卒 業 研 究 報 告

題　　目

C 言語におけるテキストファイル操作

指 導 教 員

原　央 教授

報 告 者

大成 博音

平成　13 年　2 月 9 日

高知工科大学 電子・光システム工学科

目次

第１章　　前書き

1-1．　プログラミング言語とは

コンピュータにまとまった処理をさせるためには、その処理の手順を分析し、基本的な命令を組み合わせてそれを実現しなければならない。こうした基本的な命令を組み合わせたものをプログラムと呼ぶ。

コンピュータは、動作の順序を記述した機械語のプログラムを解釈して動作する。機械語のプログラムとは0と1が並んだもの(実際には１６進数)である。機械語のプログラムはコンピュータがそのまま理解できるものであるが、人間にはただ0と1が並んでいるようにしか見えず、当然内容は理解できない。

人間とコンピュータのこのような言葉のギャップを埋めるために、プログラミング言語が存在する。プログラミング言語とは、人間にもコンピュータにも理解できるプログラムを記述するための言葉である。だだ、プログラミング言語は，我々が日常使う自然言語ほどに判りやすいわけではない。数学の公式を自然言語では表現しにくいのと同じで，プログラミング言語はコンピュータへの命令を簡潔，正確に伝えることを優先した言語である。

1-2．　プログラミング言語の種類

プログラミング言語には処理目的に応じて様々なものが存在し、C言語、C++言語、FORTRAN、Lisp、Pascal、Java、Purl、tcl/tk など様々なものがあり、HTML も今や簡単なプログラム言語の一種と言われている。プログラミング言語は、国や地域によって分かれているのではなく、その言語が得意とする内容や、発展の歴史などによって分かれている。全ての目的に使用可能で、しかも自分の要求を全て満たすようなプログラミング言語は一般的に言って存在しないので、プログラミングを行う際には、自分の目的に合わせてプログラミング言語を選択する必要がある。

本章では、Windowsに限らず利用者の最も多い標準開発言語ともいえるC言語をとりあげる。

第 2 章　Ｃ言語

2-1.　Ｃの歴史

Ｃ言語は 1972 年、ベル研究所の D.M.リッチー (Dennis M Ritchie) によって設計されたシステム記述言語である。PDP-11 という機種にオペレーティングシステム UNIX を構築する際に使用された言語であり、開発当初は主に UNIX システム上で利用されていたが、現在では UNIX 以外のシステムにおいてもＣ言語は幅広く利用されている。また、Ｃ言語は世界的な二つのプロフェッショナル向けプログラミング言語であるＣ++と Java に通じており、Ｃ++はＣ言語に基づいて作成され、Java はＣ++に基づいて作られた言語である。Ｃ言語はシステム開発言語と呼ばれるが、これはコンピュータのあらゆる細かい部分に関するプログラミングが可能であるという意味である。そして、Ｃ言語をコンパイルした機械語プログラムは実行速度が速く、サイズも小さい。Ｃ言語は多くの OS の開発言語として採用されており、Windows の開発言語もＣ言語である。そして、ワープロや表計算といった Windows のアプリケーション・ソフトウエアの多くもＣ言語によって開発されている。また、比較的新しい言語なので文法が洗練されておりプログラムを簡潔に書くことができる。Ｃ言語は最近のプログラミング言語の基盤であり、高いパフォーマンスと品質を有するソフトウェアを作成するにはＣ言語についての知識が不可欠になっている。

2-2.　Ｃのプログラムの構造

プログラムを作るには、まず人間がコンピュータにどのような手順で動いてもらいたいかを記述する必要があり、広い意味で言えば、それをプログラミングという。そして、Ｃ言語はそのプログラミングに使う言語の一種である。プログラム言語の多くは、人間が分かりやすいように英単語がベースになっている。このような人間が見て分かる状態のものをソースコードと言う。ソースコードをテキストファイルとして作成するため、テキストエディタを利用する。UNIX の場合は VI エディタや Emacs 等が有名で、Windows 場合は標準で付いてくるメモ帳や WordPad などがそうである。コンパイルとは、プログラマーが書いたプログラムをコンピュータが分かる機械語の形式に変換する作業のことである。つまりコンパイルとは、プログラマーの使う言葉から、コンピュータが使う言葉へ翻訳する作業といえる。この翻訳作業をするソフトのことを、コンパイラと言う。

以上をまとめると、Ｃ言語によるプログラミングは、(1) ソースコードをテキストエディタで作成、(2) コンパイラにより、コンピュータが実行できる機械語に変換、(3) コンパイル済み実行形式プログラムの実行、という三つの過程で作業を行う。

2-3. Cプログラムの構成要素

C言語の主な要素は、変数(variable)と関数(function)である。変数はさまざまな値を格納するために確保されたメモリ領域であり、Cでは必要な量の領域を確保しさえすれば理論的にはどんなデータでも扱うことができる。関数はプログラムが持つひとつひとつの機能を表したものであり、プログラム全体のうちどこからどこまでを「ひとつの機能」とするかは、プログラマーの自由である。すべてのCプログラムには、１つ以上の関数が含まれており、それぞれの関数には1つ以上の文(statement)が含まれている。Cのすべての文は、必ずセミコロン『；』で終わる。改行は文の区切りとしては認識されず、文はどの位置に配置しても良い。Cの一般形式を次に示す。

```
戻り値の型　関数名（仮引数）
  {
    文 ；
  }
```

「戻り値の型」にはその関数が返すデータの型(type)を指定し、「関数名」は関数の名前である。その関数に情報を渡すには、「仮引数」(formal parameter)を使用する。

関数はいくつかの例外を除いて好みの名前を付けることが出来る。関数名に使用できるのはアルファベットの大文字と小文字、数字の０～９、そしてアンダースコア『＿』だけである。また、Cでは大文字と小文字が区別される。Cプログラムは複数の関数を含む事ができるが、どのプログラムも必ず main( )関数を含まなければならない。main( )関数はプログラムを実行する開始点となり、main( )関数の『{ }』の中の文を実行する。

Cプログラムでもう一つ重要なのがライブラリ関数(library function)である。ANSI C 標準では、すべてのCコンパイラが実装しなければならないライブラリ関数群が指定されており、作成するプログラムでC標準ライブラリ(C Standard library)を使用する事が出来る。標準ライブラリにはディスク入出力、文字列操作、数値計算など、さまざまな用途の関数が含まれている。プログラムをコンパイルすると、プログラムが使用する各ライブラリ関数に必要なコードがライブラリから自動的に取り出される。BASIC や Pascal など、他のプログラミング言語では、ファイルの書き込みやサイン、コサインの計算などの操作が言語の中に組み込まれたキーワードにより実行される。C言語の利点は、このような関数がライブラリに入っているために、柔軟性が高いという点である。

ライブラリ関数の中で最も良く使われる関数が printf( )関数である。printf( )関数は、非常に多様な使い方ができるが、単純な一般形式を次に示す。

```
printf(“ 出力させる文字列 ”)
```

printf( )関数は 2 つの二重引用符『” ”』の間にある文字を画面に出力する。Cでは二重引用符に囲まれた複数の文字を文字列(string)という。また、printf( )の括弧の中の引用される文字列の事を printf( )への引数(argument)という。関数を呼び出すには、関数の名前を指定して、その後に関数に渡す引数の並びを括弧に入れて記述する。引数を必用としない場合は、引数を指定しなくてよいので、括弧の中は空のままにしておく。引数が２つ以上の場合はカンマ『,』で区切る。

ほとんどのCプログラムに共通するもう１つの構成要素はヘッダファイルである。Cでは、標準ライブラリ関数についての情報がいろいろなファイルに入ってコンパイラに付属している。これらのファイル名には必ず拡張子『 .h』が付く。Cコンパイラはライブラリ関数を扱うために、これらのファイルに入っている情報を利用する。このようなヘッダファイルをプログラムに追加するには、『#include』というプリプロセッサディレクティブ(preprocessor directive)を使用する。すべてのCコンパイラは、コンパイル処理の第一段階でプリプロセッサ(preprocessor)と呼ばれるプログラムを実行するが、これはソースファイルを実際にコンパイルする前準備としてさまざまな処理をほどこす。『#include』ディレクティブは実際にはCという言語の一部ではなく、別のファイルを読んでその内容をプログラムに挿入するようプリプロセッサに指示するディレクティブである。

最も頻繁に使用されるヘッダファイルは、『stdio.h』であり、このヘッダファイルを挿入するには次のように記述する。

```
#include <stdio.h>
```

『stdio.h』ヘッダファイルには他の情報とともに、ライブラリ関数 printf( )に関する情報が入っている。『#include』ディレクティブがセミコロンで終わっていないのは、『#include』ディレクティブがCの文を定義できるキーワードではなく、Cコンパイラに対する直接の指示になっているからである。

画面に文字を表示させるプログラムを[prog１-1]に示す。このプログラムを実行させると画面上には[prog2-2]のように文字が表示される。

```
#include <stdio.h>
int main( )
{
   printf(“Hello!!¥n”);
   return 0;
}
```

[prog 2-1] 文字出力プログラム

```
Hello!!
```

[prog 2-２] プログラム実行結果

この文字出力プログラムは，まずCコンパイラが 1 行目の『#include <stdio.h>』を読み込み、ファイルには『printf( )』に関する情報を伝える。2 行目の『int main( )』から main( )関数が開始される。3～6 行目の{ }の中で関数は構成されており、4 行目で標準ライブラリ関数『printf( )』を呼び出し、printf( )関数は引数に指定された文字列を画面に表示する。5 行目の『return 0;』により、値ゼロが呼び出し側のプロセスに返される。『main( )』から値ゼロが返されると、プログラムが正常に終了したことを意味する。( )内の『¥』の付く命令は画面上で出力位置を調整する命令として広く使われ、これをエスケープ文字と言う。代表的なものを次に示す。

