

人と技術で次代を拓く

MEITEC

Engineering Firm at The Core

改訂履歴


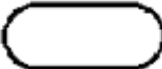


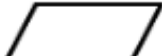




Ver.	日付	対応者	項目	改定内容	確認	承認
1.00	2023/02/01	市川	新規作成			
1.01	2023/11/30	市川	エスケープシーケンス追加	システム開発実習で画面制御で使用する機能であるため追記		
1.02	2025/03/6	喜多	CSVファイル読み書き処理追加	システム開発実習で使用するCSVファイル入出力の組込処理説明を追記		

C言語基礎

Ver 1.02

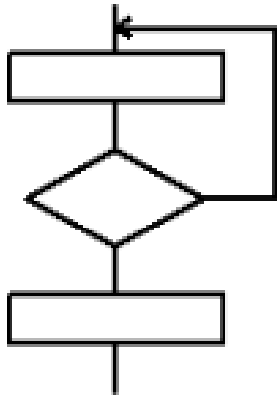
1. フローチャートの記号

フローチャート

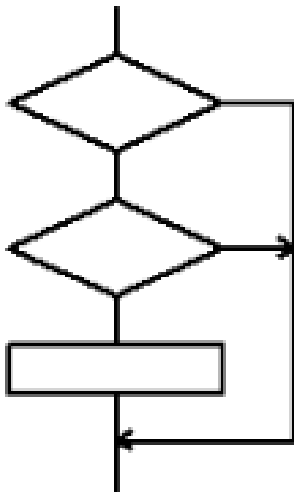
記 号	名 称	働 き
	流れ線	処理の順序を表す。順序を明確にするために、矢印を付けることもある。
	端子	処理の開始・終了を表す。
	処理	判断などの処理以外の処理を表す。
	準備	変数の宣言や初期値の設定などの処理を表す。
	入出力	データの入出力を表す。
	判断	条件により、流れが二つ以上に分岐する処理を表す。
	ループ端	繰り返しの開始と終了を表す。
	書類	プリンタなどへのデータ出力を表す。
	定義済み処理	別に用意した処理を利用することを表す。

2. フローチャート作成上の約束

フローチャート



- ① 処理の流れの方向は「左から右へ」、「上から下へ」とする。
これ以外の場合は、矢印を使用する。



- ② 流れ線は互いに交差してもよい。

- ③ 2つ以上の流れ線を集めて、1つにしてもよい。

3. フローチャートの基本型

フローチャート

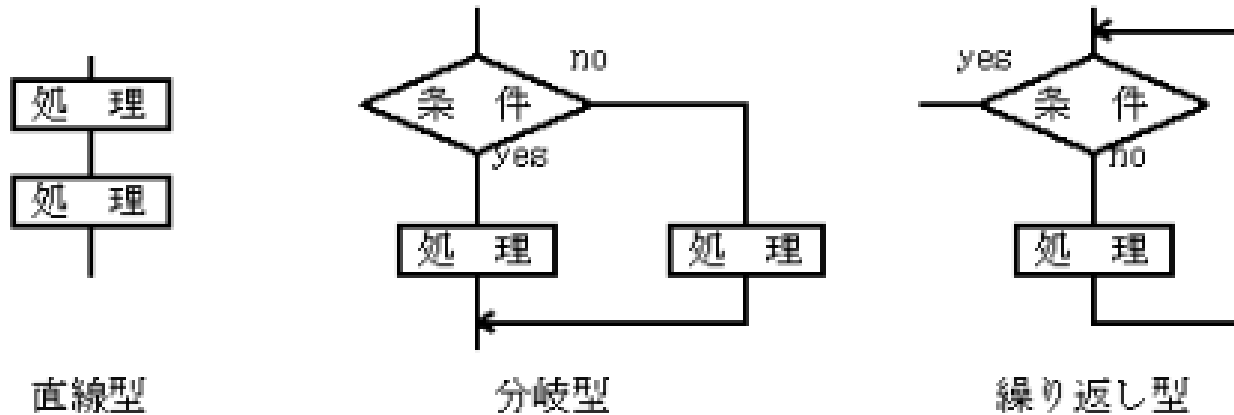


図 フローチャートの基本型

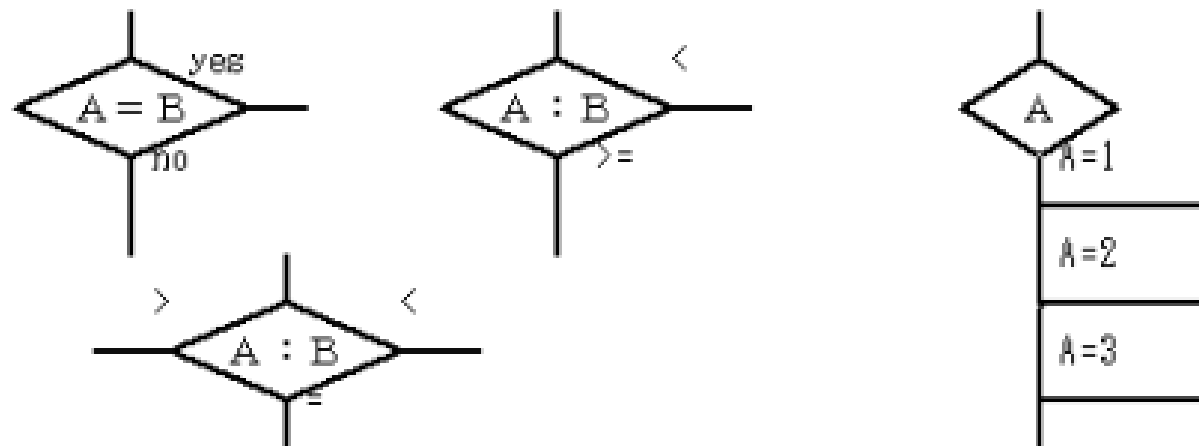


図 条件判断の表し方

状態遷移

●呼び鈴の仕様

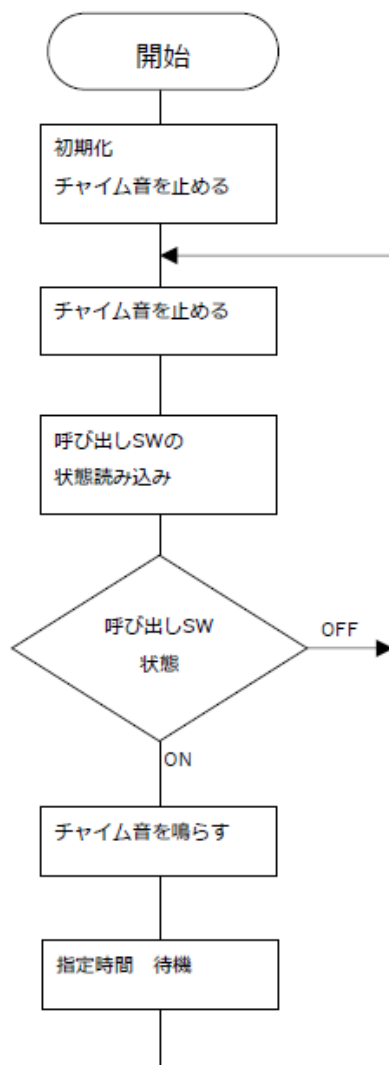
前提) 常時、電源が入っているものとする

部位の説明	入出力	概要	備考
呼び出しSW	入力	相手を呼びたい時に押す。	
チャイム	出力	指定時間呼び出し音を鳴らす。	

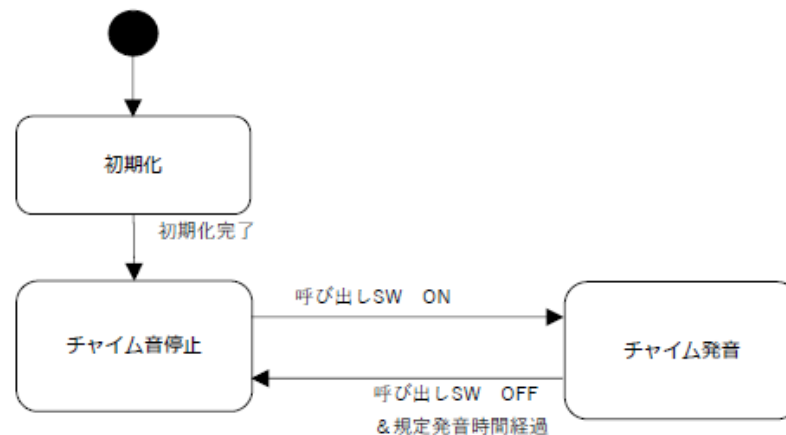


状態遷移

●フローチャート



●状態遷移図



状態遷移

●扇風機の仕様

（前提）常時、コンセントの電源が入っているものとする

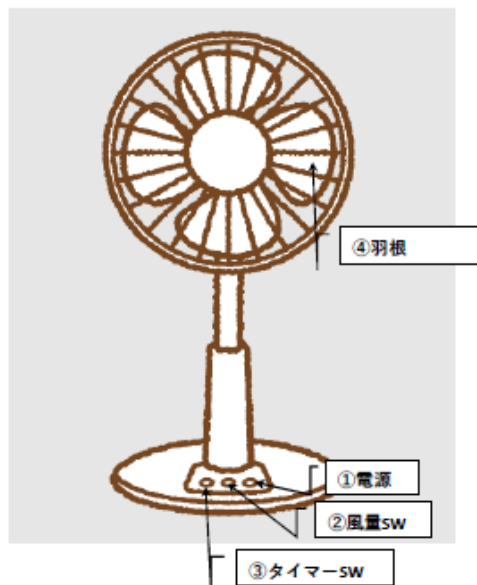
部位の説明	入出力	概要	図番	備考
電源SW	入力	電源をON/OFFする。 ONで運転開始（羽根を回す）。 OFFで運転停止（羽根を停止する）	①	（注1）
風量SW		風量を変える（弱中強の3段階） ※押下するごとにサイクリックに切り替え出力 （弱⇒中⇒強⇒弱）	②	（注2）
タイマーSW		1時間、2時間に設定できる。 ※押下するごとにサイクリックに切り替える。 （OFF⇒1時間⇒2時間⇒OFF）	③	（注3）
羽根を回す	出力	弱中強（停止）	④	

