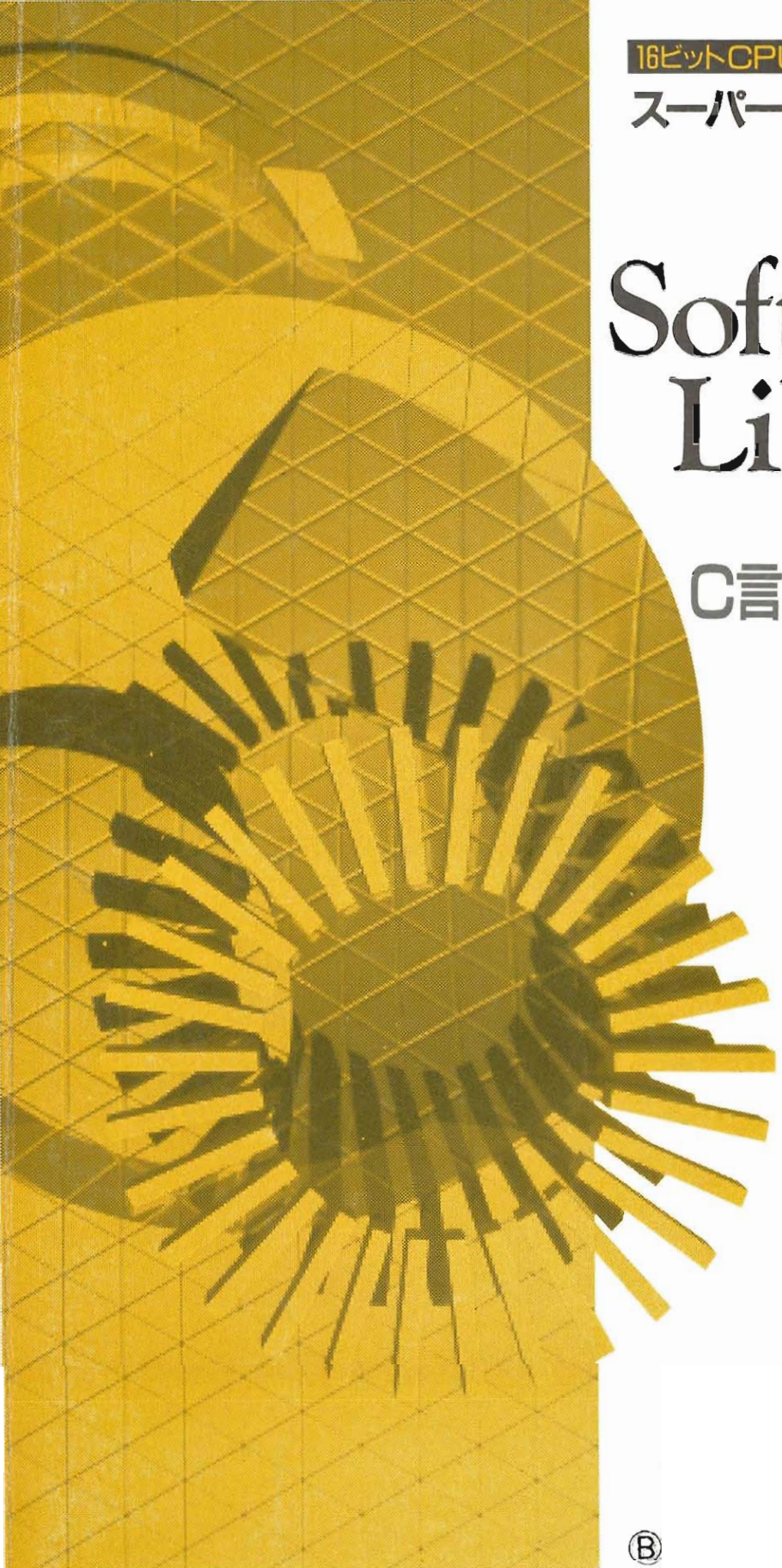


16ビットCPUポケットコンピュータ

スーパーカレッジ

Z-1

Software Library

C言語・BASIC・CASL編

CASIO

Ⓑ

- 本書の内容に関しては、将来予告なしに変更することがあります。
- 本書の内容については万全を期して作成いたしましたが、万一不審な点や誤りなどお気付きのことがありましたらご連絡ください。
- 本書使用による損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 本書の一部または全部を無断で複写することは禁止されています。また、個人としてご利用になるほかは、著作権法上、当社に無断では使用できませんのでご注意ください。
- 万一、本機使用により生じた損害、逸失利益または第三者からのいかなる請求につきましても、当社では一切その責任を負えませんので、あらかじめご了承ください。
- 故障、修理、電池交換等に起因するデータ・プログラムの消失による、損害および逸失利益等につきましては、当社では一切その責任を負えませんので、あらかじめご了承ください。
  なお、大切なデータおよびプログラムはフロッピーディスクに記録しておいたり、また控えのフロッピーディスクを作っておくようにしてください。