エスケープ文字	意味
¥a	ビープ音
¥b	バックスペース
¥0	ヌル
¥n	改行
¥ t	水平タブ
¥¥	円記号
¥?	クェスチョン マーク

次の[prog 2-3]に２数の和と差を求め表示するプログラムを示す。

```
#include <stdio.h>
  void main()
  {
        int a=2, b=1, c, d;
        printf(“ a=%d b=%d¥n ”,a,b);
        c= a + b;
        d= a - b;
        printf(“ 和=%d  差=%d¥n ”, c, d);
  }
```

[prog 2-3] 算術演算プログラム

計算を実行するときは，通常，変数に数値を代入して（記憶させて）実行する。プログラム中の『a=2』や『b=１』は、それぞれ変数『a』、『b』に 2、１を代入する（記憶させる）という意味である。また，『c=a+b』、『d=a-b』はそれぞれ『a』と『b』の和を変数『c』に、差を変数『d』に代入するという意味である。『＝』のことを代入演算子と呼び、＋や－は算術演算子とよばれる。この『＝』は数学で言う「等号」では

なく、「代入」という意味である。等号の場合は『＝＝』と２つつなげた記号で表わす。算術演算子の代表的なものを次に示す。ここでは main( )関数を『void』で宣言している。int 型関数が『return』で値を返すのに対して、関数が値を返さない場合は戻り値の型は void 型になる。

演算子	演算
＋	和算
－	差算
＊	積算
／	商算
%	剰余

最初に『int a, b, c, d;』と宣言されたこの行の記述は、変数宣言とよばれ、以下で使用する変数の名前と型を宣言する。文中の『int』は、変数『a,b,c,d』が整数値(Integer)を記憶する整数型変数であることを表す。宣言の仕方には、いくつかの決まりがあり、『int　a,b,c;』という様に複数の同じ型の変数を一度に宣言できる。

次に、ここでも printf()関数が使われているが、今度は変数に代入された数値を表示するのに使われている。『printf(“a=%d b=%d￥n”,a,b)』では、『” ”』の中に『％ｄ』を二つ使用しており、これらはフォーマット指定子と呼ばれ、変数『a,b』に対応している。『%d』は整数値を表示することを意味する。代表的なフォーマット指定子を次に示す。

指定子	入力フォーマット
% d	符号付き 10 進整数
% f	１０ 進の浮動小数点数
% c	文字
% s	文字列

Ｃは、他のコンピュータ言語とは異なり、変数をあらかじめ宣言してなければ使用できない。変数の宣言には、Ｃコンパイラに変数の型(type)を伝えるという重要な役割がある。変数名は関数と同じく、アルファベットの大文字と小文字、数字の０～９、そしてアンダースコア『 _ 』を組み合わせて自分で作ることができ、通常はプログラムを読んだとき、その変数が何を記憶する変数であるかすぐわかるように工夫する。ただし，先頭の文字はアルファベットでなければならない。変数を宣言するには次のような一般形式を使用する。

```
型　変数名；
```

ここで、『型』にはC言語のデータ型を、『変数名』には変数の名前を入力する。
C言語は次に示すようないくつかの基本データ型 をサポートしている。

型名	意味
char	文字を記憶する変数を宣言する。
int	整数型変数を意味する。
long	int 型の倍の桁を記憶する整数型変数。
float	実数型変数を宣言する。
double	桁数が float 型の倍の実数を記憶する実数型変数を宣言する。 通常、科学技術的な計算は float より double を使う。

数値を記憶するスペースは次に示す様に、変数の型で決められている。

種類	型名	バイト数
文字	char	1
整数	short	2
	int	2 または 4
	long	4 または 8
浮動小数点	float	4
	double	8

この違いが必要な理由は、コンピュータの物理的な構造に関係している。変数とは数学的に言えば元々数値の入れ物であり、変数にはあらゆる数値や文字を入れることができる。コンピュータにはメモリーという、書きこんだり消去したりすることが可能な部分があり、変数を利用する際は必要なスペースをここから確保して、その変数専用のスペースにしているのである。普段、私達が数学で使う変数は、計算が進むに従って整数や小数、無理数、複素数という様に、中にいれるものが刻々と変化し、中に何をいれるかによって必要な入れ物の大きさが変わってくる。しかし、コンピュータの持つメモリーというスペースは容量が限られているので、少ない資源をたくさんの変数やプログラムで共有しなければならない。そのため、メモリー使用上の無駄を省くために変数型という概念が生まれたのである。変数を使う前にこの変数はどういう数値を扱い、その値の下限（最小値）・上限（最大値）はいくらであるか、ということを宣言しておくことで、無駄な領域を確保してしまうことを防ぐことができる。この宣言を変数宣言といい、変数を使いたい場合は必ず使用前に宣言する必要がある。通常、整数の場合はint を使い、int より int より小さい整数を使うときは『short』を、int より大きい整数を使うときは『long』を使う。

2-4. 算術関数を使った計算

ＣＰＵには、三角関数や対数計算をする機能はなく、これを四則演算で計算するのは面倒である。そこでCでは double 型の引数をとり、double 型の値を返す算術関数が定義されており、これらの関数は次のように分類される。

三角関数

sin( )	sin 関数を求める
cos( )	cos 関数を求める
tan( )	tan 関数を求める

指数関数と対数関数

log( )	自然対数を求める
logn( )	常用対数を求める
exp( )	自然対数の底ｅの x 乗を求める

指数計算

sqrt(x)	x の平方根を求める
pow(x , y)	xの y 乗を計算する

[prog 2-4]に算術関数を使ったプログラムを示す。

```
#include < stdio.h >
#include < math.h >
  void main( )
  {
        double x=3.1415 /4, y;

        y=sin(x);
        printf("角度%f（rad）のとき、sin の値は%f です。", x, y);
  }
```

[prog 2-4] 算術関数を使ったプログラム（2）

算術関数を使用するときは、ヘッダー部に『#include <math.h>』という記述が必要である。多くの算術関数の記述のしかたは、『y = sqrt(x)』 、『y = sin(x)』、『y = log(x)』のようになる。

すなわち、算術関数は『x』 を与え、関数の値を計算させ 『y』 で受け取る。この

とき、『x』を引数、『y』で受け取る関数の計算結果を戻り値という。

また、次の[prog 2- 5]は上記の[prog 2- 4] のプログラムを，少し変更してみた。記述が増えているが，プログラムとしての内容はまったく同じである.

```
#include < stdio.h >
#include < math.h >
#define    PI     3.1415

  void main( )
  {
        double x, y;

        /*三角関数の例 */
        x = PI/4. ;     /*角度はラジアン単位*/
        y=sin(x);
        printf("角度%f（rad）のとき、sin の値は%f です。", x, y);
  }
```

[prog 2-4] 算術関数を使ったプログラム（2）

『/*三角関数の例*/』の部分は，コメントとよばれ，コンパイラーがプログラムをコンパイルするとき，この部分はプログラムの部分とはみなさず，無視される。これはプログラムを作成する人がプログラムを読みやすくするために使う。

もう一つのの追加行は,『#define PI 3.1415』である。これはプログラム中のＰＩという文字を 3.1415 と定義すると言う意味である。このように定義された文字ＰＩを記号定数といい、普通記号定数は，目立つように大文字だけで記述する。プログラム中にＰＩが現れると，コンパイラーは，それを 3.1415 に置き換えてからコンパイルする。

以上が簡単ではあるが、Ｃ言語についての説明である。

第 3 章　　プログラム解説

作成したプログラムは以下の二つである。

- 電話帳プログラム
- ファイル操作プログラム

電話帳プログラムについては、コマンドラインでプログラムを実行し、プログラム中で指定してある一つのファイルにデータの書き込みを行う。

ファイル操作プログラムについては、電話帳プログラムと違い、コマンドラインでプログラム名とファイル名を選択する。つまり、コマンドライン引数としてファイルを指定する。ファイルは画面上で自ら新規に作成でき、また、すでにあるファイルを選択することもできる。

プログラムの解説の前にファイル入出力とコマンドライン引数について簡単に説明する。

3-1．　ファイル入出力

C言語では、ファイル入出力を実行するキーワードは何も定義されていないが、Cの標準ライブラリには実に豊富なファイル入出力関数が含まれている。また、ファイルは画面出力とは異なり、出力結果の保存が可能である。

ファイルの入力は、簡単に手順だけを説明すると以下の(1)から(3)の様になる。

(1)　ファイルを開く
(2)　データ入力
(3)　ファイルを閉じる

3-2．　ファイルシステムの基本

ファイルからデータを読み込むには入力用に、そのファイルを開く。ファイルを開くためには関数『fopen( )』を使う。fopen( )関数の一般形式は次のように記述する。

```
FILE *fopen( char *filename, char *mode);
```

fopen( )関数は FILE 型(ヘッダファイル『stdio .h』の中で定義されている型)のポインタを返す。 そこで、まず fopen( )関数が返す値を保存するため、FILE 型のポインタ『fp』を用意する。次に fopen( )関数を使ってファイルを以下のように開く。

```
fp = fopen( " test "," r");
```

