[第4章]

開発に使うためのC言語を詳細に解説

C言語の文法と記述法

FED社のWIZ-CはANSI（日本のJISにあたる米国規格協会）Cに準拠していますが，マイコン・チップ用の小規模コンパイラということもあり，実装されていないライブラリ関数もあります．この章では，C言語の文法的なことをWIZ-C固有の事柄も含めて説明します．初めての方でも具体手な使用法がわかるように，例を挙げながら説明します．

なお，WIZ-Cスタンダード版は浮動小数点が使えませんが，制御ソフトでは整数演算だけで済ませられることが多いので，さほど不自由はないでしょう．そのため，浮動小数点（float，double）に関する説明は省略します．

● はじめに

本書ではソース・リスト（プログラムが書かれたテキスト・ファイル）を読みやすくするため，また，コーディング時のミスを減らすために，ちょっとした規則を決めておきます．内容はAppendix Cに記述しておきますので，C言語の説明を一通り読み終えた後や，実際にソース・ファイルを見る前などに参照してください．

また，C言語のコーディング（プログラムを記述すること）でありがちな間違いの事例をいくつかAppendix Dに挙げています．

4-1　C言語の説明

● 数値

C言語ではプログラム中に10進数，8進数，16進数の整数値を用いることができます．10進数は普通に0～9までの数字で表現しますが，数値の頭に0を付けると10進数ではなく，8進数とみなされるようになります．8進数で表現できる数字は0～7までですから，08と書くとコンパイル・エラーになります．16進数を表すには頭に0xを付けます．

16進数で表現できる数は0～15ですが，10～15は1文字で表現できる数字がないので，アルファベットのA～Fを順に割り当てます．たとえば，10進数の「12」は16進数では「C」（小文字も可）ということになるので，0xC，0x0Cなどと書くことができます．「0x」の後の0は無視されますが，桁数をそろえる場合

に0を入れてもかまいません．

```
【例】  a = 12;        /* 10進数の数値 */
        b = 034;       // 8進数の数値
        c = 098;       // (誤)9，8は8進数の数値ではないため，コンパイル・エラーとなる
        d = 0xA0;      // 16進数の数値
```

● コメント

「/*」と「*/」(スラッシュとアスタリスク)でかこまれた範囲はコメントとして扱われ，人が読む以外はCコンパイラにとっては何の意味もありません(読み飛ばされる)．「/*」と「*/」は複数行に分かれていてもかまいません．その場合は「/*」と「*/」の行の間にある行もコメントとして扱われます．また，「//」 もコメントを表しますが，これは 「//」 が現れた位置からその行の終わりまでがコメントとなります．複数の行にはまたがりません．

「/*」と「*/」は括弧のようにネスト(入れ子)させることができるコンパイラもありますが，紛らわしいのであまり使いません．

● 文(ステートメント)

変数の宣言や，変数への代入，条件判断などプログラムを構成する命令の単位を文(ステートメント)と言います．この文の文末には「;」(セミコロン)が必要です．コンパイラは行で文を判断しているのではなく，文末のセミコロンを区切りとして文を解釈します．したがって，セミコロンで区切れば，同じ行に複数の文を書くことも問題ありません．ただし，見づらくなる場合もありますので，普通は1行に1文とします．

```
【例】  int a;                      // 1行に一つの文 (変数宣言文)
        int b; int c;               // 1行に二つの文(変数宣言文)
        a = 2;                      // (実行文)
        a++; b = 3; c = a + b;      // 1行に三つの文(実行文)
```

● プリプロセッサ

プリプロセッサとは，コンパイルが始まる前のソース・ファイルに前処理を行うもののことで，ソース・ファイルに指示を記述することで，コンパイルの開始時に自動的に処理されます．

▶ #define

置換マクロと呼ばれるもので，簡単なものでは数値や文字列などのリテラル(定数)をシンボルとして定義し，プログラム上でそのシンボルが現れたところで，定義元のリテラルなどが復元されます．宣言は次のようになります．

```
#define シンボル名 リテラル
```

具体例を示します．

```
【例】  #define GAIN 100      // GAINというシンボルを100に置換するという宣言
        int a;               // 整数型の変数aを定義
        a = GAIN * 2;        //  a = 100 * 2; と展開される(GAINが100に置換される)
```