はじめに

工学系を目指す人の基礎力として、自分でコンピュータプログラムが作れることは大切なことです。

コンピュータ言語を学ぶときに、文法解説書や専門書を読んで学ぶ方法があります。しかし、この方法では実際にコンピュータが動いているという実感をつかめず、多くの人がプログラムを作れるようになる前に興味を失いかねません。

プログラムの学習には近道はありません。プログラムの学習には、「習うより慣れろ」という言葉がぴったりと当てはまります。コンピュータにプログラムを入力し、実行して結果を見ると、初心者にもコンピュータはこういうふうに動くということを実感できます。また、プログラムを改造して結果を得たり、自分で作ったプログラムを実行してデバッグしたりしながら学ぶという方法が、いちばん身につく学習方法なのです。

本機にはC言語、BASIC、CASL、8086系アセンブリ言語の4つのプログラム言語でプログラムを作成し、実行させることができます。

C言語は、いま最も注目されているプログラム言語です。プログラムを書くときの自由度が高い高級言語です。また、アセンブリ言語のように直接メモリーを扱うことができ、細かい処理も記述できます。科学技術計算だけでなくコンピュータによる制御のプログラムを書くのにも適しています。これは、C言語の特長のほんの一例です。将来、コンピュータを扱うさまざまな分野でC言語プログラマーが活躍すると予想されています。

BASICは、学習用のプログラム言語として、または実験や研究の道具として広く使われています。本機のBASICは、特に数値計算の分野で高速で精度の高い計算ができます。学習用としてだけでなく、将来実務についたときも仕事の中で必ず役立てることができます。

CASLは、通産省の実施する「情報処理技術者試験」のためのアセンブリ言語です。言語仕様が限定されているので初心者に理解しやすく、コンピュータの動きを学ぶのには最適なアセンブリ言語です。本機では、仮想計算機COMETとアセンブリ言語をシミュレートします。

本機で簡単なキー操作を行なうことにより、プログラムを作成して実行させることができます。教室でも、自分の机の上でも、公園のベンチでも、電車の中でもどこでも気軽に言語学習ができます。どんな言語解説書を読むよりも、プログラムを作れるようになる近道となります。いつも本機を身近に置いてご活用いただきたいと思います。

なお、8086系アセンブリ言語については『Software Libraryアセンブリ言語編』をお読みください。

目次

第3章　CASL編

メニュー操作

ここで学ぶこと
☆メインメニュー
☆各モードのメニュー
☆モードの種類
☆モードに入る方法

まとめ
◎[MENU]キーを押すと、メインメニューが表示されます。
◎メニューに表示された機能に対応する数字キーや英字キーを押すと、サブメニューに入ったり、その機能が使えるようになります。
◎[CAL]キーを押すと、CALモードに入ります。
◎[BRK]キーを押すと、CALモードに入ります。
◎[SHIFT][SUB MENU]と操作すると、直前に使用したメニューに戻ります。

CALモード	(操作：[CAL])
F. COMモード	(操作：[MENU][1])
BASICモード	(操作：[MENU][2])
C言語モード	(操作：[MENU][3])
CASLモード	(操作：[MENU][4])
アセンブリ言語モード	(操作：[MENU][5])
Fxモード	(操作：[MENU][6])
モード設定	(操作：[MENU][7])

メインメニュー

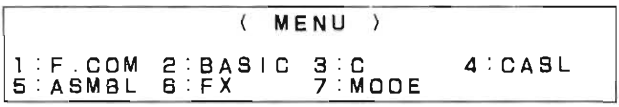
メニュー方式

本機はCALモードでマニュアル計算を行なう他に、BASIC、C言語、CASL、8086系アセンブリ言語という4種類のプログラム言語でプログラムを作成し、実行させることができます。CALモード以外の機能を利用するとき、すなわちプログラム言語を使うとき、プログラムやデータの転送や消去などのファイルの操作をするとき、本機のさまざまな設定を行なうときの操作は、メニュー方式で行ないます。