第一引数には、入力するファイル名を指定する。第二引数にはオープンモードを指定する。オープンモードとは、どのような目的でそのファイルを開くかを指定するものである。オープンモードに指定できる有効な値を次に示す。

モード	動作	ファイルあり	ファイルなし
r	読み込み専用	正常	エラー(NULL 返却)
r+	読み込み + 書き込み	正常	エラー(NULL 返却)
w	書き込み専用	サイズを 0 にする	新規作成
w+	読み込み + 書き込み	サイズを 0 にする	新規作成
a	上書き書き込み専用	最後に追加する	新規作成
a+	読み込み + 追加書き込み	最後に追加する	新規作成

『r』、『w』、『a』はそれぞれ「読み」、「書き」、「追加」の専用モードである。つまり、『r』モードで fopen したファイルに書き込み動作を行うことはできない。『r+』と『w+』は一見同じ「読み書き」可能に見るが、先に来るモードが主となるため、ファイルの有無で動作が異なる。また、『a+』の書き込みは「追加書き込み」しかできない。

ファイルを閉じるためには関数『fclose( )』を使う。fclose( )関数の一般形式は次のように記述する。

```
int fclose( FILE *fp ) ;
```

fclose( )関数の引数には、ファイルを開いたときに受け取ったファイルポインタを指定する。fclose( )関数はファイルを閉じることに成功すると 0 を返し、失敗すると- 1 を返す。次の[prog 5-3]にファイルシステム関数を使ったプログラムを示す。

```
#include< stdio.h >
#include< stdlib.h >
void main( )
{
  char str[80] = "これはテストです。¥n";
  FILE *fp;
   char *p;

   if(!(fp = fopen("test","w"))) //書き込み用に test を開く
   {printf("ファイルを開くことができません¥n");exit(1);}
  p=str;  // str を書き込む
   while(*p)
   {
    if(fputc(*p,fp)==-1)
      {printf("ファイル書き込みエラー¥n");exit(1);}
      p++;
   }
   fclose(fp);
}
```

[prog 3-1] ファイルシステム関数を使ったプログラム

このプログラムではまず、出力のために『test』という名前のファイルを開く。次に「これはテストです。」という文字列をファイルに書き込み、ファイルを閉じる。実行すれば『test』ファイルの中に文字列が書き込まれている。

3-3. ファイル入出力関数

ファイル入出力に用いられる主な関数を以下にまとめた。

（1）fopen

[書式]

FILE * fopen(char *ファイル名,char *モード);

[機能]

- ファイル名で示されるファイルを、指定モードで開く。
- ファイルが正しく開くと、ファイルポインタが返される。
- エラーの時は NULL（0）が返される。

（2）fclose

[書式]

int fclose(FILE *ファイルポインタ);

[機能]

・ fopen で開いたファイルポインタで示されるファイルを閉じる。エラーの時は、EOF(-1)が返される。

[用例]

fclose(ファイルポインタ) ;

（3）fgets

[書式]

char *fgets(char *文字列, int 数値, FILE *ファイルポインタ);

[機能]

・ ファイルポインタで指定したファイルからの１行入力。1 行の「最大文字数を引数で指定する必要がある。この文字数には'￥0'も含まれるので、実際に入力できる文字数は「最大文字数-1」となる。
・ 入力が終了したら NULL(0)を返す
・ 指定するファイルは『r』を含むモードで fopen する必要がある。

[用例]

```
char str[80];
fgets( str ,80,fp);
```

（4）fputs

[書式]

int fputs(char *文字列,FILE *ファイルポインタ);

[機能]

・ ファイルポインタで指定したファイルへの１行出力。
・ 指定するファイルは『w』か『a』を含むモードで fopen する必要がある。

[用例]

fputs("abcdef",fp) ;

（5）fgetc

[書式]

int fgetc(FILE *ファイルポインタ);

[機能]

・ ファイルポインタで指定したファイルからの１文字入力。
・ 入力が終了したら EOF を返す。
・ 指定するファイルは『r』を含むモードで fopen する必要がある。

[用例]

```
c = getc(fp);
```

（6）fputc

[書式]

```
int putc( int 出力文字, FILE *ファイルポインタ );
```

[機能]

- ファイルポインタで指定したファイルへの１文字出力。
- 指定するファイルは『w』か『a』を含むモードで fopen する必要がある。

[用例]

```
putc('a',fp);
```

（7）fscanf

[書式]

```
int fscanf( FILE *ファイルポインタ,char *制御文字列,アドレス);
```

[機能]

- ファイルポインタで指定したファイルから書式つきで入力する。scanf のファイル版。ファイルポインタを指定する以外は scanf と同じ。
- 指定するファイルは『r』を含むモードで fopen する必要がある。

[用例]

```
fscanf(fp,"%d",&dt);
```

（8）fprintf

[書式]

```
int( FILE *ファイルポインタ,char *制御文字列,変数)
```

[機能]

- ファイルポインタで指定したファイルへ書式つきで出力する。printf のファイル版。ファイルポインタを指定する以外は printf と同じ。
- 指定するファイルは『w』か『a』を含むモードで fopen する必要がある。

[用例]

```
fprintf(fp,"%8d￥n",dt);
```

3-4. コマンドライン引数

プログラムの解説の前にコマンドライン引数について簡単に説明する。

多くのプログラムはコマンドラインで引数(command-line argument)を渡すことができる。コマンドライン引数とは、オペレーティングシステムのコマンドラインでプログラムを起動するためにプログラム名を指定する際、その後ろに追加する情報である。コマンドラインで引数は情報をプログラムに渡す役目を持っている。ほとんどの

プログラムは、main( )関数に引数を使う必用はないが、main( )関数は必要なら三つまでの引数を指定することができ、コマンドライン引数を取得するにはそのうちの最初の二つを使う。この仮引数は一般に『argc』、『argv』と呼ばれ、これらの仮引数はオプションなので、コマンドライン引数を使わない場合には存在しない。argc 仮引数は、コマンドライン上の引数の数を保持する変数である。プログラムの名前が最初の引数として認識されるため、argc 仮引数は最低でも 1 となる。argv 仮引数は文字列へのポインタの配列である。argv を宣言するもっとも一般的な型は『char *argv[ ];』と記述する。空の[ ]内は、長さが決っていない配列であることを示している。コマンドライン引数はすべての文字列として main( )関数に渡され、個々の文字にアクセスするには argv に添え字指定する。例えば、『argv[0]』はプログラム名を、『argv[1]』は最初の引数を指定する。次の[prog 5-1]に簡単なコマンドライン引数を使ったプログラムを示す。

```
#include< stdio.h >
int main( int argc, char *argv[ ] )
{
  for(int a=0;a<=3;a++)
  printf( " %s",argv[a] );
  return 0;
}
```

[prog 3-2] コマンドライン引数を使ったプログラム

これは配列『argv』の最初の 4 つの要素を表示するプログラムである。 プログラム名を『com』とし、コマンドラインに引数として『a』,『b』,『c』,『d』,を指定する。このプログラムを実行すると次の[prog 5-2]の様に表示される。

```
A>com a b c d
com a b c
```

[prog 3--3] プログラム実行結果

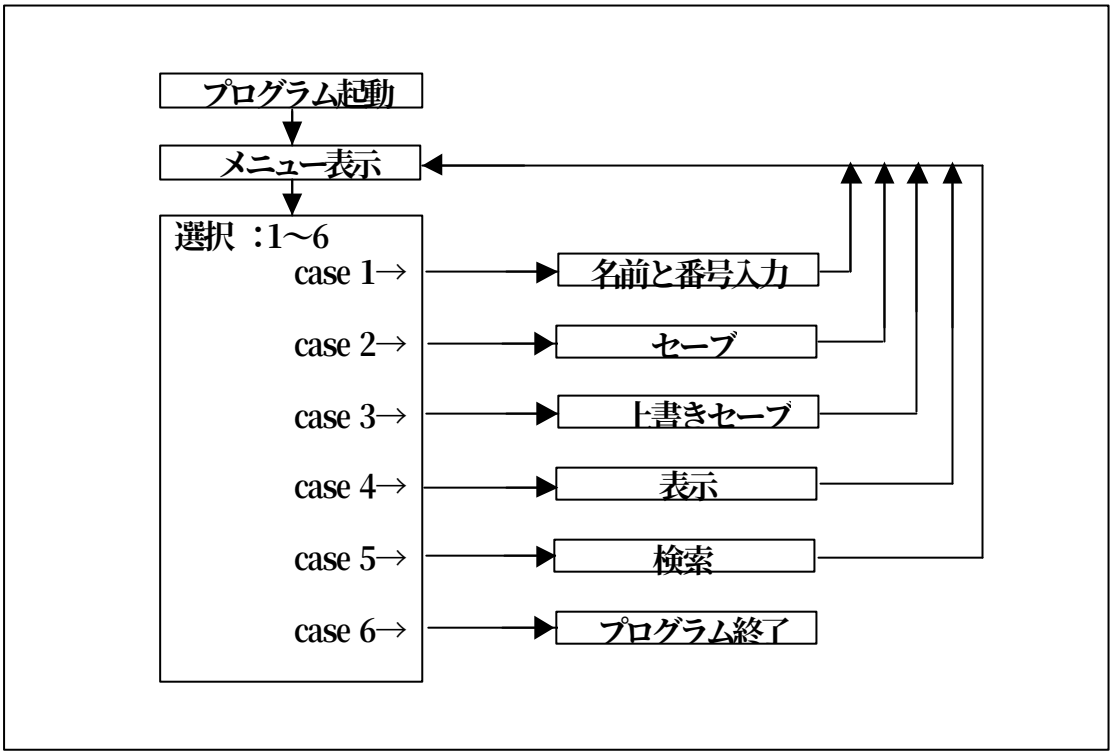
実行結果は、コマンドライン引数が表示されているが、引数の『d』が表示されていない。つまり『argv[0]』には第一引数ではなく、プログラム名が入るということがわかる。