（注1）電源をON/OFFを切り替える（ON⇒OFF、OFF⇒ON）

（注2）電源OFFした場合は、風量はリセットされる（初期値は“中”）

（注3）電源OFFした場合は、タイマーはリセットされる

【イメージ図】



◆Cプログラムのスタイル

C言語基礎1

```

/*
/*      Cプログラムのサンプル
/*
#include <stdio.h>

int main(void)
{
    int ten;
    printf("点数を入力してください。> ");
    scanf("%d", &ten);

    if (ten >= 60)
    {
        printf("合格です。%n");
        printf("おめでとう！%n");
    }
    else
    {
        printf("不合格です。%n");
        printf("もっと勉強しましょう。%n");
    }

    printf("次の試験は来週の水曜日です。%n");

    return 0;
}
    
```

コメント
 int main(void)で始まる
 セミコロン
 インデント
 /* 点数の宣言 */
 /* 点数入力 */
 /* 点数が 60 点以上のとき */
 /* 点数が 60 点未満のとき */
 複合文

1. プログラムは main(void)関数で書き始める。
2. 慣用的に小文字を用いて書く。
3. 文の終わりにセミコロン (;) をつける。
4. 複合文は { } で囲み、ブロック化する。
5. インデント (字下げ) で見やすくする。
6. コメント (注釈) は /* と */ で囲む。

◆main関数の書き方 (テンプレート)

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

```
int main (void)  
{
```

(1) 変数とデータ型 P39～P54

(2) 処理

- ・ 制御文(if/演算子/switch/for/while) P89～P128
- ・ 式と演算(演算子:加算・減算・乗算・除算) P62
- ・ 関数(引数と戻り値) P129
- ・ 配列・文字列とポインタ P167

```
return 0;  
}
```

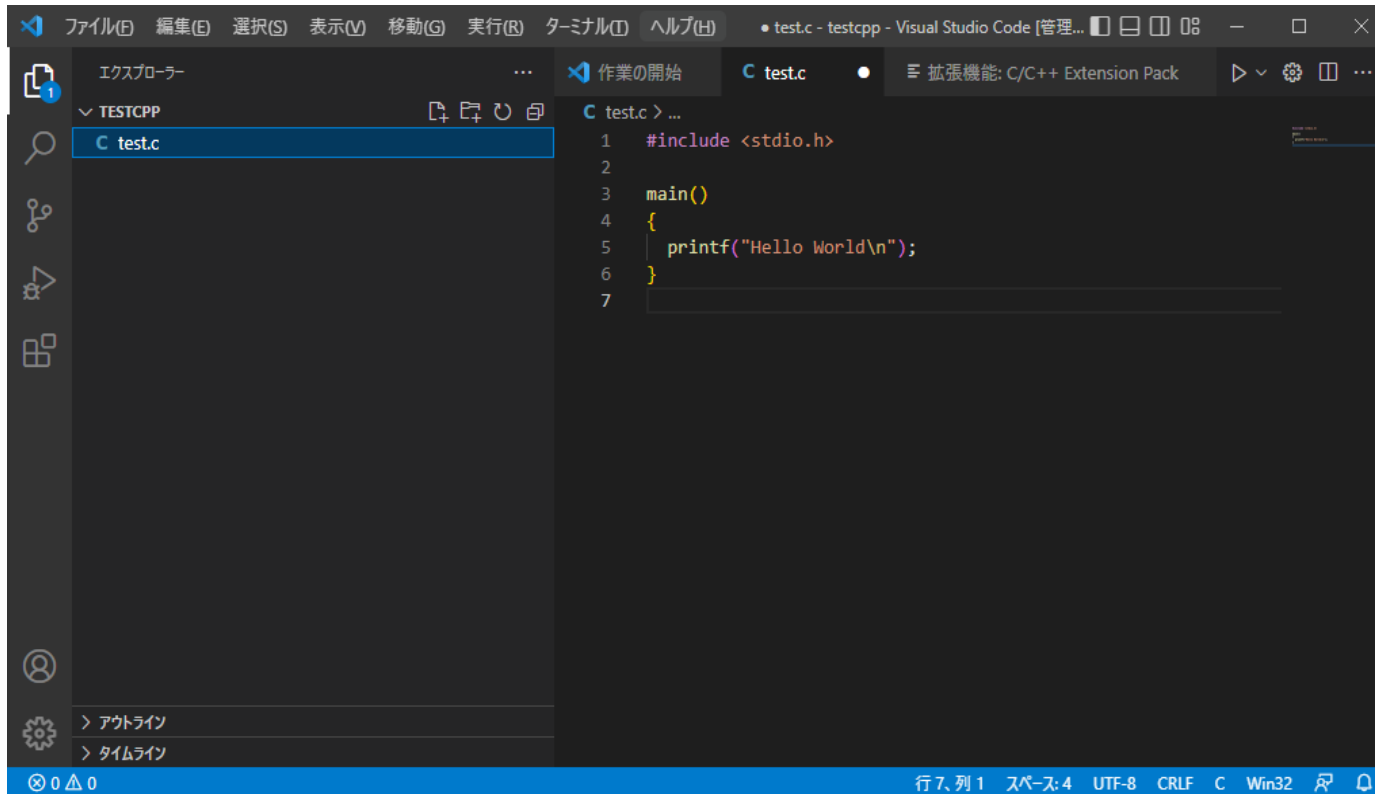
※上記のページ数は「猫でもわかるC言語」を参照のこと

上
か
ら
順
に
実
行

この書き方、お約束なので覚えて下さい！！

◆VSCode 起動例

C言語基礎1



C言語基礎1

変数の有効範囲

「有効範囲」とはその変数を参照できる範囲のことです。C言語では、変数の宣言をソースプログラムのどこに書くかによって「ローカル変数」と「グローバル変数」に分けられ、この有効範囲が異なってきます。この有効範囲は「記憶クラス」と密接な関係がありますので、まずは有効範囲から理解してください。

(1) ローカル変数 (局所変数)

- 関数内で定義され、その関数内でのみ使用できます。
- 複数の関数が同一の変数名を用いたとしても衝突しません。

※ 前章までの学習で使用していた変数はすべてこのローカル変数に該当します。

(2) グローバル変数 (広域変数)

- 関数外で定義され、定義以降のどの関数からでも使用できます。
- システムでグローバル変数としては1つの定義しか許されません。

※ 仮にある関数のローカル変数にグローバル変数と同一の名前が存在するときにはその関数内ではローカル変数が優先します。

C言語基礎1

【ローカル変数とグローバル変数の有効範囲】

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

```
void goukei(int n);
```

```
int g; ← グローバル変数 g の宣言
```

```
int main(void)
```

```
{
```

```
    int a; ← ローカル変数 a の宣言
```

```
    g = 20;
```

```
    a = 10;
```

```
    goukei( a );
```

```
    return 0;
```

```
}
```

main()
aの有効範囲

gの有効範囲

```
void goukei(int n )
```

```
{
```

```
    int a, sum = 0; ← ローカル変数  
                    a と sum の  
                    宣言
```

```
    for (a=1; a<=n; a++)
```

```
        sum = sum + a;
```

```
    printf("合計=%d\n",sum);
```

```
    printf("g = %d\n",g);
```

```
}
```

goukei()
nとaとsum
の有効範囲

フローチャート 演習

C言語基礎1

配布するフローチャートの内容についてC言語のコードを完成させてください。

3章 変数とデータ型

C言語で扱われるデータは大きく「定数」と「変数」に分けられます。

定数： 値を変えないもの

変数： 任意の値を取りうるもの

ことができます。

たとえば、`a = 10; a = 20;` では、

`a` は「変数」、`10` や `20` は「定数」になります。

※ 「`a = 10;`」 は、`a` に `10` を代入するという意味になります。
詳しくは「[3-1. 代入演算子\(=\)](#)」を参照してください。

3章 変数とデータ型

C言語基礎1

定数は具体的に下記のように分類できます。

数値定数	整数定数	8進数	先頭に「0」をつけて表記（077など）
		10進数	通常の10進数と同じ表記
		16進数	先頭に「0x」をつけて表記（0x41など）
	浮動小数点定数	小数点以下が扱える数（16.25 や 1.0e-3 など）	
文字定数	1文字のこと。'A' や 'B' のように ' ' で囲んで表記		
文字列定数	複数文字のこと。"ABC" や "computer"のように " "で囲んで表記		

※ 「浮動小数点数」とは浮動小数点表現方式の実数で、C言語では小数点以下の数を扱うときに用います。

※ 1.0e-3 は 1.0×10^{-3} (0.001) のことです。C言語では浮動小数点数をこのようにも表記します。

3章 変数とデータ型

変数とはコンピュータのメモリ上に実際に用意された作業エリアです。

```
int data;
```

【変数の主な型】

型指定	データ型	バイト幅	扱える数値の範囲
char	文字型	1	-128~127
int	整数型	2	-32768~32767
long	倍長整数型	4	-2147483648~2147483647
float	単精度浮動小数点型	4	3.4E-38~3.4E+38
double	倍精度浮動小数点型	8	1.7E-308~1.7E+308

4章 式と演算子

代入演算子 (=)

等号の右側の値を左側の変数に代入します。

定数の代入 : $a = 3;$... 変数 a に **定数 3** を代入

変数を代入 : $a = b;$... 変数 a に **変数 b** を代入

式の代入 : $a = c + 4;$... 変数 a に **式「 $c + 4$ 」** を計算して代入

自身を更新 : $a = a + 2;$... 変数 **a 自身を「 $+2$ 」** して代入

なお、次のような代入はできませんので注意してください。

これらはできません

$1 = a;$... 定数に変数や式は代入できません

$a + b = 3;$... 式に定数や変数は代入できません

4章 式と演算子

算術演算子 (+ - * / %)

加減乗除および余りを求めます。

演算	演算子	例	意味
加算	+	$a + b$	a に b を加える
減算	-	$a - b$	a から b を引く
乗算	*	$a * b$	a に b をかける
除算	/	a / b	a を b で割る
剰余算	%	$a \% b$	a を b で割った余り

4章 式と演算子

インクリメント (++)・デクリメント (--) 演算子

+1 および -1 の演算を行います。

演算	演算子	例	意味
インクリメント	++	a++	a に 1 を加える (後置演算)
		++a	a に 1 を加える (前置演算)
デクリメント	--	a--	a から 1 を引く (後置演算)
		--a	a から 1 を引く (前置演算)

4章 式と演算子

◆使用例

(例1)

```
int a = 10;
a++; ... aは11
```

(例2)

```
int a = 10;
++a; ... aは11
```

しかし、代入演算子(=)と共に用いると、結果が違ってきますので注意が必要です。

※前置演算：先に処理(++や--)をしてから代入

(例) n = 2;
 a = ++n;
 └─┘
 先に処理 : nは3
 └─┘
 後から代入 : aは3

※後置演算：先に代入をしてから処理(++や--)