関数のように使うこともできます．

このアイコンは，章末に用語解説があります

【例】
```
#define ADD(x, y) x+y // 関数のように定義されたマクロ
int a = 1, b = 2, c; // 整数型の変数 a，b，cを定義(a，bは初期値付き)
c = ADD(a, b);       // c = a + bと展開される．cの値は3
```

▶#include

ヘッダ・ファイル(インクルード・ファイル)を呼び出すことを宣言します．ヘッダ・ファイルとは#defineなどの定義や関数のプロトタイプ宣言(後述)，外部変数のextern宣言(後述)などを記述して，別ファイルとし用意したものです．通常，拡張子は".h"とします．

```
#include "ヘッダ・ファイル名"
#include <ヘッダ・ファイル名>
```

具体例を示します．

【例】
```
#include "Def.h"              // Def.hというインクルード・ファイルをこの位置に読み込む
#include <displays.h> // displays.hというインクルード・ファイルをこの位置に読
                      // み込む
```

この宣言が書かれた位置でDef.hまたはdisplays.hファイルの内容が展開されますが，ソース・ファイルが変更されるわけではありません．展開結果はコンパイルの際に生成される，拡張子がASMやLSTのファイルに出力されます．

宣言時の< >と" "の違いは，ヘッダ・ファイルを検索するファイル・パスの違いです．< >の場合は，あらかじめ決められたインクルード用フォルダから検索されます．" "の場合は，カレント・フォルダが検索されます．絶対パスで指定する場合も" "を使います．

ヘッダ・ファイルは，ソース・ファイルを整理して見やすくしたり，複数のソース・ファイルで定義などを共用する目的で使用します．

● 変数，関数の型(データ・タイプ)

変数には符号付き，符号なしの種類があり，さらにデータ長を表す型(データ・タイプ)が定義されています．関数の型とは関数が返す値の型のことです．変数型ごとのビット長はコンパイラによって違う場合があるので，ほかのコンパイラ用に作成されたプログラムを参照するときには注意が必要です．WIZ-Cの場合の各変数のビット・サイズは**表4-1**のようになります．

> CCS-Cではshort型(short　int)は1ビットの変数(ANSI非準拠)，int型は8ビットの変数，long型は16ビットの変数を表すため，プログラムを移植する際などには注意してください．

符号の有無でsigned，unsignedを付けます．省略した場合はコンパイラや，コンパイル・オプションによって違うことがあるため，明確にしたい場合やはっきりしない場合は明記するのが確実です．

符号付きにすると，符号情報として1ビット使われるため，同じ型でも表現できる数値の範囲が変わります．たとえば，8ビットのchar型の場合に表現できる数値の範囲は，

signed char → －128～127
unsigned char → 0～255

となります．

論理演算で使用する場合やPICレジスタの一時記憶などに使用する変数には，BYTE型などのunsigned型にします．PICのレジスタはすべてBYTE型(unsigned char型)で扱います．

表4-1　型(データ・タイプ)一覧
WIZ-Cで使用できるデータ・タイプの一覧．ANSI準拠，非準拠のもの，WIZ-Cで定義されているもの，本書で独自に定義したものがある．本書独自のものは，インクルード・ファイル“Def.h”で定義してあるので，使用する際は，このファイルをインクルードする必要がある．
bit型は関数の引数には使用できない，配列を定義できないなどの制限があるので注意が必要．

型(データ・タイプ)	例	ビット数	最小値	最大値
bit(1)	bit x;	1	0	1
(signed) char	char x;	8	-128	127
unsigned char	unsigned char x;	8	0	255
(signed) int	int x;	16	-32768	32767
unsigned int	unsigned int x;	16	0	65535
long	long x;	32	-2147482648	2147483647
unsigned long	unsigned long x;	32	0	4294967295
short	short x;	16	-32767	32768
unsigned short	unsigned short x;	16	0	65535
BYTE (unsigned char)	BYTE x;	8	0	255
FileReg (unsigned char)	FileReg x;	8	0	255
BOOL(2) (signed char)	BOOL x;	8	0(false)	非0(true)
WORD (unsigned int)	WORD x;	16	0	65535
DWORD(2) (unsigned long)	DWORD x;	32	0	4294967295
ulong (unsigned long)	ulong x;	32	0	4294967295