以上が、簡単なファイル入出力とコマンドライン引数についての説明である。

3-5.電話帳プログラム解説

3-5-1. プログラムの処理内容

まず、この電話帳プログラムの簡単なフローチャートを次に示す。



プログラムの処理内容として、以下の五つが実行できるプログラムを作成した。

(1) 名前と番号の入力
(2) セーブ
(3) 上書きセーブ
(4) 表示
(5) 文字検索

(1) 名前と番号の入力

名前と番号を入力する。(最大 100 人まで)

(2) セーブ

入力された名前と番号をファイルに書き込む。

(3) 追加セーブ

入力された名前と番号をファイルに追加して書き込む。

(4) 表示

ファイルに書き込まれた内容を一つずつ読み込み、すべて表示する。

(5) 文字検索

文字を指定し(最大 10 文字)、その文字に該当するものをファイルの中から検索し、表示する。

3-5-2. 電話帳プログラム説明

このプログラムは main( )関数を含め、七つの関数で構成されており、ヘッダーファイルは、『stdio.h』、『string.h』、『strlib.h』の三つを使用している。また、char 型配列『names』、『numbers』、int 型変数『val』を宣言する。このソースファイルを以下に示す。

```
1: #include<string.h>              //for strcmp,strstr
2: #include<stdlib.h>              //for exit
3: #define MAX 100                //MAX100 人
4: char names[MAX][20];            //  MAX100 人、20 文字
5: char numbers[MAX][20];          //  MAX100 人、20 文字
6: int val;                        //  名前と番号の配列番号
7:
8: int menu();                     //  メニュー関数
9: void enter();                   //  名前と番号の入力
10: void search();                 //  ファイル中の文字検索
11: void save();                   //  データセーブ
12: void load();                   //  データロード
13: void append();                 //  上書きデータセーブ
```

main( )関数では int 型変数『choice』を宣言し、menu( )関数で選択された数を代入する。19 行目から 32 行目にかけて switch 文を do-while 文で囲み、case 1 から case 5 により前節の(1)から(5)を menu( )関数で選択させる。また、case 6 の終了を選択するとプログラムは終了する。全て正常に作動した場合、不正な文字が入力された場合はメニュー画面に戻る。main( )関数のソースファイルを以下に示す。

```
15: int main()
16: {
17:   int choice;
18:
19:   do
20:   {
```

```
 21:      choice = menu();        //menu()の返す値を switch()に代入
 22:      switch(choice)
23:       {
24:      case 1: enter();break;
25:       case 2: save();break;
26:      case 3: append();break;
27:      case 4: load();break;
28:      case 5: search();break;
29:      case 6:
30:                printf("プログラムを終了します");break;
31:       }
32:   } while(choice!=6);
33:   return 0;
34: }
```

プログラムを実行すると次のようなメニュー画面が表示される。

```
------【 MENU 】---------
1] 名前と番号の入力
2] セーブ
3] 上書きセーブ
4] 表示
5] 文字検索
6] 終了
-----------------------------------
番号選択 1] ～ 6]？
```

menu( )関数では int 型変数『select』を宣言し、49 行目の scanf( )関数で入力されたその値を返す。40 行目から 48 行目にかけてメニュー画面を表示させる命令である。50 行目から 51 行目にかけては、キーボードから 1 から 6 が選択されないとプログラムを終了する命令である。メニュー画面でキーボードから 1 から 6 が選択されないときは次のように表示される。menu( )関数のソースファイルを次に示す。

```
 36: int menu()
37: {
38:     int select;
39:
40:     printf("--【 MENU 】---------¥n");
41:     printf("1] 名前と番号の入力¥n");
42:     printf("2] セーブ¥n");
43:     printf("3] 追加セーブ¥n");
44:     printf("4] 表示¥n");
45:     printf("5] 文字検索¥n");
46:     printf("6] 終了¥n");
47:     printf("----------------------¥n");
48:     printf("番号選択 1] ～ 6]？");
49:     scanf("%d",&select);
50:     if(!(select==1||select==2||select==3||select==4||select==5||select==6))
51:     {printf("番号が不正です");exit(1);}
52:     return select;
53: }
```

メニュー画面で aaa **を選択すると次のように表示される。**

```
------【 MENU 】---------
1] 名前と番号の入力
2] セーブ
3] 上書きセーブ
4] 表示
5] 文字検索
6] 終了
-------------------------------------
番号選択 1] ～ 6]？aaa
番号が不正です
```

enter( )**関数では、**57 **行目から** 68 **行目にかけて名前と番号を入力させる命令である。**

入力できる数は、名前も番号も 100 人までである。63 行目は、名前と番号の入力を終了するときの命令であり、大文字でも小文字でもキーボードから Z を打てば enter( ) 関数が終了し、メニュー画面に戻る。また、67 行目から 68 行目により入力が 100 人を超えた場合もメニュー画面に戻る。enter( )関数のソースファイルを次に示す。

```
 55: void enter()
56: {
57:     for(val=0;val<MAX;val++)
58:     {
59:        printf("名前と電話番号の入力をします。（MAX=100 人）¥n");
60:        printf("終了するときは「 z 」を入力して下さい。¥n");
61:        printf("名前は？");
62:         scanf("%s",names[val]);
63:         if(strcmp(names[val],"z")==0 || strcmp(names[val],"Z")==0) break;
64:        printf("番号は？");
65:         scanf("%s",numbers[val]);
66:     }
67:     if(val>=MAX)
68:     printf("これ以上は入力できません。¥n");
69: }
```

enter( )関数を実行すると、次のように表示される。

```
番号選択 1] ～ 6]？1
名前と番号の入力をします。 (MAX = 100 人)
終了するときは「z」を入力して下さい。
名前は？yamada
番号は？0887-56-xxxx
名前と番号の入力をします。 (MAX = 100 人)
終了するときは「z」を入力して下さい。
名前は？
```

save( )関数では、FILE 型ポインタ『fp』、int 型変数『i』を宣言する。98 行目から 99 行目にかけて、ファイルを開くことができない場合はエラーを表示させる。100 行目から 101 行目にかけては、enter( )関数で入力された名前と番号を fprintf( )関数を使用し、ファイルに書き込み命令である。102 行目は書き込み用に開いたファイルを

閉じる命令である。save( )関数のソースファイルを次に示す。

```
93: void save()
94: {
95:   FILE *fp;
96:   int i;
97:
98:   if(!(fp = fopen("P-file", "w")))
99:   {printf("ファイルを開くことができません¥n");exit(1);}
100:   for(i=0; i<val; i++)
101:   {fprintf(fp, "%-15s %5s¥n",names[i], numbers[i]);}
102:   fclose(fp);
103:   printf("ファイルにセーブしました。¥n");
104: }
```

save( )関数を実行すると、次のように表示される。

```
番号選択 1] ～ 6]？2
ファイルにセーブしました。
```

append( )関数は save( )関数とのプログラムが似ているが、111 行目のオープンモードが、追加書き込みモード『a』であるという点で異なる。append( )関数のソースコードを次に示す。

```
106: void append()
107: {
108:   FILE *fp;
109:   int i;
110:
111:   if(!(fp = fopen("P-file", "a")))
112:   {printf("ファイルを開くことができません¥n");exit(1);}
113:   for(i=0; i<val; i++)
114:   {fprintf(fp, "%-15s %5s¥n",names[i], numbers[i]);}
115:   fclose(fp);
116:   printf("ファイルに追加セーブしました。¥n");
117: }
```

append( )関数を実行すると、次のように表示される。

```
番号選択 1] ～ 6]？3
ファイルに上書きセーブしました。
```

load( )関数では、FILE 型ポインタを宣言する。また、『val』を初期値０と宣言する。『fp』124 行目から 125 行目にかけて、ファイルを開くことができない場合はエラーを表示させる。128 行目から 132 行目にかけては、fscanf( )関数を使用してファイルに書かれた名前と番号を全て読み込み、表示させる命令である。133 行目は読み込みファイルを閉じる命令である。load( )関数のソースファイルを以下に示す。

```
119: void load()
120: {
121:    FILE *fp;
122:    val=0;
123:
124:   if(!(fp = fopen("P-file", "r")))
125:   {printf("ファイルを開くことができません¥n");exit(1);}
126:    printf("名前                番号¥n");
127:    printf("------------------------¥n");
128:   while(fscanf(fp, "%s%s", names[val], numbers[val])!=-1)
129:    {
130:        printf("%-15s%5s¥n",names[val], numbers[val]);
131:        val++;
132:    }
133:   fclose(fp);
134: }
```

load( )関数を実行すると、次のように表示される。

```
番号選択 1] ～ 6]？4
名前                番号
-----------------------------------------
yamada          0887-56-xxxx
tanaka          0887-53-yyyy
kimura          0887-58-zzzz
```