(例) n = 2;
 a = n++;
 └─┘
 先に代入 : aは2
 └─┘
 後から処理 : nは3

4章 式と演算子

演算子の優先順位

種類	演算子	結合規則	優先順位
関数, 添字, 構造体メンバ参照, 後置増分/減分	() [] . -> ++ --	左→右	<div>高</div> <div>↑</div> <div>↓</div> <div>低</div>
前置増分/減分, 単項式※	++ -- ! ~ + - * & sizeof	左←右	
キャスト	(型名)		
乗除余	* / %	左→右	
加減	+ -		
シフト	<< >>		
比較	< <= > >=		
等値	== !=		
ビットAND	&		
ビットXOR	^		
ビットOR			
論理AND	&&		
論理OR			
条件	?:	左←右	
代入	= += -= *= /= %= &= ^= = <<= >>=		
コンマ	,	左→右	

4章 式と演算子

ビット演算子

ビット単位でデータ操作をするものです。対象は整数に限られます。

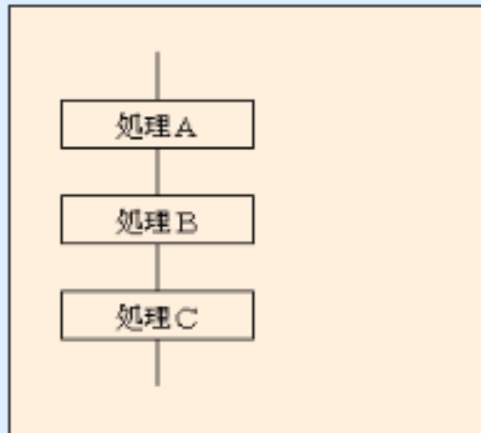
【ビット演算子】

演算子	説明
&	ビットごとの AND
	ビットごとの OR
^	ビットごとの XOR
~	ビットごとの反転 (1 の補数)
<<	左シフト
>>	右シフト

5章 制御文

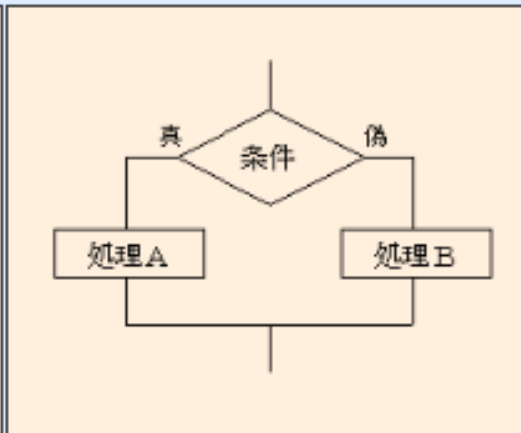
C言語基礎1

(順次構造)



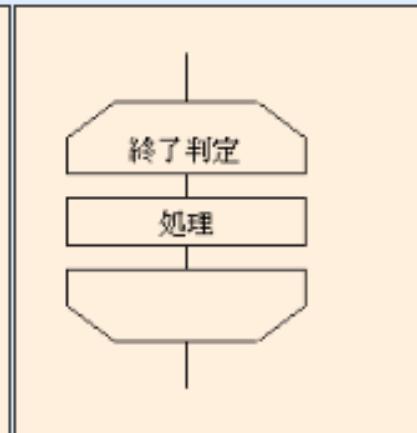
処理が現れた順番に実行する形式

(選択構造)



条件によって処理が分岐する形式

(反復構造)

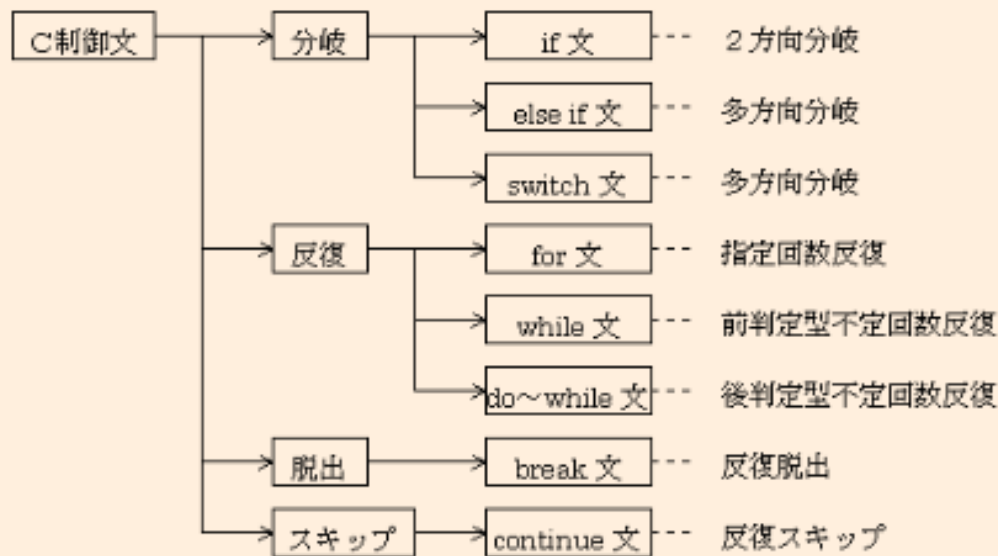


処理を繰り返し実行する形式

5章 制御文

C言語では、これらの制御構造を実現する制御文が用意されています。

また、反復処理から脱出するための「break文」や、反復処理をスキップするための「continue文」も用意されています。



5章 制御文 (If文)

パターン	書き方	フローチャート	例
基本形	<pre>if (条件式) 文1; else 文2;</pre> <p>↑ タブで字下げ</p>	<p>条件式が 真なら文1 を実行 偽なら文2 を実行</p>	<pre>if (a > 10) a = 0; else a++;</pre>
else節の省略型	<pre>if (条件式) 文1;</pre>	<p>条件式が真なら文1 を実行</p>	<pre>if (a == 0) a = 2;</pre>
複合文の形式 ({}で囲んだ文の集合を複合文と呼ぶ) ※ 複合文の else節 も省略できる	<pre>if (条件式) { 文11; 文12; } else { 文21; 文22; }</pre> <p>※ 複合文の文は {}で囲んで 複合文にする</p>	<p>(70-94-7)</p> <p>条件式が 真なら文11 と文12 を実行 偽なら文21 と文22 を実行</p>	<pre>if (a < 10) { a = 1; b = 2; } else { a = 3; b = 4; }</pre>

5章 制御文 (If文)

【2. 関係演算子】

2つの値の大小関係を比較し、真のときには1を、偽のときには0を生成します。

【関係演算子】

演算子	意味	使用例
>	より大きい	if (a > b)
>=	より大きいか、等しい (以上)	if (a >= b)
<	より小さい	if (a < b)
<=	より小さいか、等しい (以下)	if (a <= b)
==	等しい	if (a == b) (注1)
!=	等しくない	if (a != b)

5章 制御文 (If文)

【3. 論理演算子】

論理演算子を用いると、複数の条件を組み合わせることができます。

【論理演算子】

演算子	意味	使用例
&&	論理積 (かつ)	if (a > 0 && b > 0)
	論理和 (または)	if (a > 0 b > 0)
!	否定 (でない)	if (!a) (注2)

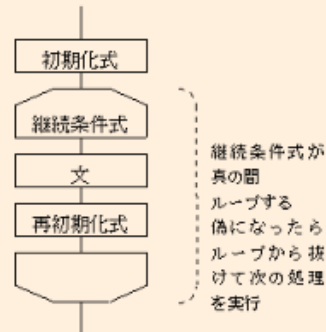
5章 制御文 (for 文)

(形式)

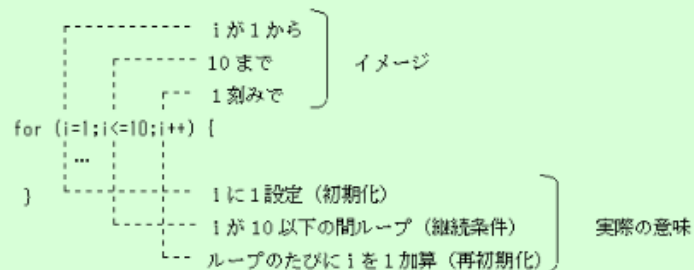
```
for (初期化式; 継続条件式; 再初期化式) {
    文;
}
```

- 文は複合文も可。
- 複数行の文の場合には { } で囲み複合文に。
単文の場合には { } は省略可能。
- 再初期化式の後に「;」は入れてはいけない。

(フローチャート)



(例)



- 初期化式は通常の代入文を記述。
- 継続条件式はこの式が真の間、ループを続けるという意味。
「～の間」であって、「～まで」ではないので注意。
- 再初期化式には「i++」ばかりではなく、「i--」や「i = i + 2」などいろいろな値をとることが可能。
- ループ変数（使用例では i）は int 型以外にも char 型や double 型なども可能。

5章 制御文 (for 文)

◆例

```
for( i=1 ; i<3 ; i++ ) {  
    文1;  
    ...  
}
```

1. 初期設定値は「i=1」です。最初のみ評価される。
2. この継続条件式は(i<3)は真です。
3. 条件式が真なので文1は実行される。
4. 再設定値が評価される(まだ1増えない)
5. 次の条件が評価される(iは1増えて2になる)
6. 条件式が真なので文1は実行される。
7. 再設定値が評価される(iは2のまま)
8. 次の条件が評価される(iは1増えて3になる)
9. 条件式は偽なので文1は実行されない

5章 制御文 (while 文)

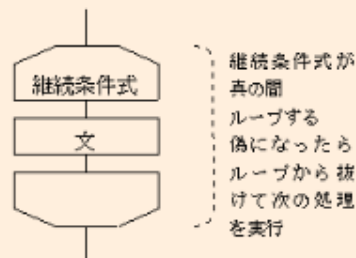
条

(形式)

```
while (継続条件式) {  
    文;  
}
```

- 継続条件が真である間、文を繰り返し実行。
- 継続条件式がはじめから偽の場合は一度も実行されない。
- 文は複合文 ({} で囲んだ文の集合) も可。
- 単文の場合には { } は省略可能。

(フローチャート)



(例)

```
int    i;  
int    a[5] = {1300, 4320, 985, 2130, -1};  
  
i = 0;  
while (a[i] != -1) {  
    printf("%d\n", a[i]);  
    i++;  
}
```