メインメニューの表示

電源をONして[MENU]キーを押すと、メニューが表示されます。これをメインメニューといいます。

(操作) [MENU]



本機の操作の基本は、このメニューからスタートします。画面に表示されたメニューの中から項目を選択することによって、いろいろな機能を使えます。

メインメニューのキー操作

メニュー項目の先頭に付けられた数字に対応するキーを押すと、その項目が選択されます。

[1]キー	1：F.COM	F.COMモードに入ります。
[2]キー	2：BASIC	BASICモードに入ります。
[3]キー	3：C	C言語モードに入ります。
[4]キー	4：CASL	CASLモードに入ります。
[5]キー	5：ASMBL	8086系アセンブリ言語モードに入ります。
[6]キー	6：FX	Fx(統計計算・ボード制御)モードに入ります。
[7]キー	7：MODE	本機の状態設定を行なうモード設定に入ります。

メインメニューで使えるその他のキー

[CAL]キー	マニュアル計算（CAL）モードに入ります。
[BRK]キー	マニュアル計算（CAL）モードに入ります。
[SHIFT][MENU]キー	直前に使ったメニューに戻ります。

各モードのメニュー

メインメニューで[1]～[7]のキーを押すと、次のメニューが表示されます。
各モードのメニューでは、次に説明する機能をメニュー形式で選択できます。
また、各画面のメニューの機能の他に、次のキーが使えます。

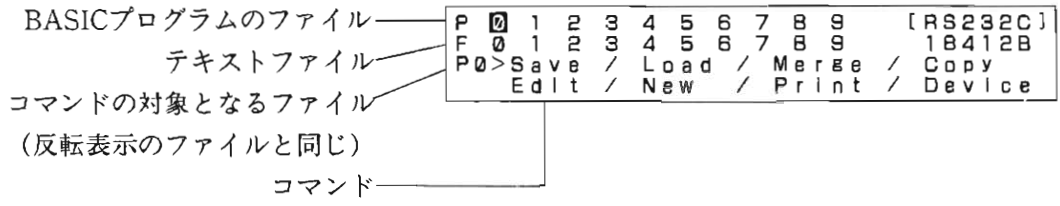
[MENU]キー	メインメニューに戻ります。
[CAL]キー	マニュアル計算（CAL）モードに入ります。
[BRK]キー	マニュアル計算（CAL）モードに入ります。

F.COMモード

ファイルの入出力と編集を行なうメニューが表示されます。このモードを、F.COMモードといいます。

画面の上2行には、ファイル名が表示されます。反転表示されているファイルがコマンドの対象となるファイルです。

(操作) [MENU][1]



F.COMメニューのキー操作

[←][→][↓][↑]	コマンドの対象となるファイルを選択します。
[S]：Save	ファイルを外部記憶装置に保存します。
[L]：Load	外部記憶装置に保存されているファイルを呼び出します。
[M]：Merge	外部記憶装置に保存されているファイルを対象ファイルにマージします。
[C]：Copy	別のファイルエリアにコピーします。
[E]：Edit	ファイルエディタでBASICプログラムやテキストファイルを編集します。
[N]：New	ファイルを消去します。
[P]：Print	接続されているプリンタに出力します。
[D]：Device	入出力デバイスの設定をします。

BASICモード

BASICプログラムの作成、実行を行なうBASICモードに入ります。

操作 MENU 2

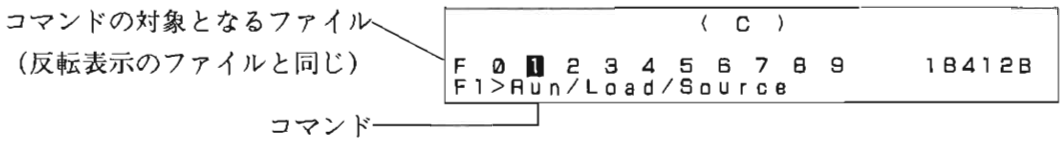
```
P 0 1 2 3 4 5 6 7 8 9      18412B
Ready P0
-
```