search( )関数では、FILE 型ポインタ『fp』、char 型配列『word』、初期値０の int 型変数『i』を宣言する。また、『a』を初期値０と宣言する。78 行目から 79 行目にかけてファイルを開くことができない場合はエラーを表示させる。80 行目から 82 行目は、検索する文字を scanf( )関数で入力し、読み込ます命令である。83 行目から 87 行目にかけては文字を検索命令で、while 文により fscanf( )関数を使用してファイルに書かれた名前と番号を全て読み込む。次に while 文の{ }内で『strstr( )』関数を使用し、キーボードから打ち込んだ文字と一致する名前と番号を全て表示させ、変数『i』に 1 を代入する。一致するデータが無ければ変数『i』は 0 のままで、89 行目の printf( ) 関数を表示させる。90 行目は読み込み用に開いたファイルを閉じる命令である。search( )関数のソースコードを以下に示す。

```
71: void search()
72: {
73:     FILE *fp;
74:     char word[10];
75:     int i=0;
76:     val=0;
77:
78:     if(!(fp = fopen("P-file", "r")))
79:     {printf("ファイルを開くことができません¥n");exit(1);}
80:     printf("文字を検索します。検索する文字を入力して下さい。¥n");
81:     printf("検索文字？（10 文字まで）＝");
82:     scanf("%s",&word);
83:     while(fscanf(fp, "%s%s", names[val], numbers[val])!=-1)
84:     {
85:             if(strstr(names[val],word) || strstr(numbers[val],word))
86:             {printf("%-10s%5s¥n",names[val],numbers[val]);i=1;}
87:     }
88:     if(i==0)
89:             printf("該当データなし¥n");
90:             fclose(fp);
91: }
```

search( )関数を実行すると、次のように表示される。

```
番号選択 1] ～ 6]？5
文字を検索します。検索する文字を入力して下さい。
検索文字？(10 文字まで)＝yamada
yamada          0887-56-xxxx
```

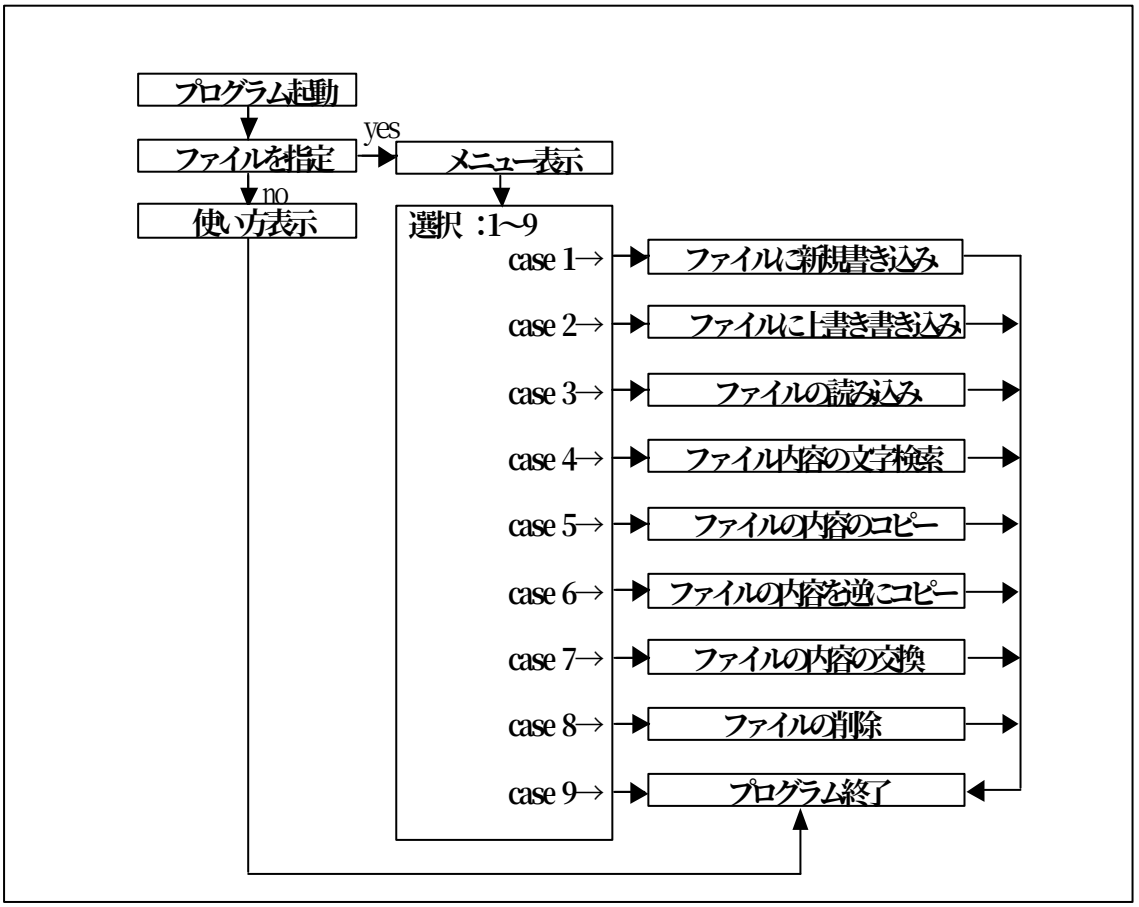
メニュー画面で 6 を選択すると、次のように表示され、プログラムを終了する。

```
番号選択 1] ～ 6]？6
プログラムを終了します。
```

3-6. ファイル操作プログラム解説

3-6-1. プログラムの処理内容

まず、このファイル操作プログラムの簡単なフローチャートを次に示す。



プログラムの処理内容として、以下の八つが実行できるプログラムを作成した。

(1) ファイルの新規書き込み
(2) ファイルの追加書き込み
(3) ファイルの読み込み
(4) ファイルの検索
(5) ファイルのコピー
(6) ファイルの逆コピー
(7) ファイルの交換
(8) ファイルの削除

(1) ファイルの新規書き込み
　コマンドラインに引数としてファイルを指定し、ファイルに内容を書き込む。指定されたファイルが存在する場合は、その内容をゼロにする。
(2) ファイルの追加書き込み
　コマンドラインに引数としてファイルを指定し、ファイルに内容を追加して書き込む。指定されたファイルが存在しない場合は、新規作成する。
(3) ファイルの読み込み
　コマンドラインに引数としてファイルを指定し、ファイルの内容を読み込む。ファイルが存在しないか、読み込めないときはエラーを出す。
(4) ファイルの検索
　コマンドラインに引数としてファイルと文字(1 文字)を指定し、指定した文字を探す。
(5) ファイルのコピー
　コマンドラインに引数として二つのファイルを指定し、別の指定したファイルに内容を複写する。一番目のコピー元ファイルが存在しないか、読み込めないときはエラーを出す。二番目のコピー先ファイルが存在しない場合は、新規作成する。
(6) ファイルの逆コピー
　コマンドラインに引数として二つのファイルを指定し、別の指定したファイルに内容を逆に複写する。一番目のコピー元ファイルが存在しないか、読み込めないときはエラーを出す。二番目のコピー先ファイルが存在しない場合は、新規作成する。
(7) ファイルの交換
　コマンドラインに引数として二つのファイルを指定し、その内容を交換する。二つのファイルが存在しない場合は、エラーを出す。交換はその二つのファイル以外にもう一つ一時的にファイルを使用する。
(8) ファイルの削除
　コマンドラインに引数としてファイルを指定し、そのファイルを削除する。

3-6-2. プログラム説明

このプログラムは main( )関数のみで構成されており、ヘッダーファイルは、『stdio.h』、『strlib.h』『ctype.h』『conio,h』の四つを使用している。また、CTRL_Z を ASCII コードである 26 として定義している。このソースコードを以下に示す。

```
1: #include<string.h>
2: #include<stdlib.h>     //  for exit
3: #include<ctype.h>
4: #include<conio.h>      //  for getche
5: #define CTRL_Z 26      //  CTRL_Z は ASCII コードである 26 として返される
```