この条件でループから脱出

-----a[i]が -1 でない間ループせよ

ループ範囲

実行結果

```
1300  
4320  
985  
2130
```

- for文と異なり、初期化式および再初期化式は while文の外に記述する。
- for文も while文も共に反復制御を行うが、一般に for文は「〇回処理を繰り返す」ときに使用し、while文は「~の間処理を繰り返す」ときに使用する。ただし、while文を用いても「〇回処理を繰り返す」という制御は可能である。(以下の例を参考のこと)

C言語基礎1

5章 制御文 (do while 文)

条件式を後判定して反復制御を行います。

(形式)

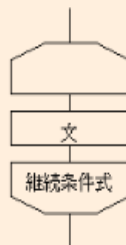
```
do {
    文;
} while (継続条件式); ← 忘れないこと
```

- まず文を実行してから、継続条件の判定を行う。

継続条件式が真である間、文を繰り返し実行。

- while文は一度も実行されないことがあるが（最初から条件が「偽」のとき）、do~while文ではとりえず 1回は文を実行する。
- 文は複合文（{ } で囲んだ文の集合）も可。
- 単文なら { } 省略可。
- 継続条件式の後の (;) を忘れないよう注意。

(フローチャート)



まず文を 1 回
実行してか
ら、継続条件
式の判定を行
う。
継続条件式が
真ならループ
する。
偽になったら
ループから抜
けて次の処理

(例)

```
int main(void)
{
    int    wa,data;
    wa = 0;
    do {
        printf("整数値を入力");
        scanf("%d",&data);
        wa = wa + data;
        printf("wa = %d\n",wa);
    } while (data != 0);
    return 0;
}
```

ループ開始

dataを入力

waに加算

ループ範囲

dataが 0 でない間ループせよ

- while文と異なり、まず処理をするため、data が 0 かが判定するため、ループ処理の前に data を入力する必要がない。
- do~while文は使用頻度は高くないが、「とりえず 1回は実行する。場合によっては繰り返す」場合に便利。

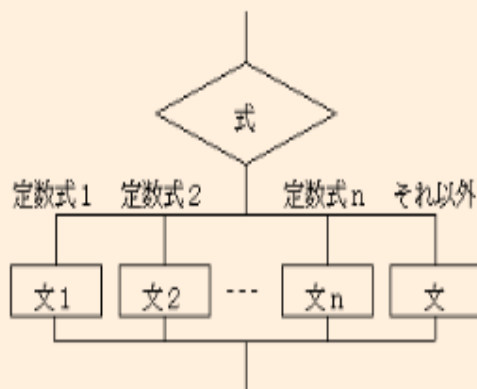
5章 制御文 (switch 文)

条件式を判定して多方向分岐を行います。

(形式)

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

(フローチャート)



式の値が

定数式1 と等しければ、文1 実行

定数式2 と等しければ、文2 実行

定数式n と等しければ、文n 実行

それ以外ならば、文実行

制御文 演習

C言語基礎1

【練習問題①】

int 型の変数 x、y にそれぞれ数値を入力し、x が y より大きい場合に、“xはyより大きい” という文を表示するプログラムを作成しなさい。

【練習問題②】

int 型の変数 x、y にそれぞれ数値を入力し、x が y より大きい場合には“x は y より大きい”、x が y より小さい場合には“xはyより小さい” と表示するプログラムを作成しなさい。

制御文 演習

C言語基礎1

【練習問題③】

2つの整数値を入力し、大きい方（小さい方）の数を表示するプログラムを作成しなさい。

【練習問題④】

int 型の変数 x、y にそれぞれ数値を入力し、x が y より大きい場合には“xはyより大きい”、x が y より小さい場合には“xはyより小さい”、x と y が等しい場合には“xとyは等しい”と表示するプログラムを作成しなさい。

制御文 演習

【練習問題⑤】

試験の点数を入力し、対応する成績を表示するプログラムを3種類作成しなさい。

試験は100点満点（0点～100点）とし、点数と成績の対応を以下のようにします。

ケース1

60点以上：“合格”

60点未満：“不合格”

ケース2

80点以上：“たいへんよくできました。”

60点以上、80点未満：“よくできました。”

60点未満：“ざんねんでした。”

ケース3

80点以上：“優”

70点以上、80点未満：“良”

60点以上、70点未満：“可”

60点未満：“不可”

制御文 演習

C言語基礎1

【練習問題⑥】

曜日と、午前、午後、夜間の区別を入力し、病院が開いているか、休診であるかを表示するプログラムを作成しなさい。

- 開いているか、休診であるかは、次の表に従います。

	日曜	月曜	火曜	水曜	木曜	金曜	土曜
午前	休診	○	休診	○	○	休診	○
午後	休診	○	○	○	○	○	休診
夜間	休診	○	○	休診	○	○	休診

※ 曜日の入力、午前、午後の入力は、次のようなガイドを表示して数値で行います。

>0=日曜、1=月曜、2=火曜、3=水曜、4=木曜、5=金曜、6=土曜
 >0=午前、1=午後、2=夜間

制御文 演習

C言語基礎1

【練習問題⑦】

月を表す数値を入力し、その月の初めから年末までにある祝日を表示するプログラムを作成しなさい。

表示する祝日は以下の日とします。

- 1月：元日、成人の日
- 2月：建国記念の日
- 3月：春分の日
- 4月：昭和の日
- 5月：憲法記念日、みどりの日、こどもの日
- 7月：海の日
- 9月：敬老の日、秋分の日
- 10月：体育の日
- 11月：文化の日、勤労感謝の日
- 12月：天皇誕生日

※ switch文を使用すること。

制御文 演習

C言語基礎1

【演習問題②】

プログラム（C言語）とフローチャートを作成して下さい

月を表す数値を入力し、その月の初めから年末までにある祝日を表示するプログラムを作成しなさい。

表示する祝日は以下の日とします。

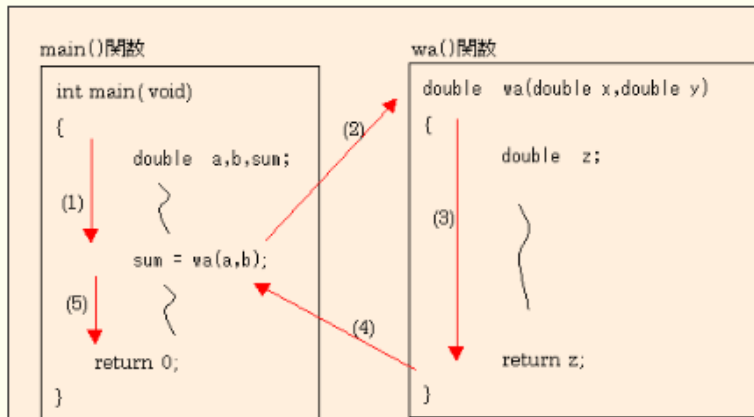
- 1月：元日、成人の日
- 2月：建国記念の日
- 3月：春分の日
- 4月：昭和の日
- 5月：憲法記念日、みどりの日、こどもの日
- 7月：海の日
- 9月：敬老の日、秋分の日
- 10月：体育の日
- 11月：文化の日、勤労感謝の日
- 12月：天皇誕生日

※ switch文を使用すること。

6章 関数

(1) 関数を用いた場合の処理の流れ

まず、関数を用いた場合、プログラムはどのような順番で実行されるのかを下図に示します。下図では main() から wa() という関数が呼ばれるようになっています。



1. main()の先頭から wa()までを実行。
2. wa()に処理が飛ぶ。
 このとき wa()にデータ a, b を渡す。
 これを実引数（じつひきすう）と呼ぶ。
3. wa()は渡された a, b を x, y に代入して処理を実行。
 この x, y を仮引数（かりひきすう）と呼ぶ。
4. main()関数に戻る。
 このとき、データzの値を返す。このzをリターン値と呼ぶ。
5. リターン値を sum に代入して、main()の残りを実行。

6章 関数

C言語基礎1

(2) 関数の使用例

次に実際に上の例をプログラムで記述してみます。

```
#include <stdio.h>

double    va(double x,double y); → 関数vaのプロトタイプ宣言
                                     main()より先に記述する

int main(void)
{
    double a,b,sum;
    a = 11.11;
    b = 22.22;
    sum = va(a,b); → 関数vaを呼ぶ
    return 0;
}

double    va(double x,double y) → 関数vaの本体
{
    double z;
    z = x+y; → その和を計算する
    return z; → 答えを返す
}
```

このように、関数を使用する場合には、

1. ユーザ関数のプロトタイプ宣言を main() よりも前に記述する。この宣言は、コンパイラに関数の情報を与える。この宣言を行うとコンパイラは関数の引数と返却値のチェックを行う。
2. main()の中（ユーザ関数の中でもよい）で、ユーザ関数を呼ぶ。
3. main()に続けてユーザ関数の本体部を記述する。（ユーザ関数を別ファイルとして作成する場合がありますが、このホームページでは扱いません。）

6章 関数

(3) 各宣言の説明

プロトタイプ宣言、関数、return、void について説明します。

【プロトタイプ宣言】

コンパイラに、関数名、return型、引数の型を知らせる

```
return型 関数名(引数型と仮引数名の並び);
```

- (例) double wa(double x, double y);
- 関数名の名付け方については[第2章](#)を参照のこと。
- リターン値が無い場合は、return型はvoidとなる。
- return型を省略すると、return型は「int型」となる。
- 引数がない場合は関数名(void)とする。
- セミコロンを忘れないこと。

6章 関数

【関数】

```
return型 関数名(引数型と仮引数名の並び)
{
    関数内変数の宣言
    実行文
    return文;
}
```

- リターン値が無い場合は、return型はvoidとなる。
- 引数がない場合は関数名(void)とする。
- セミicolonは不要。

【return】

1. 値を返す必要があるとき

return 値; が一般的

return 式; 直接式でも可

2. 値を返す必要のないとき

return;

