8. 変数の種類と有効範囲

1. 変数の種類(グローバル変数<global variable>とローカル変数<local variable>)

- グローバル変数:関数の外で使用宣言し、全ての関数で利用できる変数
- ローカル変数:関数の中で使用宣言し、宣言した関数だけで利用できる変数

```
行番号
1  #include <stdio.h>
2
3  int a;                    //グローバル変数 a 宣言
4
5  void fnc();               //プロトタイプ宣言
6
7  //メイン関数
8  int main(void){
9
10       a=3;                //グローバル変数 a を宣言&数値 3 を代入
11       fnc();              //関数呼び出し
12       printf("a=%d¥n",a); //グローバル変数 a 表示
13       a=4;                //グローバル変数 a に数値 4 を代入
14
15 }
16
17 //関数 fnc
18 void fnc(){
19
20       a++;                //グルーバル変数 a をカウントアップ+1
21
22 }
```

全関数で有効

＜図 1(1):グローバル変数サンプルプログラム＞

```
行番号
1  #include <stdio.h>
2
3  void fnc();                    //プロトタイプ宣言
4
5  //メイン関数
6  int main(void){
7
8        int a=3;                 //ローカル変数 a(メイン関数)を宣言&数値 3 を代入
9        fnc();                   //関数呼び出し
10       printf("a=%d¥n",a);      //ローカル変数 a(メイン関数)を表示
11       a=4;                     //ローカル変数 a(メイン関数)に数値 4 を代入
12
13 }
14
15 //関数 fnc
16 void fnc(){
17
18       int a=0;                 //ローカル変数 a(関数 fnc)を宣言&数値 0 を代入
19       a++;                     //ローカル変数 a(関数 fnc)をカウントアップ+1
20
21 }
```

有効範囲が異なる

＜図 1(2):ローカル変数サンプルプログラム＞

2. 変数の有効範囲

① 変数を使用宣言することで、値を記憶するための領域がメモリ内に準備される＜メモリの確保＞
② 変数に値を保存したり、演算や出力したりして利用する
③ 最後に記憶領域が廃棄（解放）されることで、この領域が別の用途に使われる＜メモリの解放＞

- グローバル変数の場合

プログラム本体の処理が始まる前に、1 度だけメモリが確保される

⇩

プログラム終了時にメモリが解放される

- ローカル変数の場合

関数内で変数を使用宣言したときに、変数の記憶領域がメモリ内に準備される

⇩

関数が終了するときに、記憶領域が破棄（解放）されて、メモリが別の用途に使われる