カーソルが表示されて、BASICマニュアルコマンドが入力できます。

C言語モード

C言語プログラムの実行、ロード、編集を行なうC言語モードに入ります。

操作 MENU 3



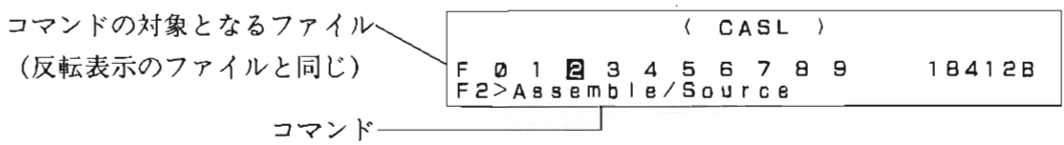
C言語メニューのキー操作

⇦⇨	コマンドの対象となるファイルを選択します。
R：Run	C言語プログラムを実行します。
L：Load	C言語プログラムをロードします。
S：Source	C言語プログラムを編集するテキストエディタに入ります。

CASLモード

CASLプログラムのアセンブルと編集を行なうCASLモードに入ります。

操作 MENU 4



CASLメニューのキー操作

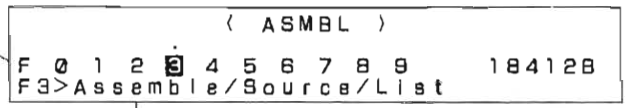
⇦⇨	コマンドの対象となるファイルを選択します。
A：Assemble	CASLソースプログラムをアセンブルします。
S：Source	CASLソースプログラムを編集するテキストエディタに入ります。

アセンブリ言語モード

8086系アセンブリ言語のソースファイルのアセンブル、編集、リストの出力を行なうアセンブリ言語モードに入ります。

(操作) MENU 5

コマンドの対象となるファイル
(反転表示のファイルと同じ)



コマンド

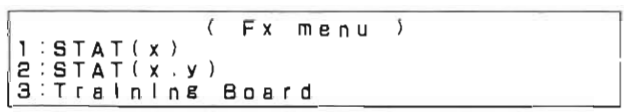
アセンブリ言語メニューのキー操作

⇦⇨　コマンドの対象となるファイルを選択します。
A：Assemble　アセンブリ言語ソースプログラムをアセンブルします。
S：Source　アセンブリ言語ソースプログラムを編集するテキストエディタに入ります。
L：List　アセンブリ言語ソースプログラムをアセンブルし、リストを出力します。出力先は、画面またはプリンタが選択できます。

Fxモード

Fxモードに入り、統計計算とオプションボード接続用のプログラムを起動できます。

(操作) MENU 6



Fxメニューのキー操作

1 1：STAT(x)　1変数統計計算を行なうFxサブメニューに入ります。
2 2：STAT(x,y)　2変数統計計算を行なうFxサブメニューに入ります。
3 3：Training Board　オプションの入出力ボードを接続した場合に、ボードとの通信プログラムを起動して、ボードの制御を行ないます。

モード設定

各種モードを設定するメニューに入ります。

(操作) MENU 7

モード設定メニューのキー操作

⇨⇦　メニュー項目間のカーソルの移動を行ないます。
⇩⇧　設定内容を切り替えます。

各メニューの設定内容については、『ガイドブック』をご覧ください。

入力モードとカーソルの形

ここで学ぶこと
☆入力モードの種類
☆カーソルの形の見分け方

まとめ
◎CAPSキーとSHIFT カナで、入力する文字や記号の種類を変えます。
◎カーソルの形によって、現在入力できる文字の種類がわかります。

キーボードは少ないキーで多くの文字や記号が入力できるように、SHIFTキーやCAPSキー、SHIFT カナで切り替えることによって、一つのキーにいくつもの機能を割り当ててあります。

本機の入力モードは、

①CALモード、BASICモードおよびモニターモード、C言語とCASLの実行モード

②テキストエディタ(C言語、CASLのソースプログラム入力時およびF.COMモードのEDIT時)

の2種類に分かれます。それぞれ、カーソルの形とINSキーの働きが異なります。

CALモード、BASICモードおよびモニターモード、C言語とCASLの実行モード

このモードでは、いつも上書きモードになります。

カーソルの形は次のようになります。

上書きモード	英字モード	▬
	カナモード	▬

このモードでSキーを押すと、Sシンボルが点灯します。

INSキーは、1文字分のスペースを挿入する機能となります。カーソルは、挿入されたスペースの位置に置かれます。

テキストエディタ(C言語モード、CASLのソースプログラム入力時およびF.COMモードのEDIT時)

