switch case 文

複数方向への分岐を行う制御文として、switch case 文がある。書式は

```
  switch (式) {
    case 式 1: 文 1
               break;
    case 式 2: 文 2
               break;
     …
    case  式 n: 文 n
               break;
    default: 文
 }
```

のようになり、switch の ( ) の式と、case の式 1〜式 n が比較され、一致する case に書かれている文を実行し、break で switch 文を抜ける。break が書いてないと、次の case に書かれている文に実行が進んでしまうので、注意!
また、一致する case がない時は、default が実行される (default は省略可)。

[例題] 1 文字入力し、それが ‘c’ または ‘C’ なら “C”、‘j’ または ‘J’ なら “Java”、‘r’ または ‘R’ なら “Ruby”、これら以外なら “????” と表示しなさい。

```
#include <stdio.h>
main()
{
   char c;

   c = getchar();
   switch(c) {
     case 'c':
     case 'C':  printf("C\n");
                break;
     case 'j':
     case 'J':  printf("Java\n");
                break;
     case 'r':
     case 'R':  printf("Ruby\n");
                break;
     default:   printf("????\n");
  }
}
```

ファイルの入出力

これまではキーボードから入力し、画面上に出力するプログラムばかりであったが、ファイルへの入出力を見てみよう。

ファイルへの入出力はファイルポインタを指定して行う。ファイルポインタは、バッファの位置、バッファ内の現在の文字位置、ファイルを読むのか書くのか、エラーあるいはファイル終端の検知といったファイルに関する情報を含む構造体を指す。ファイルポインタを使っての入出力の手順は以下のようになる。

1. fopen 関数でファイルを開き、ファイルポインタを確保する。
   もしファイルが開けない時は、そこで実行を終了した方がよい。
2. ファイルポインタを使用して、データの入出力を行う。
3. fclose 関数でファイルを閉じる。

以下に例を示す。

ファイルからの入力の例

```
#include <stdio.h>
main()
{
  FILE *fp;          /* FILE は<stdio.h>で定義されている */
  int c;

  if((fp=fopen("e:\\xyz.txt","r")) == NULL) {  /* "r" … read permission */
     printf("Can't open file\n");
     exit(1);        /* 強制的に終了 */
  }

  for(c=getc(fp); c!=EOF; c=getc(fp))              /* EOF … end of file */
     putchar(c);    /* ファイルが終了するまで 1 文字ずつ標準出力へ書き出す */

  fclose(fp);
}
```

ファイルへの出力の例

```
#include <stdio.h>
main()
{
  FILE *fp;

  if((fp=fopen("e:\\pqr.txt","w")) == NULL) { /* "w" … write permission */
     printf("Can't open file\n");
     exit(1);               /* 強制的に終了 */
  }

  fprintf(fp,"this is a pen\n");

  fclose(fp);
}
```

fopen 関数の第 1 引数はファイル名、第 2 引数はそのファイルを取り扱うモードを指定する。モードとしては、"r"(読込み)、"w"(書出し)、"a"(追加) の 3 種類がある。

(注意)
fopen 関数の引数のファイル名中の '\\' は'¥ ¥' のことである。

以下に主な入出力関数をまとめてみる。

関数名	機能
getchar	標準入力から 1 文字を読む関数 (マクロ)
putchar	標準出力へ 1 文字を出力する関数 (マクロ)
printf	標準出力ファイルに形式付きで出力する関数
getc	高水準ファイルから 1 文字を読む関数
putc	高水準ファイルへ 1 文字を出力する関数
ungetc	高水準ファイルへ 1 文字を戻す関数
gets	標準入力から 1 行読み込み、メモリバッファ上に書き込む関数
puts	標準出力へ文字列を出力する関数
fopen	高水準ファイルを開く関数
fclose	高水準ファイルを閉じる関数
fgetc	高水準ファイルから 1 文字を読み込む関数
fputc	高水準ファイルへ 1 文字を出力する関数
fgets	高水準ファイルから 1 行読み込む関数
fputs	高水準ファイルへ文字列を出力する関数
fseek	高水準ファイルの中の新しい位置にポインタを移す関数
scanf	標準入力から書式付きでデータを受けとる関数
fscanf	高水準ファイルから書式に従って入力する関数
fprintf	高水準ファイルへ書式に従って出力する関数
sscanf	メモリバッファから書式に従ってデータを入力する関数
sprintf	出力バッファに書式に従ってデータを出力する関数
ferror	高水準ファイルのエラーを調べる関数
perror	stderr にメッセージを出力する関数
clearerr	外部変数 errno を 0 にする関数
creat	低水準ファイルを作る関数
open	低水準ファイルを開く関数
close	低水準ファイルを閉じる関数
read	低水準ファイルから 1 ブロックを読みとる関数
write	低水準ファイルへ 1 ブロックを書き込む関数
lseek	低水準でオープンされたファイルの中の位置を動かす関数
unlink	ファイルシステムから 1 個のファイルを消去する関数
eof	低水準ファイルの終りを調べる関数