まず、8 行目で int 型変数『select』を宣言する。10 行目から 47 行目にかけてはコマンドラインの引数にファイル名を指定していない場合の表示を if 文で囲んでおり、内容はファイルの処理内容の記述の仕方を表示させる命令である。49 行目から 69 行目にかけては、コマンドラインの引数にファイル名、または文字を指定した場合の表示を else 文で囲んでおり、内容はファイルの処理内容を表示し、scanf( )関数を使用して番号を入力する。68 行目から 340 行目にかけて switch 文で囲み、scanf( )関数で入力した番号により case 1 から case 8 で前節の(1)から(8)を選択させる。case 9 で終了を選択するとプログラムは終了する。全て正常に作動した場合、もしくは途中でエラーが発生した場合もプログラムは終了する。また、338 行目の default により不正な文字が入力された場合、1～9 が入力されるまで繰り返される。このソースコードを以下に示す。

```
6:   int main(int ac,char *av[])
7:   {
8:         int select;
9:  /**************【プログラムの内容を示す(ファイル名なし)】**************/
10:        if(ac==1)
11:        {
12:            printf("コマンドライン上でのテキストファイル操作を行います。");
13:            printf("コマンドラインにファイル名または文字を代入して下さい。¥n");
14:            printf("----");
15:            printf("【 MENU 】");
16:            printf("---------------------------------------");
17:            printf("------------------------¥n");
18:            printf("・ ファイルの新規書き込み ：");
19:            printf("[使用方法]：<プログラム名>_<ファイル名>¥n");
20:            printf("  《処理》新規作成するファイルを指定し内容を書き込む。¥n");
```

```
21:         printf("・ ファイルの追加書き込み ：");
22:         printf("[使用方法]：<プログラム名>_<ファイル名>¥n");
23:         printf("  《処理》ファイルを指定し、内容を追加して書き込む。¥n");
24:         printf("・ ファイルの読み込み       ：");
25:         printf("[使用方法]：<プログラム名>_<ファイル名>¥n");
26:         printf("  《処理》ファイルを指定し、内容を読み込む。¥n");
27:         printf("・ ファイル中の文字検索     ：");
28:         printf("[使用方法]：<プログラム名>_<ファイル名>_<文字>¥n");
29:         printf("  《処理》ファイルを指定し、指定した 1 文字を探し出す。¥n");
30:         printf("・ ファイルのコピー         ：");
31:         printf("[使用方法]：<プログラム名>_<コピー元>_<コピー先>¥n");
32:         printf("  《処理》ファイルを指定し、別の指定したファイルに");
33:         printf("内容を複写する。¥n");
34:         printf("・ ファイルの逆コピー       ：");
35:         printf("[使用方法]：<プログラム名>_<コピー元>_<コピー先>¥n");
36:         printf("  《処理》ファイルを指定し、別の指定したファイルに");
37:         printf("内容を逆に複写する。¥n");
38:         printf("・ ファイルの交換           ：");
39:         printf("[使用方法]：<プログラム名>_<ファイル１>_<ファイル 2>¥n");
40:         printf("  《処理》ファイルを指定し、別の指定したファイルと");
41:         printf("内容を交換する。¥n");
42:      printf("・ ファイルの削除           ：");
43:         printf("[使用方法]：<プログラム名>_<ファイル名>¥n");
44:         printf("  《処理》ファイルを指定し、削除する。¥n");
45:         printf("----------------------------------------");
46:         printf("----------------------------------------¥n");
47:     }
48: /**************【プログラムの内容を示す(ファイル名あり)】**************/
49:     else
50:     {
51:         printf("----");
52:         printf("【 MENU 】");
53:         printf("----------------------------------");
54:         printf("------------------------¥n");
55:         printf("1) ファイルの新規書き込み¥n");
56:         printf("2) ファイルの追加書き込み¥n");
57:         printf("3) ファイルの読み込み¥n");
```

```
58:         printf("4) ファイルの検索¥n");
59:         printf("5] ファイルのコピー¥n");
60:         printf("6] ファイルの逆コピー¥n");
61:         printf("7] ファイルの交換¥n");
62:         printf("8] ファイルの削除¥n");
63:         printf("9] 終了¥n");
64:         printf("---------------------------------------");
65:         printf("---------------------------------------¥n");
66:         printf("選択？1] ～ 9]");
67:           scanf("%d",&select);
68:         switch(select)
69:           {
```

プログラムを起動すると、コマンドラインの引数の数によってそれぞれファイルの処理の仕方が違うので、まずプログラムのみ指定された場合と、ファイルを指定された場合とでプログラムを大きく二つに分けた。

コマンドラインでプログラムのみを実行すると次のような画面が表示される。
(ac==1)

```
----【 MENU 】----------------------------------------------------------------------------------------
・ ファイルの新規書き込み ：[使用方法]：<プログラム名>_<ファイル名>
  《処理》新規作成するファイルを指定し、内容を書き込む。
・ ファイルの追加書き込み ：[使用方法]：<プログラム名>_<ファイル名>
  《処理》ファイルを指定し、内容を追加して書き込む。
・ ファイルの読み込み　　：[使用方法]：<プログラム名>_<ファイル名>
  《処理》ファイルを指定し、内容を読み込む。
・ ファイルの検索　　：[使用方法]：<プログラム名>_<ファイル名>_<文字>
  《処理》ファイルを指定し、指定した 1 文字を探し出す。
・ ファイルのコピー　　：[使用方法]：<プログラム名>_<コピー元>_<コピー先>
  《処理》ファイルを指定し、別の指定したファイルに内容を複写する。
・ ファイルの逆コピー　　：[使用方法]：<プログラム名>_<コピー元>_<コピー先>
  《処理》ファイルを指定し、別の指定したファイルに内容を逆に複写する。
・ ファイルの交換　　：[使用方法]：<プログラム名>_<ファイル１>_<ファイル 2>
  《処理》ファイルを指定し、別の指定したファイルと内容を交換する。
・ ファイルの削除　　：[使用方法]：<プログラム名>_<ファイル名>
  《処理》ファイルを指定し、削除する。

-----------------------------------------------------------------------------------------------------
```

また、コマンドライン引数としてファイルを指定した場合は次のようにメニュー画面が表示される。(!(ac==1))

```
----【 MENU 】-------------------------------------------------
1] ファイルの新規書き込み
2] ファイルの追加書き込み
3] ファイルの読み込み
4] ファイルの検索
5] ファイルのコピー
6] ファイルの逆コピー
7] ファイルの交換
8] ファイルの削除
9] 終了
--------------------------------------------------------------------------
選択？1] 〜 9]
```

case 1では、FILE 型ポインタ『fp1』、char 型変数『ch1』を宣言する。75 行目から 77 行目にかけては、引数にファイル名の数を間違って指定した場合にエラーを表示させる。78 行目から 80 行目にかけては、指定したファイルを開くことができない場合のエラーを表示させる。81 行目から 90 行目にかけては、指定したファイルに文字を書く命令である。もし書き込みに失敗した場合、エラーを表示させる。109 行目の『getchar( )』は通常変数に代入して使われるが、ここでは代入はしていない。この場合、入力された文字は捨てられてしまう。110 行目から 116 行目にかけては、ファイルに文字を書き込む命令である。もし読み込みや書き込みに失敗した場合、エラーを表示させる。117 行目から 119 行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。case 1 のソースコードを以下に示す。

```
70: /*****************【case 1：ファイルの新規書き込み 】******************/
71:         case 1:
72:                 FILE *fp1;
73:                 char ch1;
74:
75:                 if(!(ac==2))
76:                 {printf("error01：[使用方法]：<プログラム名>_<ファイル名>");
77:                 exit(1);}
78:                 if(!(fp1=fopen(av[1],"w")))
79:                 {printf("error01：ファイルを開くことができません。");
80:                 exit(1);}
```

```
81:                 printf("ファイルの内容を書き込んで下さい。¥n");
82:                 printf("プログラムを終了するときは「Ctrl + z 」を押します。¥n");
83:                 getchar();          //改行消去
84:                 while((ch1=getche())!=CTRL_Z)
85:                 {
86:                   fputc(ch1,fp1);
87:                    if(ferror(fp1))
88:                   {printf("error01：ファイルに書き込めません¥n");fclose(fp1);
89:                   exit(1);}
90:                 }
91:                 if(fclose(fp1)==-1)
92:                 {printf("error01：ファイルを閉じることが出来ません¥n");
93:                 exit(1);}
94:                 printf("¥n“%s”ファイルに書き込みました。",av[1]);
95:                 break;
```

case 1 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？1
ファイルに内容を書き込んで下さい。
プログラムを終了するときは「CTRL + z」を押します。
Hello!!
ファイルに書き込みました。
```

case 2 は case 1 のプログラムが似ているが、文字の書き込みが新規であるか追加であるかという点で異なる。case 2 のソースコードを以下に示す。

```
96: /*****************【case 2：ファイルの追加書き込み 】*****************/
97:         case 2:
98:                 FILE *fp2;
99:                 char ch2;
100:
101:                if(!(ac==2))
102:                {printf("error02：[使用方法]：<プログラム名>_<ファイル名>¥n");
103:                exit(1);}
```

```
104:            if(!(fp2=fopen(av[1],"a")))
105:            {printf("error02：ファイルを開くことができません¥n");
106:            exit(1);}
107:            printf("ファイルの内容を書き込んで下さい。¥n");
108:            printf("プログラムを終了するときは「Ctrl + z 」を押します。¥n");
109:            getchar();                    //改行消去
110:            while((ch2=getche())!=CTRL_Z)
111:            {
112:              fputc(ch2,fp2);
113:              if(ferror(fp2))
114:              {printf("error02：ファイルに書き込めません！¥n");fclose(fp2);
115:              exit(1);}
116:            }
117:            if(fclose(fp2)==-1)
118:            {printf("error02：ファイルを閉じることが出来ません¥n");
119:            exit(1);}
120:            printf("¥n“%s”ファイルに追加して書き込みました。",av[1]);
121:            break;
```

case 2 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？2
ファイルに内容を書き込んで下さい。
プログラムを終了するときは「CTRL + z」を押します。
Everyone!!
ファイルに上書き書き込みました。
```

case 3 では FILE 型ポインタ『fp3』、char 型変数『ch3』を宣言する。127 行目から 129 行目にかけて、引数にファイル名の数を間違って指定した場合はエラーを表示させる。130 行目から 132 行目にかけて、指定したファイルを開くことができない場合はエラーを表示させる。133 行目から 139 行目にかけては、ファイルに書かれた文字を読み込み、表示させる命令である。140 行目から 142 行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。134 行目から 139 行目にかけて、for 文を使い{ }内の文を無限にループさせる。そして『fgetc(fp3)』により一文字づつ『fp3』ファイルから文字を最後まで読み込み、『putchar(ch3)』を使うことで、ファ