このモードでは、上書きモードと挿入モードを切り替えられます。INSキーを押すごとに、上書きモードと挿入モードが切り替えられます。

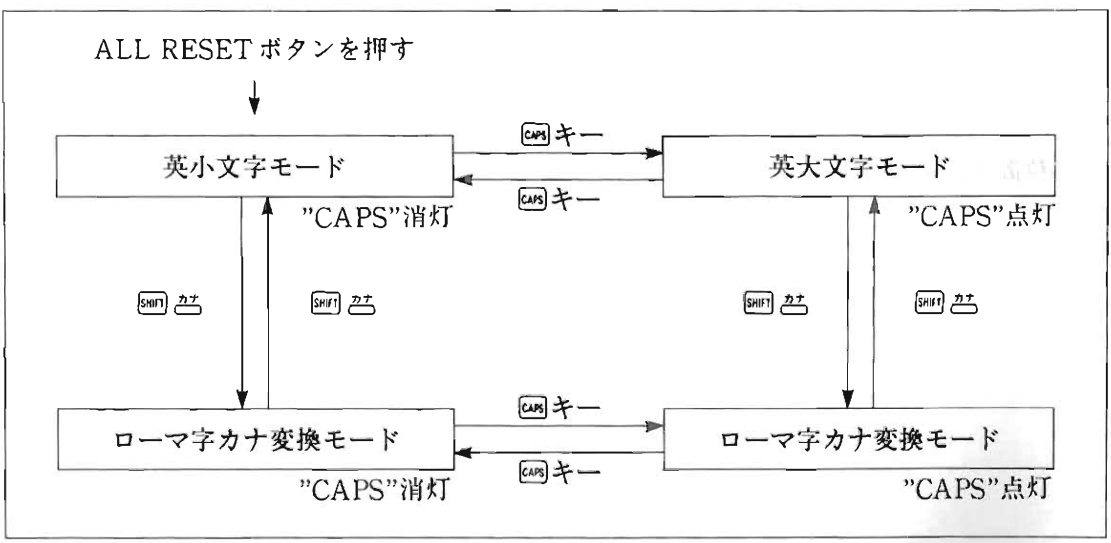
カーソルとシンボルの点灯状態は、次のような形になります。

	モード	"S" シンボル	"CAPS" シンボル	カーソル	Aキーを押したときの表示
挿入モード	英大文字	消灯	点灯	■	A
	英小文字	消灯	消灯	■	a
	シフト	点灯	点灯／消灯	■	@
	カナ	消灯	点灯／消灯	▬	ア
上書きモード	英大文字	消灯	点灯	—	A
	英小文字	消灯	消灯	—	a
	シフト	点灯	点灯／消灯	▬	@
	カナ	消灯	点灯／消灯	▬	ア

挿入モードの保持

英大文字、英小文字、カナの入力モードは、電源をOFFにしても保持されます。次に電源をONにしたときは、直前の入力モードとなります。

下に、CAPSキーとSHIFT カナによる入力モードの変化を示します。



第1章
C言語編

プログラムの入力と実行

ここで学ぶこと
☆プログラムの入力
☆プログラムの実行

まとめ
◎MENUキーを押して、メインメニューを表示させます。
◎Cキーを押して、C言語モードに入ります。
◎Sキーを押すと、プログラム入力モードに入ります。
◎Lキーを押すと、プログラムがC言語インタプリター上にロードされます。RUNと入力すると、プログラムが実行されます。
◎Rキーを押すと、プログラムがC言語インタプリター上にロードされて実行されます。

〔プログラム例1〕 あいさつのプログラム

```
main ( )
{
        printf ("コンニチハ!") ;
}
```

このプログラムは、画面に「コンニチハ!」という文字列を表示するものです。
C言語では、特別な場合を除いて小文字でプログラムを記述します。
まず、プログラムを入力する操作を行ないます。

C言語モードに入る

電源をONにします。
MENUキーを押して、メインメニューを表示させます。

操作　電源ON
　　MENU

```
                ( MENU )

1:F.COM  2:BASIC  3:C        4:CASL
5:ASMBL  6:FX     7:MODE
```

Cキーを押して、C言語モードに入ります。このメニューをC言語メニューと呼びます。

操作　C

```
                ( C )

F 0 1 2 3 4 5 6 7 8 9        184128
F1>Run/Load/Source
```

オールリセットした後にC言語モードに入ると、自動的にファイルエリアとして「F1」が指定されます。←→キーによって、ファイルエリアを選択することができます。

プログラムの入力

プログラムの入力は、ファイルエディタで行ないます。

ファイルエディタは次のように起動します。

操作 [S]

プログラムを入力します。CAPSシンボルが点灯していなくて、小文字モードとなっていることを確認してください。

操作 [M][A][I][N]['('][')'][↵]