関数 演習

【練習問題①】

整数の 2 乗を計算する関数を作成しなさい。1つの int 型引数を取り、結果を戻り値として返すこと。

また、関数の動作を検証できるようにプログラムを作成しなさい。

【練習問題②】

2つの整数の平均を計算する関数を作成しなさい。2つの int 型引数を取り、結果を戻り値として返すこと。

また、関数の動作を検証できるようにプログラムを作成しなさい。

※ 計算は整数で行い、小数点以下は切り捨ててよい。

関数 演習

【練習問題③】

2つの整数の大きい方を選ぶ関数を作成しなさい。2つの int 型引数を取り、大きい方の数値を戻り値として返すこと。

また、int 型の変数 x、y、z にそれぞれ数値を入力し、作成した関数を使用して最も大きい数値を表示するプログラムを作成しなさい。

関数 演習

C言語基礎1

【練習問題④】

サイズを示す数値を引数とし、何等かの文字で例のような三角形を表示する関数を作成しなさい。

その関数を使用してサイズ 3、4、5 の3つの三角形を表示するプログラムを作成しなさい。

例：

```
$
$$
$$$

$
$$
$$$
$$$$

$
$$
$$$
$$$$
$$$$$
```

関数 演習

C言語基礎1

【練習問題⑤】

九九のひとつの段を表示する関数を作成しなさい。何の段（1～9）であるかを引数とします。

その関数を使用して、九九表を作成しなさい。

【練習問題⑥】

1文字を引数として、その文字がアルファベット小文字であれば大文字に変換して返す関数を作成しなさい。（小文字でなければそのままを返す）

その関数を使用して、入力された文字列を大文字に変換して表示しなさい。

7章 ポインタ

ポインタとは、「変数のアドレスを記憶する変数」と定義することができます。

C言語の特徴にポインタが使用できることがあげられますが、ポインタからC言語がわからなくなったという話もよく耳にします。

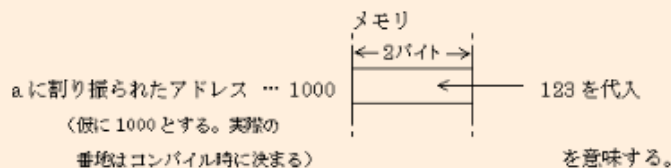
けれども、順を追ってきちんと消化していけば、ポインタは難しいものではありません。読み飛ばさず、じっくりと理解するようにしてください。

●変数とポインタ

(1) 変数とアドレス

ポインタについて理解するには「アドレス」とは何かをまず理解してください。

`int a = 123 ;` は実際には下図のように、「メモリ上のある番地（下図では1000番地）に変数a としての領域を確保し、その領域に 123 を格納する」ということになります。



この時、

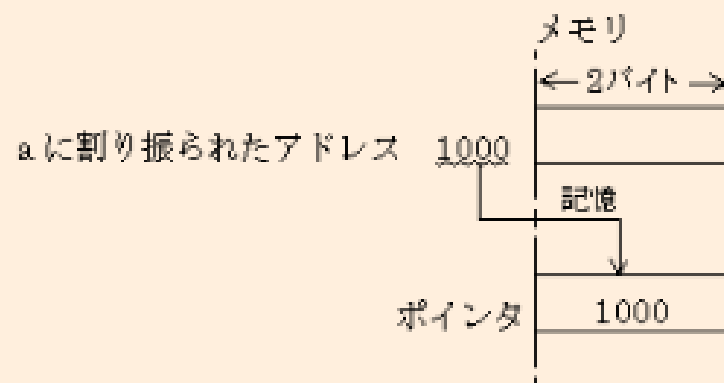
7章 ポインタ

a	変数a の値 (123) を示す
&a	変数a のアドレス (1000) を示す

と表現します。 [参考1](#)

(2) ポインタとは

ポインタとは**アドレス変数**、つまり**変数のアドレスを記憶する変数**のことです。



7章 ポインタ

ポインタは必ず、

1. 宣言
2. 値（アドレス）の設定
3. 使用

の 3ステップで用います。

ちょっと面倒ですが、下の使い方をよく見て、この 3ステップを確認してください。

間違いやすいのは「*」の使い方です。各ステップで「*」がどのように使われているかよく注意してください。

C言語基礎1

7章 ポインタ

の3ステップで用います。

ちょっと面倒ですが、下の使い方をよく見て、この3ステップを確認してください。

間違いやすいのは「*」の使い方です。各ステップで「*」がどのように使われているかよく注意してください。

```
int main( void)
{
```

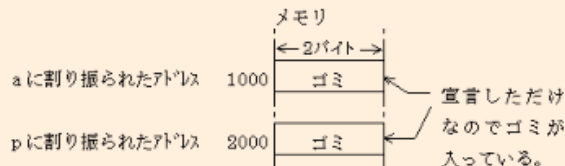
```
    int    a,b;
    int    *p;
```

```
    p = &a;
    *p = 100;
    b = *p + 1;
```

① **ポインタの宣言**：ポインタを使うと宣言する。
 (アドレスを記憶する変数の)データ型 *ポインタ名;

「*」をつけるとポインタのことになる

```
int    a,b;
int    *p; ... ポインタ p の宣言
          ( a のアドレスを記憶するので int 型)
```



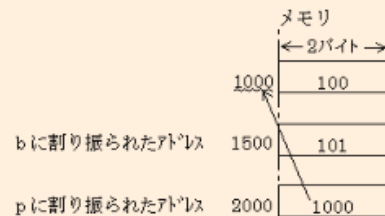
② **ポインタの値設定**：ポインタに変数のアドレスを設定
 p = &a; ... ポインタ p に変数 a のアドレスを代入
 これを「ポインタ p は a を指す」という

*が付かないことに注意



③ **ポインタを使ったデータのアクセス**：
 *p は「ポインタ p の指すアドレスの中身」のこと

```
*p = 100; ... ポインタ p の指すアドレスの中身に 100 を代入
b = *p + 1; ... ポインタ p の指すアドレスの中身 + 1 を b に代入 (100)
```



ポインタ 演習

【練習問題①】

次のコードに追加し、“12”と表示されるプログラムを完成させなさい。

ただし、変数 x に数値 12 を直接代入するのではなく、ポインタ変数 p を使用して x の値を 12 にすること。

```
int    x;  
int*   p;  
  
// xの値を12にする  
  
printf( "%d\n", x );
```

ポインタ 演習

【練習問題②】

次のコードに追加し、‘h’ と ‘w’ を大文字にすることにより、“Hello World” と表示されるプログラムを完成させなさい。

ただし、配列の添え字を使用しないこと。（str[0] = 'H';と書いてはいけない）

※ ‘h’ は 0 番目、‘w’ は 6 番目の文字であることを利用してよい。

```
char str[] = "hello world";  
  
// 'h' と 'w' を大文字にする  
  
printf( "%s¥n", str );
```

ポインタ 演習

C言語基礎1

【練習問題③】

次のプログラムを作成しなさい。

- 10 個の数値を入力する。
- 入力された順番と逆の順番で 10 個の数値を表示する。

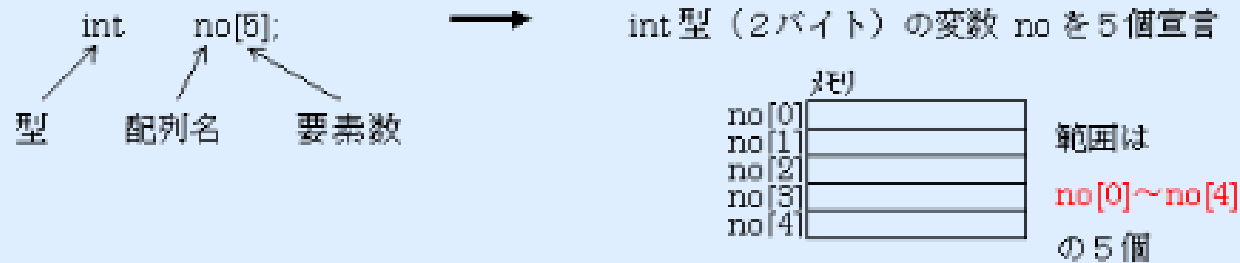
ただし、配列の添え字を使用しないこと。

8 章.配列と文字列

C言語基礎1

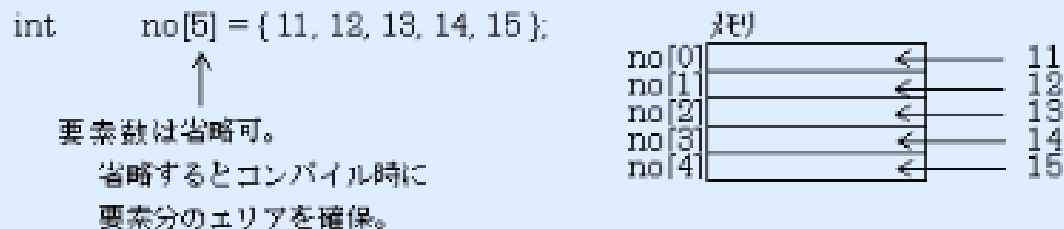
(1) 配列の宣言

変数と同様、プログラムの先頭で型と配列名、要素数の宣言を行います。



(2) 配列データの初期化

{ } を使って初期化リストを書きます。



8 章.配列と文字列

C言語基礎1

2 - 4. 文字と文字列

(1) 文字データとは

人間は「A」という文字データを表すのに 'A' と記述する
コンピュータは 'A' という記述を 65 というコードで扱う

つまり、'A' と 65 は等しいのです。この 'A' に対応する 65 というコードを「ASCIIコード」と言い、ASCIIコードを表にしたものを「ASCIIコード表」と言います。

文字コードにはたくさんの種類が存在しますが、コンピュータ用の英数字のコードとして最も広く用いられているのはASCIIコードです。このホームページでも、文字コードはASCIIコードを基に記述してあります。

(2) 文字データと文字列データ

C言語では、文字データと文字列データとは以下のように区別されます。

(文字)

メモリ上の1バイトに格納される。

' ' (一重引用符) で囲む。

```
char moji = 'A';
```

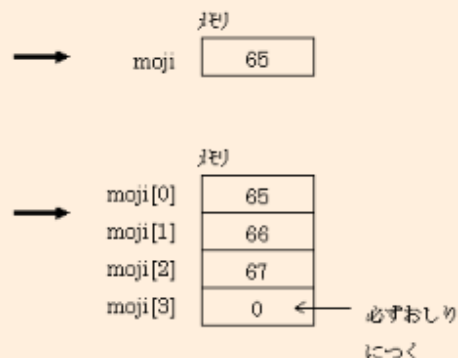
(文字列)