イルに書き込まれた文字を表示させる命令である。140 行目から 142 行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。case 3 のソースコードを以下に示す。

```
122: /*******************【case 3：ファイルの読み込み 】*********************/
123:        case 3:
124:              FILE *fp3;
125:              char ch3;
126:
127:              if(!(ac==2))
128:              {printf("error03：[使用方法]：<プログラム名>_<ファイル名>¥n");
129:              exit(1);}
130:              if(!(fp3=fopen(av[1],"r")))
131:              {printf("error03：ファイルを開くことができません¥n");
132:              exit(1);}
133:              printf("“%s”ファイルを読み込みます。¥n",av[1]);
134:              for(;;)    //無限ループ
135:              {
136:                 ch3=fgetc(fp3);
137:                   if(ch3==-1)break;
138:                 putchar(ch3);
139:              }
140:              if(fclose(fp3)==-1)
141:            {printf("error03：ファイルを閉じることが出来ません。");
142:              exit(1);}
143:              break;
```

case 3 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？3
ファイルを読み込みます。
Hello!! Everyone!!
```

case 4 では、FILE 型ポインタ『fp4』、char 型変数『ch4』を宣言する。149 行目から 151 行目にかけて、引数にファイル名の数を間違って指定した場合はエラーを表示

させる。152 行目から 154 行目にかけて、指定したファイルを開くことができない場合はエラーを表示させる。155 行目から 161 行目にかけては、指定した文字を検索させる命令である。155 行目から 161 行目にかけての while 文では、『while ((ch4=fgetc(fp4)) !=-1)』とすることにより、『fp4』ファイルに書かれている文字を最後まで一文字づつ読み込み、『ch4』と『av[2]』が一致すれば表示させる。行目から行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。『ch4』と『av[2]』が一致しなければ、161 行目の printf( )関数を実行してプログラムを終了する。

case 4 のソースコードを以下に示す。

```
144: /*******************【case 4：ファイル中の文字検索 】******************/
145:        case 4:
146:              FILE *fp4;
147:              char ch4;
148:
149:              if(!(ac==3))
150:              {printf("error04：[使用方法]：<プログラム名>_<ファイル名>_<文字>");
151:              exit(1);}
152:              if(!(fp4=fopen(av[1],"r")))
153:              {printf("error04：ファイルを開くことができません。");
154:              exit(1);}
155:              while((ch4=fgetc(fp4)) !=-1)  // fp4 の中の av[2]を検索する
156:              {
157:                 if(ch4==*av[2])
158:                 {printf("“%s”の中に文字「 %s 」を見つけました。",av[1],av[2]);
159:                 exit(1);}
160:              }
161:              printf("“%s”の中に文字「 %s 」はありません・・・",av[1],av[2]);
162:              if(fclose(fp4)==-1)
163:              {printf("error04：ファイルを閉じることが出来ません。");
164:              exit(1);}
165:              break;
```

コマンドライン引数に“ e ”を指定し、case 4 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？4
ファイル中に文字「e」を見つけました。
```

case 5 では、FILE 型ポインタ『fp5a』、『fp5b』、char 型変数『ch5』を宣言する。171 行目から 173 行目にかけて、引数にファイル名の数を間違って指定した場合はエラーを表示させる。174 行目から 179 行目にかけて、それぞれ指定したファイルを開くことができない場合はエラーを表示させる。180 行目の feof( )関数『feof(fp5a)』は、『fp5a』が終わると 0 以外の値を返し、それ以外は 0 を返す。feof( )関数を『while(!feof(fp5a)) ch5=fgetc(fp5a)』とすることにより、feof 関数と fgetc( )関数を使用してファイルの終わりまで読み込むことを示す。180 行目から 191 行目にかけては、ファイル 1 からファイル 2 へ内容をコピーさせる命令である。もし読み込みや書き込みに失敗した場合、エラーを表示させる。192 行目から 197 行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。全て正常にプログラムが終了すると、198 行目の printf( )関数を実行してプログラムを終了する。case 5のソースコードを以下に示す。

```
166: /*********************【case 5：ファイルのコピー 】**********************/
167:        case 5:
168:              FILE *fp5a, *fp5b;
169:              char ch5;
170:
171:              if(!(ac==3))
172:              {printf("error05：[使用方法]：< ﾌﾟﾛｸﾞﾗﾑ>_< ｺﾋﾟｰ元>_< ｺﾋﾟｰ先>");
173:              exit(1);}
174:              if((fp5a=fopen(av[1],"r"))==0)
175:              {printf("error05：コピー元ファイルを開くことができません。");
176:              exit(1);}
177:              if((fp5b=fopen(av[2],"w"))==0)
178:              {printf("error05：コピー先ファイルを開くことができません。");
179:              exit(1);}
180:              while(!feof(fp5a))   //コピーする
181:              {
182:                ch5=fgetc(fp5a);
183:                if(ferror(fp5a))
184:                {printf("error05：コピー元ファイルを読み込めません。");
185:                exit(1);}
186:              if(!feof(fp5a))
187:                fputc(ch5,fp5b);
188:                if(ferror(fp5b))
189:                {printf("error05：コピー先ファイルを書き込めません。");
```

```
190:                exit(1);}
191:            }
192:            if(fclose(fp5a)==-1)
193:            {printf("error05：コピー元ファイルを閉じることが出来ません。");
194:            exit(1);}
195:            if(fclose(fp5b)==-1)
196:            {printf("error05：コピー先ファイルを閉じることが出来ません。");
197:            exit(1);}
198:            printf("“%s”ファイルを“%s”にコピーしました。",av[1],av[2]);
199:            break;
```

コマンドライン引数に“A”、“B”というファイルを指定し、case 5 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？5
A ファイルを B ファイルにコピーしました。
```

case 6 では、FILE 型ポインタ『fp6a』、『fp6b』、int 型変数『bite』、char 型変数『ch6』を宣言する。206 行目から 208 行目にかけて、引数にファイル名の数を間違って指定した場合はにエラーを表示させる。209 行目から 214 行目にかけて、それぞれ指定したファイルを開くことができない場合はエラーを表示させる。行目は fseek( )関数を使用し、『fseek(fp6a, 0, SEEK_END)』とすることで『fp6a』ファイルの内容の終端から0バイト移動した位置(内容の最後)に移動する。216 行目の『bite=ftell(fp6a)』はファイル『fp6a』現在位置の位置情報を返す。217 行目は『bite』を一文字づつ後退させる。218 行目から 230 行目の while 文で、『bite』がゼロになるまで(内容の最後からはじめに戻るまで)ファイル 1 からファイル 2 へ内容を逆にコピーさせる命令である。while{ }内の内容は、まず fseek( )関数を使用し、『fseek(fp6a,bite, SEEK_SET)』とすることにより、『fp6a』ファイルの内容のはじめから『bite』バイト分移動した位置を示す。そして、fgetc( )関数を使用して『fp6a』ファイルの内容を逆から読み取り、fputc( )関数を使用して『fp6b』ファイルに書き込む。もし読み込みや書き込みに失敗した場合、エラーを表示させる。231 行目から 236 行目にかけては、ファイルを閉じることができない場合のエラーを表示させる。すべての作業が正常に行われると 237 行目の printf( )関数を実行してプログラムを終了する。case 6 のソースコードを以下に示す。

```
200: /*********************【case 6：ファイルの逆コピー 】*********************/
201:         case 6:
202:                 int bite;
203:                 char ch6;
204:                 FILE *fp6a, *fp6b;
205:
206:                 if(!(ac==3))
207:                 {printf("error06：[使用方法]：< ﾌﾟﾛｸﾞﾗﾑ>_< ｺﾋﾟｰ元>_< ｺﾋﾟｰ先>");
208:                 exit(1);}
209:                 if(!(fp6a=fopen(av[1], "r")))
210:                 {printf("error06：コピー元ファイルを開くことができません。");
211:                 exit(1);}
212:                 if(!(fp6b=fopen(av[2], "w")))
213:                 {printf("error06：コピー先ファイルを開くことができません。");
214:                 exit(1);}
215:                 fseek(fp6a, 0, SEEK_END);  //fp6a の終端に達する
216:                 bite=ftell(fp6a);
217:                 bite=bite-1;
218:                 while(bite>= 0)    // fp6a から fp6b へ逆順にコピー
219:                 {
220:                   fseek(fp6a, bite, SEEK_SET);
221:                   ch6=fgetc(fp6a);
222:                   if(ferror(fp6a))
223:                    {printf("error06：コピー元ファイルを読み込めません。");
224:                    exit(1);}
225:                   fputc(ch6,fp6b);
226:                   if(ferror(fp6b))
227:                   {printf("error06：コピー先ファイルを書き込めません。");
228:                   exit(1);}
229:                    bite--;
230:                 }
231:                 if(fclose(fp6a)==-1)
232:                 {printf("error06：入力ファイルを閉じることが出来ません。");
233:                 exit(1);}
234:                 if(fclose(fp6b)==-1)
235:                 {printf("error06：出力ファイルを閉じることが出来ません。");
```

```
236:            exit(1);}
237:            printf("“ %s ”ファイルを“ %s ”に逆コピーしました。",av[1],av[2]);
238:            break;
```

コマンドライン引数に“ A ”、“ B ”というファイルを指定し、case 6 **を実行すると次のように画面に表示される。**