プログラムを1行入力した後は、必ず↵キーを押してください。

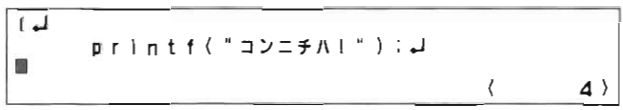
続けて、次のように入力します。

操作 [SHIFT]['{'][↵]

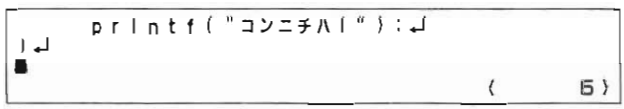
[TAB][P][R][I][N][T][F]['('][SHIFT]['"']

[SHIFT][カナ]['K']['O']['N']['N']['I']['T']['I']

['H']['A'][SHIFT][カナ][SHIFT]['-'][SHIFT]['"'][')'][':'][↵]



操作 [SHIFT]['}'][↵]



これでプログラム入力できました。プログラムは、本体内のファイルエリア「F 1」に保存されています。

プログラムのロード(Load)と実行(Run)

プログラムのロードと実行は、C言語メニューから行ないます。プログラムを呼び出すことをロードといいます。

操作　MENU 3

```
                    〈 C 〉
F 0 * 2 3 4 5 6 7 8 9          18375B
F1>Run/Load/Source
```

プログラムが記憶されているファイルエリアは、「＊」が表示されます。
ロードと実行の対象となるファイルエリアは、反転表示されているものです。

ご注意

◎他のファイルエリアに記憶されているプログラムをロードと実行の対象にしたいときは、←→キーで反転表示を移動させて、ファイルエリアを変更します。

操作　L

```
Load F1
>_
```

これで、先ほど作成してファイルエリアF1に記憶されているプログラムが、C言語インタプリター上にロードされました。プロンプト「＞」はコマンド入力待ち状態を表わしています。

操作　R U N

```
Load F1
>run_
```

RUNは、プログラムを実行するというコマンドです。上記のようにキーボードからR U Nと入力するか、ワンキーコマンドを利用してSHIFT RUNと操作します。

ご注意

◎ワンキーコマンドを使ったときは画面に大文字で「ＲＵＮ」と表示されますが、機能は同じです。

その後↵キーを押すと、コマンドが実行されます。

操作　↵

```
>run
コンニチハ！
>_
```

プログラムが実行されて、結果が画面に表示されました。

ご注意

◎コマンドには、RUNのほかにEDIT、TRON、TROFFがあります。コマンドの詳細はマニュアルコマンドリファレンス(78ページ)をご覧ください。

C言語メニューから直接プログラムを実行する方法

C言語メニューでプログラムをロードし、コマンド「RUN」によってプログラムを実行させる方法の他に、C言語メニューから「Run」を選んで直接プログラムを実行させる方法があります。

操作 MENU 3

```
                    ( C )

F 0 * 2 3 4 5 6 7 8 9      183758
F1>Run/Load/Source
```

操作 R

```
Load F1
>RUN
コンニチハ！
>_
```

このようにRキーを押すと、プログラムのロードと実行が連続して行なわれます。

プログラムエラーの編集

ここで学ぶこと
☆エラーの発見
☆プログラムの修正
☆トレースモード

まとめ
◎エラーメッセージには、エラーの発生した行番号とエラーの内容が表示されます。
◎プログラムの修正は、ファイルエディタで行ないます。
◎文字列や行の検索を行なえます。
◎トレースモードで、プログラムを１ステップずつ実行させることができます。

エラーの発生

作成したプログラムにエラーがあると、ロードした時点または実行した時点でエラーメッセージが表示され、コマンド入力待ち状態になります。

エラーメッセージは、エラーの発生したファイル番号、行番号、およびエラーの内容を表示します。

〔プログラム例２〕 エラーの発生するプログラム

```
main ( )
{
        printf ("コンニチハ!";
}
```

このプログラムには次のような誤りがあります。

３行目……「)」がない → 正しくは「printf(”コンニチハ！”);」

このプログラムを実行すると、次のようになります。

操作 MENU ⊐ R

```
Load F2
(F1-3) : Syntax error
>_
```

画面の３行に表示されたエラーメッセージは、「ファイルエリアF1の３行目にSyntax errorが発生しました」という意味です。

エラーの修正

エラーを確認して修正するために、ファイルエディタを起動します。

(操作) [E][D][I][T]

```
Load F2

(F1-3) : Syntax error
>edit_
```

(操作) [↵]

```
■    printf("コンニチハ!");↵
!↵
                                    ( 3)
```

