタ変数が NULL じゃないか確認 */
```

```
int var = 10;
int* p;     /* ポインタ変数が未初期化 */
func(p);    /* 間違ったアドレスを渡す事になる */
```

【リファレンス】

関数の引数を「参照渡し」にしたい場合、C++では「ポインタ」に加えて「リファレンス」が使用できます。

＜リファレンス変数の宣言＞

```
データ型& リファレンス変数名 = 参照する変数名;
もしくは
データ型 &リファレンス変数名 = 参照する変数名;
```

＜リファレンスの使い方＞

```
int  var = 10;     //変数 var を作り、値 10 を代入
int& ref = var;    //リファレンス変数 ref を作り、変数 var を参照する。
ref = 20;          //ref に値を代入すると、var に値が代入される
std::cout << var << std::endl;    //出力結果は「20」
std::cout << ref << std::endl;    //これの出力結果も「20」
```

宣言時にデータ型の末尾か、リファレンス変数名の頭に「&」を付けるだけです。
ポインタと違い、リファレンス変数に値を代入、参照する際に「&」「*」は不要です。

＜構造体のリファレンス＞

```
struct personal {
    std::string name;
    int age;
};

int main() {

    personal var;           //C++では「struct」の記述は不要
    var.name = "A 太郎";
    var.age = 20;
    personal& ref = var;    //構造体変数 var を参照する。
    ref.age = 30;

    std::cout << "名前：" << ref.name << std::endl;
    std::cout << "年齢：" << ref.age << std::endl;    //出力結果は「年齢：30」
```

＜構造体変数のリファレンス化＞

```
構造体型& リファレンス変数名 = 参照する構造体変数名;
```

リファレンス変数のメンバ変数を使用する場合は、通常の構造体と同じ「.（ドット）」を使用します。
ポインタのように「->（アロー演算子）」は使用しません。

【リファレンスの注意点】

＜リファレンス変数は必ず変数を参照しないといけない＞

```
int& ref;    //この記述はエラー
```

＜リファレンス変数と参照する変数のデータ型は同じでないといけない＞

```
double var;
int& ref = var;    //この記述はエラー
```

＜const 変数は参照できない＞

```
const int VAR = 10;
int& ref = VAR;    //この記述はエラー
```

＜参照先を変更する事はできない＞

```
int x, y;
int& ref = x;
&ref = y;    //この記述はエラー
```

＜リファレンス変数はメモリの中に領域を作らない＞

ポインタ変数はメモリの中に領域が作られる。

```
int var = 10;
int* p = &var;
```

アドレス：100 番地 変数名：var 値：10	アドレス：101 番地 変数名：p 値：100 番地

リファレンス変数はメモリの中に作られず、参照する変数と同じ領域を使用します。

```
int var = 10;
int& ref = var;
```

アドレス：100 番地 変数名：var（別名：ref） 値：10

リファレンスは 1 つの変数に別名を付けるイメージです。

だから「var=20」と「ref=20」は同じ物になるのです。

領域を作らないのでメモリの節約にもなります。

【リファレンスによる参照渡し】

```
int main() {

    void func(int&);     //関数のプロトタイプ宣言
    int var = 10;
    func(var);           //変数 var を参照渡し
    cout << var;         //出力結果は「20」

    return 0;
}

void func(int& ref) {    //引数にリファレンス変数
    ref = 20;
}
```

＜関数の宣言＞

```
戻り値のデータ型 関数名 ( データ型& 仮引数名) {
```

＜関数のプロトタイプ宣言＞

```
戻り値のデータ型 関数名 ( データ型&) {
```

＜関数の使用＞

```
関数名( 実引数 );
```

引数に「&」を付けてリファレンス変数に指定するだけです。

【const 付き引数】

```
void func(const int* arg){    //引数 arg の値は変更できない。
void func(const int& arg){
```

引数にも「const」を付けることができます。
ポインタやリファレンスを使用した「参照渡し」の引数でも、値を変更することを禁止にできます。