```
選択 1] ～ 9]？6
A ファイルを B ファイルに逆コピーしました。
```

case 7 **では、**FILE **型ポインタ『**fp7a**』、『**fp7b**』、『**fp7temp**』、**char **型変数『**ch7**』を宣言する。**244 **行目から** 246 **行目にかけて、引数にファイル名の数を間違って指定した場合はエラーを表示させる。**247 **行目から** 255 **行目にかけて、それぞれ指定したファイルを開くことができない場合はエラーを表示させる。**256 **行目から** 266 **行目にかけては、ファイル** 1 **を一時ファイルにコピーする命令である。**feof( )**関数と** fgetc( )**関数を『**while(!feof(fp7a)) ch5=fgetc(fp7a)**』とすることにより、ファイルの終わりまで読み込む命令となり、**feof( )**関数と** fputc( )**関数『**if(!feof(fp7a))fputc(ch7,fp7temp)**』とすることにより、ファイルの内容を全て『**p7temp**』に書き込む命令となる。もしそれぞれの命令で読み込みや書き込みに失敗した場合、**ferror( )**関数によりエラーを表示させる。行目から行目にかけては、ファイルを閉じることができない場合はエラーを表示させる。この方式で一時ファイルをファイル** 2 **にコピーし、一時ファイルをファイル** 2 **にコピーすることでファイルの交換を行う。すべての作業が正常に行われると** 316 **行目の** printf( )**関数を実行してプログラムを終了する。**case 7 **のソースコードを以下に示す。**

```
239: /**********************【case 7：ファイルの交換 】**********************/
240:        case 7:
241:            FILE *fp7a, *fp7b, *fp7temp;
242:            char ch7;
243:
244:            if(!(ac==3))
245:            {printf("error07：[使用方法]：< ﾌﾟﾛｸﾞﾗﾑ>_<ﾌｧｲﾙ 1>_<ﾌｧｲﾙ 2>");
246:            exit(1);}
247:            if(!(fp7a=fopen(av[1], "r")))
248:            {printf("error07：1 番目のファイルを開くことができません。");
249:            exit(1);}
```

```
250:            if(!(fp7b=fopen(av[2],"r")))
251:            {printf("error07：2 番目のファイルを開くことができません。");
252:            exit(1);}
253:            if(!(fp7temp=fopen("temp", "w")))
254:            {printf("error07：一時ファイルを開くことができません。");
255:            exit(1);}
256:            while(!feof(fp7a))     // fp7a を fp7temp にコピー
257:            {
258:              ch7=fgetc(fp7a);
259:              if(ferror(fp7a))
260:              {printf("error07：一番目のファイルを読み込めません。");
261:              exit(1);}
262:              if(!feof(fp7a))fputc(ch7,fp7temp);
263:              if(ferror(fp7temp))
264:              {printf("error07：一時ファイルを書き込めません。");
265:              exit(1);}
266:            }
267:            if(fclose(fp7a)==-1)
268:            {printf("error07：ファイルを閉じることが出来ません。");
269:            exit(1);}
270:            if(!(fp7a=fopen(av[1], "w")))
271:            {printf("error07：1 番目のファイルを開くことができません。");
272:            exit(1);}
273:            while(!feof(fp7b))      // fp7b を fp7a にコピー
274:            {
275:              ch7=fgetc(fp7b);
276:              if(ferror(fp7b))
277:              {printf("error07：二番目のファイルを読み込めません。");
278:              exit(1);}
279:              if(!feof(fp7b)) fputc(ch7,fp7a);
280:              if(ferror(fp7a))
281:              {printf("error07：一番目のファイルを書き込めません。");
282:              exit(1);}
283:            }
284:            if(fclose(fp7b)==-1)
285:            {printf("error07：ファイルを閉じることが出来ません。");
```

```
286:            exit(1);}
287:            if(fclose(fp7temp)==-1)
288:            {printf("error07：ファイルを閉じることが出来ません。");
289:            exit(1);}
290:            if(!(fp7b=fopen(av[2], "w")))
291:            {printf("error07：2 番目のファイルを開くことができません。");
292:            exit(1);}
293:            if(!(fp7temp=fopen("temp", "r")))
294:            {printf("error07：一時ファイルを開くことができません。");
295:            exit(1);}
296:            while(!feof(fp7temp))  // fp7temp を fp7b にコピー
297:            {
298:              ch7=fgetc(fp7temp);
299:              if(ferror(fp7temp))
300:              {printf("error07：一時のファイルを読み込めません。");
301:              exit(1);}
302:              if(!feof(fp7temp)) fputc(ch7, fp7b);
303:              if(ferror(fp7b))
304:              {printf("error07：二番目のファイルを書き込めません。");
305:              exit(1);}
306:            }
307:            if(fclose(fp7a)==-1)
308:            {printf("error07：ファイルを閉じることが出来ません。");
309:            exit(1);}
310:            if(fclose(fp7b)==-1)
311:            {printf("error07：ファイルを閉じることが出来ません。");
312:            exit(1);}
313:            if(fclose(fp7temp)==-1)
314:            {printf("error07：ファイルを閉じることが出来ません。");
315:            exit(1);}
316:            printf("%s ”ファイルと“%s ”ファイルを交換しました。",av[1],av[2]);
317:            break;
```

コマンドライン引数に“A”、“B”というファイルを指定し、case 7 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？7
A ファイルと B ファイル交換しました。
```

case 8 では、char 型変数『x』を宣言する。の 321 行目から 323 行目にかけて、引数にファイル名の数を間違って指定した場合はエラーを表示させる。324 行目から 332 行目にかけて、削除するかどうかを scanf( )関数を使用し、Yes /No で選択させ、Yes ならそのファイルを削除する命令である。326 行目は toupper( )関数を使用し、『if(toupper(getchar( ))=='Y')』は、キーボードから入力された文字が大文字の『Y』、小文字の『y』ならば、次の文を実行するという命令である。328 行目は、remove( ) 関数を使い、『remove(av[1])』は『av[1]』に一致するファイルを削除する命令である。Yes /No の選択で、削除をやめる場合は『Y』以外のキーを打つと終了する。case 8 のソースコードを以下に示す。

```
318: /**********************【case 8：ファイルの削除】************************/
319:        case 8:
320:              char x;
321:              if(!(ac==2))
322:              {printf("error08：[使用方法]：<プログラム名>_<ファイル名>");
323:              exit(1);}
324:              printf("削除しますか？(Yes→Y/No→Y 以外)： ");
325:              scanf("%c",&x);
326:              if(toupper(getchar())=='Y')
327:              {
328:                 remove(av[1]);
329:                 printf("“%s”ファイルを削除しました。",av[1]);
330:              }
331:              else
332:              printf("削除をやめます。");
333:              break;
```

case 8 を実行すると次のように画面に表示される。

```
選択 1] ～ 9]？8
削除しますか？(Yes→Y/No→Y 以外)：Y
ファイルを削除しました。
```

case 9 でプログラムを終了させる。

```
334: /*************************【case9：終了】*************************/
335:         case 9:
336:                 printf("プログラムを終了します。");
337:                 break;
338:         default:
339:                 printf("番号が不正です。");
340:       }
341:   }
342:     return 0;
343: }
```

第 4 章　　まとめ

ファイル入出力を使った電話帳プログラムでは、すべての命令をメニュー画面から行うためにそれぞれのファイル処理を一つの関数にし、独立させた。

コマンドライン引数を使ったファイル処理プログラムでは、コマンドライン引数の数によってそれぞれファイルの処理の仕方が違うので、まず①if 文での『(ac==1)』の場合と、②else 文での『(!(ac==1))』の場合とで大きく二つに分けた。①の場合は画面上でファイル処理をするため、コマンドライン引数に何を指定すればよいかを表示させる。②の場合はそれぞれのファイル処理の内容を選択するわけであるが、コマンドライン引数でのファイルや文字の入力が正しくなければ、エラーメッセージとしてそのファイル処理の使い方を表示させる。それぞれのファイル処理プログラムは、必要とするコマンドライン引数が異なるため、ファイル処理が正常に行われるとプログラムは終了する。

また、改善点として、書き込まれたファイルの内容についての編集などがあげられる。

第 5 章　　後書き

プログラムを完成させるまでには、①プログラムを作成、②予想通りの実行結果になるかを確認、③動作がおかしければその原因の究明、④プログラムを修正、⑤プログラム完成という五つの工程がある。しかし、この①から④の工程を繰り返すことで⑤の完成へ導くわけであるが、プログラムを作成する中で(1)読みやすいプログラム、(2)安全なプログラム、の二つの点について特に配慮した。(1)については余計なコメントは避け、制御の流れが単純になるように何度もプログラムを検討した。(2)については、実行中に『Ctrl＋C』を押すことのないようなエラーに対する処置を常に心がけた。自分で作り、自分で使うプログラミングにおいてはエラーに対して自分が注意すればよいので、プロのプログラマーが作るような極めて頑丈なプログラムを作る必要はないが、実行を行うのが第三者ということを考えて、なるべくそれに近い頑丈なプログラムを作成するようにした。

C 言語を学習したことで、次のステップとしてプロフェッショナル向けプログラミング言語である C++、Visual C++、Java などをこれから勉強を進めていきたい。

第 6 章　謝辞

最後に卒業研究に関する指導をしてくださった原 央教授、武田光由先生に深く感謝申し上げます。

第 7 章　参考文献

- 塚越一雄／著 ： [決定版] はじめての C＋＋ (技術評論社)
- Herbert Schildt／著　SE 編集部／翻訳　柏原正三／監修 ： 独習 C 改訂版 (翔泳社)
- Steve Oualline／著　望月康司／監訳　谷口 功／訳：C実践プログラミング　(オライリージャパン社)