エラーの発生した行が画面の最上部に表示されます。また、同時にカーソルがその行の先頭に表われます。

(操作) [⇨]キーを数回押して、「)」を挿入する位置にカーソルを移動します。

```
     printf("コンニチハ!"■↵
!↵
                                    ( 3)
```

(操作) [)]

```
     printf("コンニチハ!")■↵
!↵
                                    ( 3)
```

これでエラーが修正できました。では、プログラムを実行してみましょう。

(操作) [SHIFT][SUB MENU]

```
                    ( C )

F 0 ■ 2 3 4 5 6 7 8 9      183758
F1>Run/Load/Source
```

(操作) [R]

```
Load F1
>RUN
コンニチハ!
>_
```

(ご注意)

◎プログラムの修正は、C言語メニューから[S]キーを押して、プログラム入力画面を呼び出して行なうこともできます。

プログラムの修正

まず、修正したい箇所に↑↓←→キーでカーソルを移動します。修正の操作方法には、次の4通りがあります。

(操作)

①挿入する文字を入力……文字を挿入します。改行したいときは↵キーを押します。
②DEL……文字を削除します。
③INS……キーを押して上書きモードにします。正しい文字を入力します（もう一度押すと、挿入モードに戻ります)。
④SHIFT L.CAN……1行全部を削除します。

修正が終わったら、BRKキーを押します。

カーソルの移動

ファイルエディタでは、作成したプログラムの先頭から末尾まですべてをスクロールさせて見ることができます。

カーソルの移動には、↑↓←→キーを使う以外に次の方法があります。

(操作)

①SHIFT L.TOP……カーソルのある行の行頭に移動します。
②SHIFT L.END……カーソルのある行の行末に移動します。
③SHIFT F.TOP……ファイルの先頭に移動します。
④SHIFT F.END……ファイルの末尾に移動します。

プログラムの中の文字、文字列、行番号の検索

プログラムの中の文字、文字列や行番号による検索ができます。プログラムの修正に便利な機能です。ここでは、C言語のプログラムを例に説明します。

文字列「in」を検索する場合

(操作) SRCH

```
main()↵
{↵
        printf("コンニチハ!");↵
search?_                              〈      1〉
```

(操作) I N ↵
(検索したい文字列を入力)

```
main()↵
{↵
        printf("コンニチハ!");↵
                                      〈      1〉
```

(操作) NEXT(次の「in」にカーソルが移動します)

```
        printf("コンニチハ!");↵
}↵

                                      〈      3〉
```

行番号３を検索する場合

操作　SHIFT LINE

```
main()↵
{↵
        printf("コンニチハ!");↵
line?_                                  (       1)
```

操作　3 ↵

(検索したい行番号を入力)

```
■       printf("コンニチハ!");↵
1↵

                                        (       3)
```

３行目が表示されます。

プログラムをトレースする

本機には、プログラムを1ステップずつ実行させて、プログラムの動きを確認することができるトレース機能があります。

C言語メニューからプログラムをロードして、次のようにトレースモードを指定します。

操作 T R O N

```
load F1
>tron_
```

操作 ↵

```
>tron
>_
```

操作 R U N ↵

```
>run
(F1-3) printf(コンニチハ!");
Break?_
```

これから実行されるソース行が表示されます。

ここで↵キーを押すと、プログラムが実行されます。

操作 ↵

```
(F1-3) printf(コンニチハ!");
Break?_
コンニチハ!
>_
```

プログラムが終了すると、プロンプト「>」が表示されます。

トレース実行時は、以下のようなキー操作が行なえます。

キー	動　　作
↵ または T	トレースモードのままで、実行を再開します。
N	トレースモードを解除して、実行を再開します。
D	トレースモードのままで、変数モードに入ります。
BRK	トレースの実行を途中で中断します。

また、トレースモードを解除するには、プロンプト「>」に対して T R O F F と入力します。

コラム●変数モード●

変数モードとは、トレース実行により表示されているステップ、またはブレークモード指定時のプログラムに使われている変数の型とその値を表示するモードです。

このモードに入ると「var>」と表示され、以下のようなキー操作が行なえます。

- 変数名↵……変数の型とその値を表示します。
- ↵のみ……変数モードを指定する前の状態(モード)に戻ります。

文字と数値を出力するプログラム

ここで学ぶこと
☆コメントの書き方
☆main関数
☆；と｜｜
☆変数の型宣言
☆変数への代入
☆printf関数
☆￥n(改行)
☆%文字
☆データの編集形式と種類
☆代表的なエスケープシーケンス
☆データの表示