上表中の各関数の具体的な使い方は、オンラインヘルプやリファレンスマニュアルを見てほしい。
以下に printf、fprintf、sprintf における出力書式指定フィールドの形式をまとめておく。

%[flag][width][.precision][L]type

フィールド	文字	機能
flag	−	左につめて出力する
	+	+ か − の符号を必ずつけて出力する
	0	数字の入らない桁に 0 がつめられる
width	1 から n	出力する最低の幅を、具体的な数で表す
	*	出力する最低の幅を、引数から読みとる
.precision	.	浮動小数点の小数部分を表示することを示す
	1 から n	最大の幅、または小数部分の幅を具体的な数で表す
	*	小数部分の幅を引数から読みとる
L	L	引数が整数であれ浮動小数点であれ、long であることを示す
	l	
type	d	整数型の引数を符号付き 10 進数で出力する
	u	整数型の引数を符号なし 10 進数で出力する
	o	整数型の引数を符号なし 8 進数で出力する
	x	整数型の引数を符号なし 16 進数で出力する
	c	引数を 1 文字として出力する
	s	引数を文字列として、\0 または指定された出力幅まで出力する
	e	引数を指定形式 ([−]m.n…nE[+−]XX) で出力する。小数部分の桁数は指定がなければ 6 である。
	f	引数を実数形式 ([−]m…m.n…n) で出力する。小数部分の桁数は指定がなければ 6 である。
	g	上記の e 形式か f 形式のうちの短いほうで出力する

データの型

宣言時のデータの型

C 言語での基本的なデータの型はわずかに次のとおりである。

- char
- int
- float
- double

基本データ型の前には次の修飾子を付けることができる。

- int の前には short か long
- float と double の前には long
- char と整数型の前には unsigned

以下に主なデータ型の表せる範囲を示す。

データ型	バイト数	数の範囲
char	1	−128 〜 127
unsigned char	1	0 〜 255
int	2	−32768 〜 32767
	4	−2147483648 〜 2147483647
unsigned int	2	0 〜 65535
	4	0 〜 4294967295
long	4	−2147483648 〜 2147483647
unsigned long	4	0 〜 4294967295
float	4	6 桁の浮動小数点数
double	8	14 桁の浮動小数点数

修飾子または基本データ型の前には、記憶クラスを記してその変数の存在時間を指定することができる。

- auto … 自動変数はスタック上にとられる。すなわち、関数が実行中のときのみ有効となる。記憶クラスを指定しないと auto となる。
- register … 自動変数の一種だが、コンパイラにレジスタに割り当てることを期待するときに用いる。
- static … 永久な記憶領域にとられる。関数実行が終っても値を保持するので、再度関数が呼ばれても前の値のままである。宣言時に static 変数に初期値を代入する場合、コンパイル時に一度だけ行われる。
- global … 関数の外で宣言されるとこのクラスになる。スタック上にはとられない。
- external … external で指定された場合、コンパイラはその変数は他のモジュールで定義されたものとみなす。したがって、その変数に対してはメモリを割り当てない。

さらに、型修飾子として `const` を任意の型の前に付けることができ、その変数の初期化はできるものの、その後の変更はできないことを宣言できる (引数でもらう時に、`const` をつけることによって、その関数内では値を参照するだけで、変更しないことを明示するような場合に使う)。

おわりに

C 言語を完全に把握するには、正確なことが書かれている本を手元に置いて学習を続けてください。たとえば、
・ カーニハン & リッチー著, 石田晴久訳「プログラミング言語 C」, 共立出版
は C 言語のバイブルと言われています。
・ 内田智史編著「C 言語によるプログラミング　基礎編、応用編」, オーム社
も良い本です。