メモリ上の複数バイトに格納され、終了コードとして、0 が最後につく。

配列で表す。

" " (二重引用符) で囲む。

```
char moji[4] = "ABC";
```



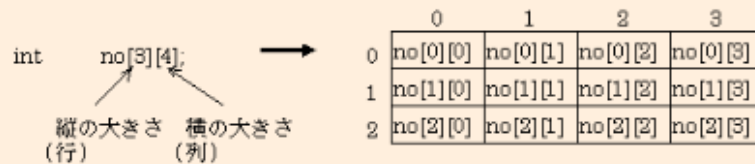
8 章.配列と文字列

C言語基礎1

2次元配列

(1) 2次元配列の宣言

プログラムの先頭で型と配列名、縦(行)と横(列)の大きさの宣言を行います。



※ 概念上は上の図ようになるが、メモリ上はこのように割当てられている。

メモリ

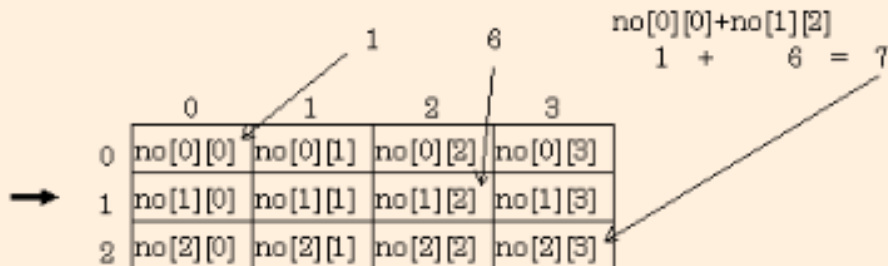
<code>no[0][0]</code>
<code>no[0][1]</code>
<code>no[0][2]</code>
<code>no[0][3]</code>
<code>no[1][0]</code>
<code>no[1][1]</code>
<code>no[1][2]</code>
<code>no[1][3]</code>
<code>no[2][0]</code>
<code>no[2][1]</code>
<code>no[2][2]</code>
<code>no[2][3]</code>

8 章.配列と文字列

C言語基礎1

配列の添字を直接指定してデータをアクセスします。

```
int    no[3][4];
no[0][0] = 1;
no[1][2] = 6;
no[2][3] = no[0][0] + no[1][2];
```



(3) 2次元配列データの初期化

{ } を使って初期化リストを書きます。

```
int    no[3][4] = { { 11, 63, 77, 94 },
                    { 61, 35, 2, 25 },
                    { 103, 66, 22, 10 } };
```

	[0]	[1]	[2]	[3]
[0]	11	63	77	94
[1]	61	35	2	25
[2]	103	66	22	10

8 章.配列と文字列

C言語基礎1

{ } を使って初期化リストを書きます。

```
int    no[3][4] = { { 11, 63, 77, 94 },
                   { 61, 35,  2, 25},
                   { 103, 66, 22, 10} };
```

→

	[0]	[1]	[2]	[3]
[0]	11	63	77	94
[1]	61	35	2	25
[2]	103	66	22	10

(4) 2次元配列で複数の文字列を設定

複数の文字列は 2次元配列で表すことができます。

```
char str[3][7] = {
    "ABC", "DEFGHI", "JK"
};
int i;
for (i = 0; i < 3; i++) {
    printf("%s\n", str[i]);
}
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
[0]:	'A'	'B'	'C'	'\0'			
[1]:	'D'	'E'	'F'	'G'	'H'	'I'	'\0'
[2]:	'J'	'K'	'\0'				

※ 2次元配列の各先頭要素のアドレスは、**配列名【添字】**で表す点に注意。
詳しくは[ポインタの章](#)で説明。

(実行結果)

```
ABC
DEFGHI
JK
```

配列と文字列 演習

C言語基礎1

【練習問題①】

次のプログラムを作成しなさい。

- 10 個の数値を入力する
- 入力された各々の数値を 2 倍にして表示する。

【練習問題②】

次のプログラムを作成しなさい。

- 10 個の数値を入力する。
- 入力された順番と逆の順番で 10 個の数値を表示する。

配列と文字列 演習

【練習問題③】

次のプログラムを作成しなさい。

- 10 個の数値を入力する。
- 入力された数値を偶数と奇数に分類して表示する。

表示例：

```
偶数：42 54 32 8  
奇数：7 35 71 13 21 45
```

※ 最初に偶数だけ、次に奇数だけを表示する。

数値を繰り返して入力し、合計が 100 を超えた場合、または入力が 10 回行われた場合、入力を止め入力された数値をすべて表示するプログラムを作成しなさい。

配列と文字列 演習

【練習問題④】

数値を繰り返して入力し、合計が 100 を超えた場合、または入力が 10 回行われた場合、入力を止め入力された数値をすべて表示するプログラムを作成しなさい。

【練習問題⑤】

次のコードに追加し、入力された数値を 16 桁の 2 進数で表示するプログラムを完成させなさい。

```
int binary[16];
int value;
int i;

scanf( "%d", &value );

// 配列 binary に 0 か 1 を代入する
for( i = 0 ; i < 16 ; i++ )
    printf( "%d", binary[i] );
```

配列と文字列 演習

C言語基礎1

【練習問題①】

char 型の変数 a、b、c に文字 'A'、'B'、'C' を代入し、3文字連続して“ABC”と表示するプログラムを作成しなさい。

【練習問題②】

char 型の変数 a に文字 'A' を代入し、その変数に演算を施し、小文字 'a' にして表示するプログラムを作成しなさい。

※ 文字コード（参考として）

'A' = 0x41 = 65

'a' = 0x61 = 97

配列と文字列 演習

【練習問題③】

列数を示す数値を入力し、‘a’～‘z’の26文字を指定された列数に従って表示するプログラムを作成しなさい。

例：列 7

```
abcdefg  
hijklmn  
opqrstu  
vwxyz
```

【練習問題④】

次のプログラムを作成しなさい。

- 文字列を入力する。
- 入力された文字列を逆順に表示する。

例：

```
apple ←入力  
elppa ←逆順に表示
```

※ 1文字ずつ後ろから表示すればよい。

配列と文字列 演習

【練習問題⑤】

次のコードに追加し、入力された数値を 16 桁の 2 進数で表示するプログラムを完成させなさい。

```
char    sbinary[17];  
int     value;  
int     i;  
  
scanf( "%d", &value );  
  
// 配列 sbinaryに'0'か'1'を代入する  
  
printf( "%s", binary );
```

9章.構造体

C言語基礎2

複数のデータをまとめて扱うには配列を用いましたが、配列では同じ型のデータしかまとめて扱う事はできません。

実際にプログラムを組んでいると、異なる型のデータをまとめて扱いたい場合がしばしばあります。たとえば、学生の成績を扱うときに、int型の学生番号と、char型配列の氏名と、double型の点数をまとめて扱えれば便利だと思いませんか。

実は、この章で学習する「構造体」は幾つかの異なる型のデータをまとめて 1つのデータ型として扱うものなのです。

● 構造体の使用手順

```
#include <stdio.h>

/* (1) 構造体の型枠の宣言 */
struct seiseki {
    int no; /* 学生番号 */
    char name[20]; /* 氏名 */
    double average; /* 平均値 */
};

int main(void)
{
    int i;
    /* (2) 構造体の宣言 */
    /* (3) 構造体の初期化 */
    struct seiseki seito1 = { 5, "KASAHARA", 83.5 };
    struct seiseki seito2[20] = {
        { 1, "SAKURAI", 78.6 },
        { 2, "NAGANO", 57.3 },
        { 3, "TAKESHITA", 66.4 },
    };

    /* (4) 構造体の参照 */
    printf("%d %s %5.1f\n", seito1.no, seito1.name, seito1.average);
    for(i = 0; i < 3; i++) {
        printf("%d %s %5.1f\n",
            seito2[i].no, seito2[i].name, seito2[i].average);
    }

    return 0;
}
```

9章.構造体

C言語基礎2

(実行結果)

```
5 KASAHARA 83.5
1 SAKURAI 78.6
2 NAGANO 57.3
3 TAKESHITA 66.4
```

(1) 構造体の型枠の宣言

まず、複数のデータ（メンバと呼びます）をまとめて1つの構造体の型枠を宣言します。
 この宣言は単に構造体の型を作ったにすぎず、領域の割当ては行われていません。

(書き方)

```
struct タグ名 {
    データ型 メンバ名;
};
```

(例)

```
struct seiseki {
    int    no;
    char   name[20];
    double average;
};
```

※ この型枠の有効範囲は宣言する場所によって異なります。
 関数外で宣言した場合 ⇒ その位置より下の全関数で有効
 関数内で宣言した場合 ⇒ 宣言した関数内でのみ有効

9章.構造体

C言語基礎2

(2) 構造体の宣言

(1) で作った「型枠」を使って実際にデータを宣言し、メモリ上に領域を確保します。

構造体は「**構造体変数**」として 1つの構造体を扱う事も出来ますし、複数の構造体をまとめて「**構造体配列**」として扱う事も出来ます。

(書き方)

struct タグ名 変数名の並び;

(例)

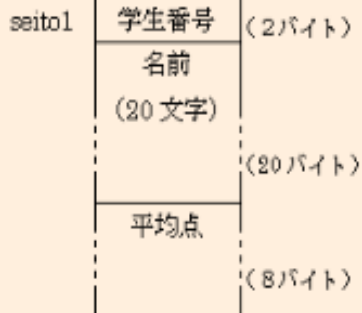
```
struct seiseki seito1;
struct seiseki seito2[20];
```

【例 1. 構造体変数の例】

seiseki という構造をもつ seito1 という「変数」を宣言

```
struct seiseki seito1;
```

メモリ →



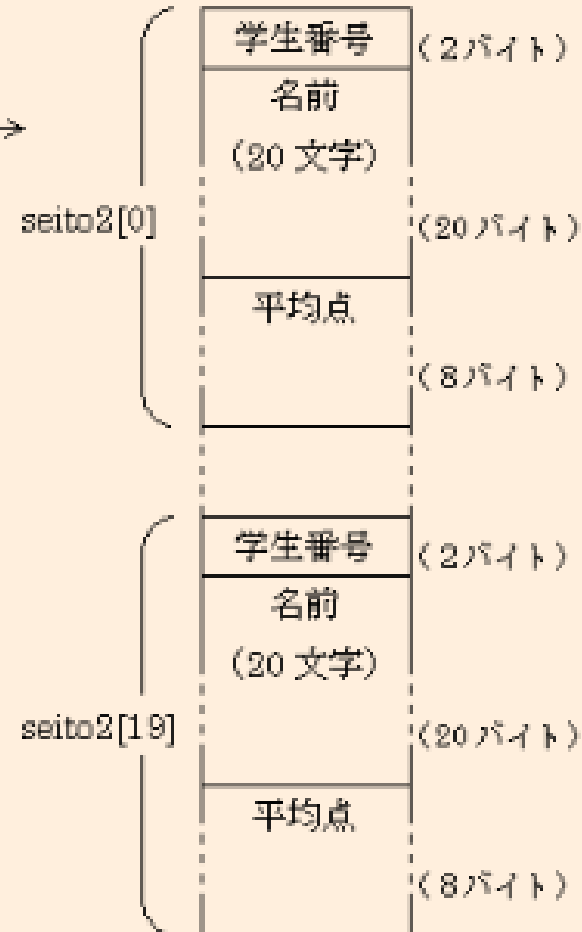
9章.構造体

C言語基礎2

seiseki という構造をもつ seito2 という「配列」を宣言