(注1)：ANSI非準拠
(注2)：本書で独自に定義した型．"Def.h"で定義

● 変数の定義

C言語では，変数を使用する場合は，前もって必ず宣言しておかなくてはなりません．変数名には演算子などの記号は使えませんが，「_」(アンダ・スコア)は使えます．また，1文字目に数字を使うことはできません．通常，大文字小文字は区別されますが，設定により，区別なしにできる場合もあります．それから，必ず実行文(代入や条件判断文など)の前に宣言していなくてはなりません．宣言の形は，

```
変数型 変数名;
```

となります．たとえば，次のようになります．

【例】
```
int a;          // int型のaという名前の変数を宣言
char a2;        // char型のa2という名前の変数を宣言
```

宣言時に初期値をもたせることもできます．次のようにします．

【例】
```
int a3 = 10;  // 宣言時に初期値10を設定
```

同じ型の変数は一つの宣言に変数名をカンマで並べて宣言できます．

【例】
```
int b, c, d;          // int型の変数a，b，cを定義
int e, f = 12, g;     // fのみ初期値付きで定義
```

● 変数のスコープ

変数の種類により適用範囲(その変数が有効なプログラム上の範囲)が変わります．この範囲のことをスコープといいます．**図4-1**に変数のスコープをまとめて説明した図を示します．

▸auto変数

自動変数とも呼ばれます．通常，宣言時にautoは省略します．宣言時にスタック領域に作られます．関数の{ }の中で定義され，関数がコールされるたびに生成され，関数を抜けると破棄されるため，関数内で使用する一時的な変数ということになります．なお，この変数は宣言された時点ではどんな値が入っているかわからないため，必ず，初期化や代入した後に参照してください．

```
【例】  char a;        // 関数の中で宣言
        int b = 50;    // 初期値付きの宣言
```

▸外部(グローバル)変数

mainを含む全関数の外側で宣言された変数です．この変数は同一のソース・ファイル内であれば，すべての関数の中から使うことができます(定義されているファイル内でのグローバル変数)．

また，複数のソース・ファイルがある場合は，参照する側のソース・ファイルでextern宣言を使って外部変数を使用することを宣言すれば，extern宣言のあるすべてのソース・ファイルから一つの共通の変数として参照できます．

```
【例】  int MaxVal;                    // すべての関数の外側で宣言
        int MinVal = 0;                // 初期値付きの宣言
        extern int GlobalVal; // ほかのソース・ファイルにある外部変数をこのソース・ファイ
                                       // ル内でも使用
```

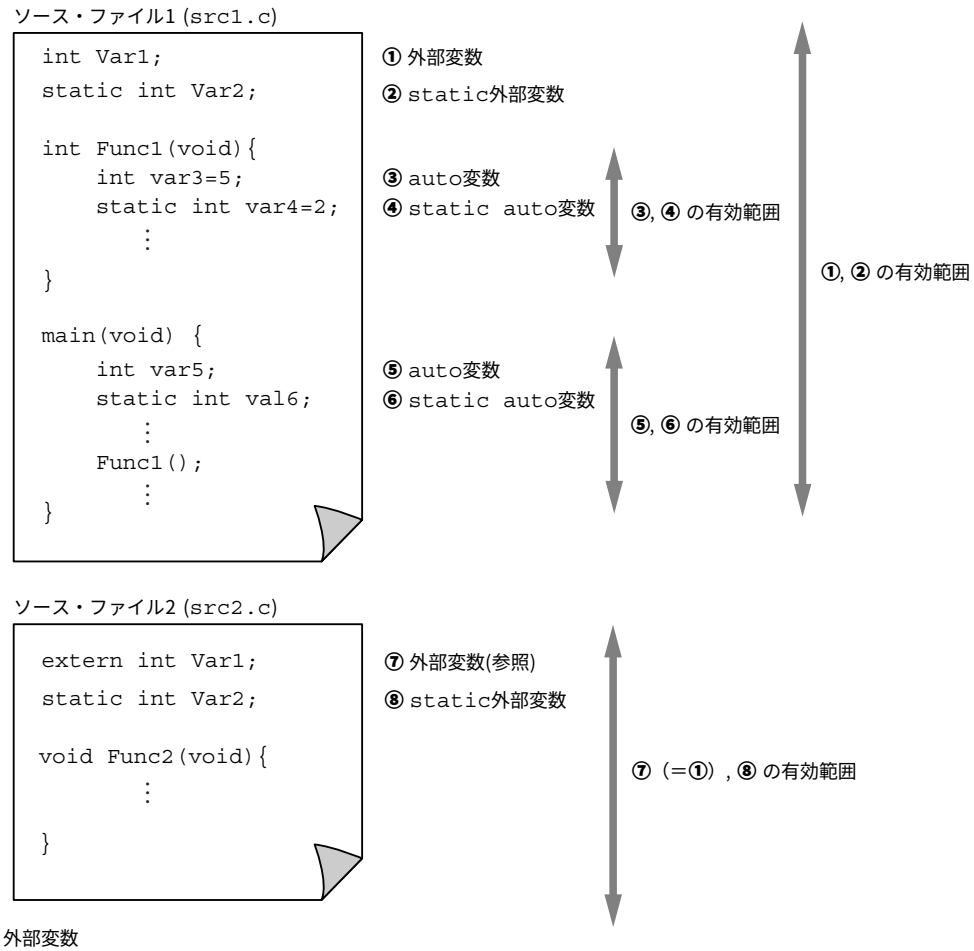
▸static変数

静的(スタティック)変数とも呼ばれ，関数の{ }の中で定義されている場合でもauto変数と違って関数を抜けても破棄されずに永久に保持されます(値も保持されたまま)．そのため，次回同じ関数がコールされた場合，前回の値を継続して使うことができます．ただし，このstatic変数は，関数の外から直接参照することはできません．

一般的なCコンパイラでは，static変数は初期値を定義すると宣言の部分で最初の一度だけ初期値が有効になります．関数の中で初期値付きのstatic変数が定義されているときは，その関数がはじめてコールされたときだけ初期値が代入されていますが，2回目以降のコールでは前回の値を保持しているだけで，初期値の代入は起こりません．また，初期値なしのstatic変数宣言の場合は，暗黙のうちに0に初期化されます．

```
【例】  static int NextVal;          // 外部変数の場合は，すべての関数の外側で宣言(0で初期化)
        static int PrevVal = -1;    // 初期値付きの宣言
```

FED WIZ-C Ver.10では関数内で初期値付きstatic変数を定義すると，関数がコールされるたびに初期値の代入が起こります．これは，一般的なCコンパイラとは違っています．したがって，ほかのコンパイラで使用する感覚では初期値付きstatic変数は使えません．ただし，初期値なしのstatic変数は正常に最初のコール時だけ0に初期化されているため問題ありません．開発元のFED社のコメントによると，これはバグではなく，仕様ということです．



① 外部変数
この変数はソース・ファイル1 内の全域で有効．ほかのソース・ファイルからでも，参照する側のソース・ファイル内でextern宣言をすれば使用可能．
② static外部変数
この変数はソース・ファイル1内の全域で有効．ほかのソース・ファイルからは参照できない．そのため，ほかのソースで同一名のstatic外部変数があっても別の変数とみなされる(関数名に関しても同様)．
③ 初期値付きauto変数
この関数がコールされるたびに初期値が代入される．関数を抜けると破棄される．
④ 初期値付きstatic auto変数
この関数が一番初めにコールされたときは初期値で初期化されている．2回目以降のコールでは初期値は代入されないが，前回コールされたときの値が保持されている．static変数は関数を抜けても破棄されない．

注意 Wiz-Cの場合は，コールされるたびに初期化されるので注意．

⑤ auto変数
変数が宣言された時点では変数の中には何が入っているかわからない．関数を抜けると破棄される．
⑥ 初期値なしstatic auto変数
関数が一番初めにコールされたときには‘0’に初期化されている．2回目以降のコールでは初期化されないが，前回コールされたときの値が保持されている．static変数は関数を抜けても破棄されない．
⑦ 外部参照の外部変数
⑦ はソース・ファイル1 の **①** を参照することを宣言している．つまり ① と ⑦ は同じ変数．
⑧ static外部変数
ソース・ファイル1の **②** と同名であるが，**⑧** と **②** はまったく別の変数．

図4-1　変数のスコープ(有効範囲)