```
struct seiseki seito2[20];
```

メモリ →



9章.構造体

C言語基礎2

構造体は一般の変数や配列同様、(2)の宣言時に合わせて初期化を行う事が出来ます。

1. 構造体変数の初期化

{ } の間に、各メンバ名をカンマで区切って記述。

```
struct seiseki seito1 = { 5, "KASAHARA", 83.5 };
```

2. 構造体配列の初期化

各配列要素ごとに、{ } で区切って記述。

```
struct seiseki seito2[20] = {  
    { 1, "SAKURAI", 78.6 },  
    { 2, "NAGANO", 57.3 },  
    { 3, "TAKESHITA", 66.4 },  
};
```

9章.構造体

C言語基礎2

構造体の各メンバは、「構造体変数名.メンバ名」のようにピリオド (.) を用いて指定し、この (.) を「ドット演算子」と呼びます。

1. 構造体変数の参照

```
printf("%d %s %5.1f\n\n", seito1.no, seito1.name, seito1.average);
```

2. 構造体配列の参照

```
for(i = 0; i < 3; i++) {  
    printf("%d %s %5.1f\n",  
           seito2[i].no, seito2[i].name, seito2[i].average);  
}
```


構造体 演習

C言語基礎2

【練習問題①】

何月であるかを表す数値（1～12）と降水量(整数：mm)をメンバーとする構造体を定義し、4ヵ月分の月と降水量を入力して表示するプログラムを作成しなさい。

表示例：

```
1月    49mm
2月    60mm
3月    115mm
4月    130mm
```

構造体 演習

C言語基礎2

【練習問題②】

以下の内容のプログラムを作成しなさい。

- `int`型のメンバー `x` と `y` を持つ構造体 `POINT` を定義する。
- 構造体 `POINT` を引数とし、各メンバーの値を2倍にした `POINT` を戻り値とする関数を定義する。
- 構造体 `POINT` 型の変数の各メンバーに値を設定し、作成した関数によって値を2倍にする。
- 構造体 `POINT` 型の変数の各メンバーの値を表示する。

構造体 演習

C言語基礎2

【練習問題③】

以下の内容のプログラムを作成しなさい。

- 学生の試験結果を表す、次のメンバーを持った構造体を定義する。
 - ・ 名前 (char型配列)
 - ・ 試験の得点 (int型)
 - ・ 成績評価 (char型)
- 成績評価を設定する関数を定義する。
 - ・ 作成した構造体のポインタを引数とする。
 - ・ 試験の得点によって成績評価に文字を設定する。
80点以上→'A'、70点以上→'B'、60点以上→'C'、60点未満→'D'
- 作成した構造体の変数に、名前と試験の得点を入力する。
- 作成した関数によって、成績評価を設定する。
- 構造体の内容 (名前、試験の得点、成績評価) を表示する。

```
名前は？  
田中  
得点は？  
75  
田中、75点、成績 B
```

構造体 演習

C言語基礎2

【練習問題④】

前問で作成した構造体と関数を使用して、4人分の名前と得点を入力し、評価を設定して表示するプログラムを作成しなさい。

10章 ファイル入出力

標準出力 printf()

画面に書式付きで出力します。

printf() の f は "format" (書式) の f です。printf() は書式指定を行うことにより、同じ「65」という数値でも、10進数で出力したり、文字で出力したりというように出力形式を変えることができます。

(書き方)

```
printf( 書式文字列, 可変個引数 );
```

(1) 書式文字列

""で囲まれた文字列

【1. 変換指定文字列】

% ではじまり、出力データの変換形式を指定します。

尚、「%」を文字として表現するには「%%」と記述します。

標準出力 printf()

C言語基礎2

【printf の変換指定文字】

変換指定文字	意味	使われるデータ型
%c	1文字として出力する	文字型
%d	10進数で出力する	整数型
%x	16進数で出力する	
%o	8進数で出力する	
%f	[-]dddd.dddddddの形式で出力する	浮動小数点型
%e	指数形式で出力する	
%s	文字列として出力する	文字列

標準出力 printf()

C言語基礎2

【2. エスケープシーケンス(拡張表記)】

¥ で始まります。通常の方法で表現できない文字を ¥ を先頭に付けることにより出力することができます。

【代表的なエスケープシーケンス】

エスケープシーケンス	意味	ASCIIコード (16進)
¥n	復帰改行	0A
¥a	警報音	07
¥t	タブコード	09
¥b	バックスペース	08
¥¥	文字としての ¥	5C
¥'	文字としての '	2C
¥"	文字としての "	22
¥0	文字列終了コード	0

(例)

(例) int a = 65;

printf("data = %d %x %c¥n", a, a, a);

↑ a を 10 進出力

↑ a を 16 進出力

↑ a を文字出力



出力結果
data = 65 41 A (復帰改行)

エスケープシーケンス 例

文字コード	実行内容
¥033[0J	カーソル位置から画面右下まで消去
¥033[2J	画面クリア
¥033[nA	カーソルをn行上へ移動
¥033[nB	カーソルをn行下へ移動
¥033[1m	ハイライト(太字)
¥033[4m	下線
¥033[0m	デフォルトに戻す

```
#include <stdio.h>

int main()
{
    //画面をクリアしてカーソルを1行目1桁目に移動
    printf("\033[2J\033[1;1H");
    return (0);
}
```

文字コード	文字色変更
¥033[30m	黒
¥033[31m	赤
¥033[32m	緑
¥033[33m	黄
¥033[34m	青
¥033[35m	マゼンタ
¥033[36m	シアン
¥033[37m	白
¥033[39m	デフォルトに戻す

標準入力 scanf()

C言語基礎2

キーボードから書式付きで入力します。

scanf() の f は "format" (書式) の f です。scanf() は printf() と同様に書式指定を行うことにより、「A」のキーを押しても、16進数で入力したり、文字で入力したりというように入力形式を変えることができます。

(書き方)

```
scanf( 書式指定文字列, 格納可変個引数 );
```

(1) 書式指定文字列

"" で囲まれた文字列

【scanf の変換指定文字】

変換指定文字	意味	使われるデータ型
%c	1文字として入力する	文字型
%d	10進数で入力する	整数型
%x	16進数で入力する	
%o	8進数で入力する	
%f	浮動小数点数を入力する	浮動小数点型
%s	文字列を入力する	文字型配列

標準入力 scanf()

C言語基礎2

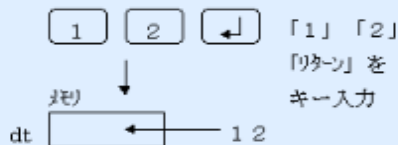
(2) 格納可変個引数

入力データを格納する領域のアドレス（アドレスについては[ポインタの章](#)で詳しく説明します。）を指定します。

【1. 変数にキーボードからデータを入力】

変数のアドレスを「&変数名」で記述します。

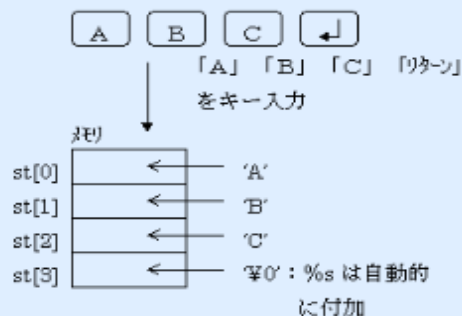
(例) int dt;
 scanf("%d",&dt);
 ↓
 アドレスを示す。忘れない
 ように注意



【2. 文字型配列にキーボードから文字列を入力】

配列の先頭要素のアドレスを「配列名」で記述します。「&」は不要です。

(例) char st[100];
 scanf("%s",st);
 ↓
 アドレスを示す。「&」は不要なことに注意



10章 ファイル入出力

テキストファイルの読み書き

ファイルの取り扱い

今までは、入出力を行う場合、すべて画面に表示してきました。
画面に表示すれば、結果がすぐにわかって便利だからです。
しかし、画面に表示された結果は、プログラムが終わると消えてしまいます。
また、膨大な量のデータを画面に表示するのは現実的ではありません。

このような場合、ファイルにデータを保存するのが普通です。
ファイルとして保存されたデータは、ディスク上に保存されるので、
消えることはなく、コピーや再編集が容易です。

ここでは、C言語でファイルを読み書きする方法を説明します。

10章 ファイル入出力

ファイルの開閉

プログラムからファイル进行操作する手順は、おおむね次のような順番で行われます。

ファイル进行操作する手順

ファイルを開く → ファイルに読み書きする → ファイルを閉じる

つまり、ファイル操作には、**ファイルの開閉**が必須となっています。

そこで、C言語には、ファイルを開閉する関数が用意されています。

開く関数が **fopen**関数、閉じる関数が **fclose**関数です。

この関数を使うには、stdio.h のインクルードが必要になります。

それぞれの関数の使い方は、次の通りになります。

ファイルを開閉する関数

```
FILE型のポインタ変数 = fopen(ファイル名, モード);  
fclose(FILE型のポインタ変数);
```

ファイル名とは、その名の通りのファイル名です。

フルパスで指定することも、名前だけで指定することもできます。

10章 ファイル入出力

ファイルの開閉

プログラムからファイルを操作する手順は、おおむね次のような順番で行われます。

ファイルを操作する手順

ファイルを開く → ファイルに読み書きする → ファイルを閉じる

つまり、ファイル操作には、**ファイルの開閉**が必須となっています。

そこで、C言語には、ファイルを開閉する関数が用意されています。

開く関数が **fopen**関数、閉じる関数が **fclose**関数です。

この関数を使うには、`stdio.h` のインクルードが必要になります。

それぞれの関数の使い方は、次の通りになります。

ファイルを開閉する関数

```
FILE型のポインタ変数 = fopen(ファイル名, モード);  
fclose(FILE型のポインタ変数);
```

ファイル名とは、その名の通りのファイル名です。

フルパスで指定することも、名前だけで指定することもできます。

10章 ファイル入出力

モード文字列	目的
r	読み込み。ファイルがない時は失敗。
r+	読み書き。ファイルがない時は失敗。
w	書き込み。ファイルがあっても空のファイルを作る。
w+	読み書き。ファイルがあっても空のファイルを作る。
a	追加書き込み。ファイルがない時は作る。
a+	追加読み書き。ファイルがない時は作る。

FILE型は今までに出てきたことがない新しい型ですが、その正体は構造体です。fopen関数を実行すると、ファイル情報を持つFILE型へのポインタが返されます。このポインタは、以後、開いたファイルの識別子として使うだけなので、ポインタ特有の操作を行ったり、構造体の要素を使うことはありません。慣習的に、FILE型へのポインタ変数を、**ファイルポインタ**と呼びます。

ここまで説明したことがわかれば、ファイルの開閉ができます。
次のプログラムは、test.txtと言う名前のファイルを書き込みのために開く例です。

10章 ファイル入出力

FILE型は今までに出てきたことがない新しい型ですが、その正体は構造体です。fopen関数を実行すると、ファイル情報を持つFILE型へのポインタが返されます。このポインタは、以後、開いたファイルの識別子として使うだけなので、ポインタ特有の操作を行ったり、構造体の要素を使うことはありません。慣習的に、FILE型へのポインタ変数を、**ファイルポインタ**と呼びます。

ここまで説明したことがわかれば、ファイルの開閉ができます。
次のプログラムは、test.txtと言う名前のファイルを書き込みのために開く例です。

ソースコード

```
#include <stdio.h>

int main(void)
{
    FILE *file;
    file = fopen("test.txt", "w");
    fclose(file);
    return 0;
}
```

このプログラムを実行すると、test.txtという名前のファイルが作成されます。
今回は開いただけなので、当然中身は空っぽです。

10章 ファイル入出力

fcloseの役割

今回のプログラムだけだと、fclose関数はとくに必要ないように見えてしまいます。

しかし、fclose関数には重要な役割がありますので、忘れてはいけません。

現代のパソコンでは、同時に複数のアプリが動作できます。
もし、同じファイルを同時に別々のアプリで書き換えてしまうと、
どちらを反映すれば良いのかわからなくなってしまいます。

そこで、fopen関数で書き込みができるように開いたファイルには、
他のソフトで書き換えられないよう、ロックしています。
fclose関数はロックを解除して、他のアプリから使えるようにするために必要です。

10章 ファイル入出力

ファイルへの書き込み

ファイルへテキストを書き込む関数は、多くの種類が用意されています。その中に、馴染みのprintf関数と良く似た、fprintf関数があります。fprintf関数の使い方は、次の通りです。

fprintf関数

```
fprintf(ファイルポインタ, 書き込み文字列, 変数・・・);
```

使い方は、ファイルポインタを指定すること以外、printf関数とまったく同じですが、指定した文字列は、画面ではなく**ファイルに書き込まれます**。
次のプログラムは、test.txtファイルに Hello,world の文字列を書き込みます。

10章 ファイル入出力

ソースコード

```
#include <stdio.h>

int main(void)
{
    FILE *file;
    file = fopen("test.txt", "w");
    fprintf(file, "Hello,world");
    fclose(file);
    return 0;
}
```

このプログラムを実行すると、test.txtファイルの中身は次のようになります。
test.txtファイルは、実行ファイルと同じフォルダに作られます。

実行後の「test.txt」

Hello,world

10章 ファイル入出力

printf関数と同じように、変数の値を書き込むこともできます。
次のプログラムは、test.txtファイルに変数iの値を書き込みます。

ソースコード

```
#include <stdio.h>

int main(void)
{
    int i = 100;
    FILE *file;
    file = fopen("test.txt", "w");
    fprintf(file, "%d", i);
    fclose(file);
    return 0;
}
```

このプログラムを実行すると、test.txtファイルの中身は次のようになります。

実行後の「test.txt」

100

10章 ファイル入出力

ファイルからの読み込み

ファイルのテキストを読み込む関数にも多くの種類がありますが、馴染みのscanf関数に似た**fscanf**関数があります。

先頭でファイルポインタを指定することを除き使い方はscanf関数と同じです。

scanf関数はキーボードから読み込むため、実行すると入力待ちになりましたが、fscanf関数では、ファイルのテキストを先頭から読み込みます。

次のプログラムは、test.txtファイルから先頭の数字を読み込んで表示します。

10章 ファイル入出力

ソースコード

```
#include <stdio.h>

int main(void)
{
    int i;
    FILE *file;
    file = fopen("test.txt", "r");
    fscanf(file, "%d", &i);
    fclose(file);
    printf("%d\n", i);
    return 0;
}
```

このプログラムの実行結果は、test.txtファイルの中身によって異なります。
test.txtファイルの中身が次の通りだった場合、実行結果は次の通りになります。

実行前の「test.txt」ファイルの中身

100

10章 ファイル入出力 (CSV)

Comma Separated Valuesの略
各項目がカンマ(,)で区切られたテキストデータ

Excelにより読み書き、編集が容易

	0	1
1	"A,お茶",20,110	←
2	"B,コーラ",20,120	←
3	"C,珈琲",20,130	←
4	"D,オレンジ",20,150	←
5	"E,紅茶",20,160	←

	A	B	C
1	A,お茶	20	110
2	B,コーラ	20	120
3	C,珈琲	20	130
4	D,オレンジ	20	150
5	E,紅茶	20	160

MS業務でのCSVファイルの取り扱い

制御装置の演算結果、状態などのアナログデータ、デジタルデータが機器の出力情報のフォーマットとしてCSVファイルを利用している。

10章 ファイル入出力 (CSV)

以下は、fscanf関数によるCSVファイルの読出方法を示す。

```
void R_ProductData() { /* Zaiko.csvを読み込む */
```

```
    FILE* fps;
```

```
    int ret;
```

```
    int i;
```

```
    fps = fopen("Zaiko.csv", "r");
```

```
    i=0;
```

```
    while(ret=fscanf(fps, "%[^,],%d,%d", shohin[i], &kosuu[i], &nedan[i]) > 0) {
```

```
        printf("%s,%d,%d¥n", shohin[i], kosuu[i], nedan[i]);
```

```
        i++;
```

```
    }
```

```
    fcnt = i;
```

```
    fclose(fps);
```

```
    return;
```

```
}
```

半角スペースを忘れない

Zaiko.csv ファイル
"A:お茶",20,110
"B:コーラ",20,120
"C:珈琲",20,130
"D:オレンジ",10,150
"E:紅茶",15,160

%sでは、カンマ区間文字列を取得できない為、%[^,]とすること。

10章 ファイル入出力 (CSV)

変数定義 (実体)

```
char shohin[N][m]          shohin[0][10]
                             shohin[1][10]
                             shohin[2][10]
                             shohin[3][10]
                             shohin[4][10]

Int kosuu[N]

Int nedan[N]
```

	0	1	2	3	4	5	6	7	8	9
A	:		お		茶	¥0				
B	:		コ		ー		ラ		¥0	
C	:		珈		琲	¥0				
D	:	オ	レ	ン	ジ	¥0				
E	:		紅		茶	¥0				

Csvファイル N行時の変数定義

m 1要素当たりのデータバイト数 例は10バイト

ポインタによるアドレス参照 (先頭アドレスの指定)

```
char *shohin_ptr[m] -> char mByteからなる変数
shohin_ptr = shohin;
```

```
Int *kosuu_ptr
kosuu_ptr = kosuu;
```

```
Int *nedan_ptr
nedan_ptr = nedan;
```

kosuu[0]	20
kosuu[1]	20
kosuu[2]	20
kosuu[3]	10
kosuu[4]	15

nedan[0]	110
nedan[1]	120
nedan[2]	130
nedan[3]	150
nedan[4]	160

10章 ファイル入出力 (CSV)

ポインタを使用して所定のデータを編集する。
指定の行数への移動は 配列Indexを用いるか、
ポインタ変数を指定のアドレスに移動させておくこと。

```
char *shohin_ptr[m] -> char mByteからなる変数  
strcpy(* shohin_ptr,"F: Water");
```

```
Int *kosuu_ptr  
*kosuu = 50;  
  
Int *nedan_ptr  
*nedan = 200;
```

Zakio.csv ファイル

```
"A:お茶",20,110  
"B:コーラ",20,120  
"C:珈琲",20,130  
"D:オレンジ",10,150  
"E:紅茶",15,160
```

10章 ファイル入出力 (CSV)

以下は、fprintf関数によるCSVファイルの書込方法を示す。

```
void W_ProductData(){ /* Zaiko.csv を書き込む*/  
    FILE* fps;  
    int ret;  
    int i;  
  
    fps = fopen("Zaiko.csv","w");  
    for(i=0; i < fcnt; i++) {  
        ret=fprintf(fps,"%s,%d,%d¥n",shohin_ptr,kosuu_ptr,nedan_ptr);  
    }  
    fclose(fps);  
    return;  
}
```

Zaiko.csv ファイル
"A:**お茶**",20,110
"B:**コーラ**",20,120
"C:**珈琲**",20,130
"D:**オレンジ**",20,150
"E:**紅茶**",20,160

ファイル入出力 (CSV) 演習

周期表.CSVファイルを使用して以下の内容をC言語で作成してください。

以下に示される周期律表を構造体を用いた配列(MAX120)を作成し、周期表データベースを作成してください。

※データをプログラム内に記述するか、CSVファイルを読み込むかは、講師指示に従ってください。(周期表.csvに周期表データは保存されています)

原子番号 : unsigned char型

元素記号 : char型配列(文字列2文字)

周期 : unsigned char型

族 : unsigned char型

原子量 : float型 下に示す表で[]の有るものは[]が付いたままのデータとなっております。

標準入力より入力した原子番号に対応する原子の元素記号、周期、族、原子量を以下の様に表示し、次の入力を待ちます。

入力: 3

出力: Li[2-1] 6.941

数字以外のものが入力されたら処理を終了してください。データベースに存在しない原子番号が入力された場合は、入力待ちを継続します。

ファイル入出力 (CSV) 演習

CSVデータ(抜粋)				
原子番号	周期	族	元素記号	原子量
1	1		1 H	1.008
2	1		1 He	4.003
3	2		1 Li	6.941
4	2		2 Be	9.012
5	2		13 B	10.81
6	2		14 C	12.01
7	2		15 N	14.01
8	2		16 O	16
9	2		17 F	19
10	2		18 Ne	20.18
11	3		1 Na	22.99
12	3		2 Mg	24.31

貯金箱

C言語基礎2

人と技術で次代を拓く

MEITEC

Engineering Firm at The Core