文法のまとめ

◎コメントは、プログラム中のどこにでも書けます。
◎プログラムの実行は、必ずmain関数から始まります。
◎文の終わりには「;」を付け、まとまった文(ブロック)は「{」と「}」で囲みます。
◎使用する変数は、型宣言します。
◎変数には、プログラムの中で数値などが代入できます。
◎文字や数字を表示するには、printf関数を使います。
◎「%文字」はprintf関数の書式で、データの出力位置と出力形式を指定します。
◎エスケープシーケンスは、特別な意味を持つ文字定数です。printf関数の書式で使用します。
◎改行には、「¥n」を使います。

〔プログラム例3〕 文字や数値を表示するプログラム

```
①  main()
②  {
③      char moji;
③      int  suuji;
④
④      moji='A';
⑤      printf("モジ=%c¥n", moji);
⑤      printf("スウジ=%d¥n", suuji);
②  }
```

実行例 RUN↵

```
モジ=A
スウジ=123

>_
```

プログラムの説明

①main関数

C言語のプログラムの冒頭には、必ずmain関数を書きます。main関数は主プログラムであることを示す特別な関数です。C言語のプログラムはそれ自体が関数の集まりであり、関数を定義したり、関数を組み合わせて構成します。

②プログラムの最小単位「文」

プログラムの最小単位が文です。一つの文(単文)の終わりには必ず「;」(セミコロン)を付けます。

単文がいくつか集まったものを、複文(またはブロック)といいます。複文は「{」と「}」で囲みます。

③変数の型宣言

プログラムで使用する変数は、プログラムの初めに宣言します。宣言する内容は、変数の名前とその変数の型です。

ここでは、mojiという名前の変数と、suujiという名前の変数を宣言しています。

```
char  moji;  char型（文字型）の変数宣言
int   suuji;  int型（整数型）の変数宣言
```

変数の型については、後の「簡単な計算をするプログラム」で学びます。

コラム●コメント●

プログラムのタイトルなどを、プログラム中にコメント（注釈）として書き込むことができます。コメントは、「/*」と「*/」で囲みます。

行の先頭に「/*」、間に文字や記号、最後に「*/」を書いて終了すると、その全体がコメント行になります。

コメントはプログラムの実行に影響しません。コメント中には、何を書いてもエラーになりません。

次の例のように、コメントは関数を書いた行にも書くことができます。

```
printf("モ シ゛ =%c¥n", moji) ; /* moji ヲ ヒョウシ゛ スル */
```

また、複数の行にまたがるコメントを書くこともできます。

```
/*********************************/
/*          ｹﾞｰﾑ ﾉ ﾌﾟﾛｸﾞﾗﾑ          */
/*          (C)KOTAROU, February 1992          */
/*********************************/
```

ただし、「/*/* コメント */*/」のようにコメントの中にコメントは書けません。

④データの代入

変数間のデータの代入には式を使います。式の右辺の内容が、式の左辺の変数に格納されます。

ここでは、変数mojiにAが、変数suujiに123が代入されています。なお、「A」の前後に付けられた「'」と「'」(シングルクォーテーション)は、「A」が1文字の文字定数であることを示しています。

文字や数値を出力する関数

⑤printf関数

⑤の2行は、いずれもprintf関数による文です。printf関数はデータを表示させる基本的な関数で、C言語では最もよく使われる関数の一つです。

printfに続く( )の中身を引数といいます。引数が複数の場合は「,」(カンマ)で区切ります。この場合、引数は二つあります。二つの引数の意味は、次のようになっています。

```
printf("モ ジ゛ =%c¥n", moji);
        書式          変数
printf("スウジ゛ =%d¥n", suuji);
        書式          変数
```

printf関数は、上の書式で指定した型に変数の部分のデータを変換して出力します。

書式の部分に文字列を書くと、そのとおりに表示されます。もちろん文字列を書かなければ表示されません。ここでは「モジ=」と「スウジ=」の部分が表示されます。

変数の内容を表示したい場所に「%文字」を書きます。「%d」は整数(10進数)を表示するとき、「%c」は1文字を表示するときに書きます。主な%文字を次に示します。

%文字	機　　能
%d	整数を10進数として出力
%x	整数を16進数として出力
%o	整数を8進数として出力
%c	整数を文字コードとして1文字を出力
%f	浮動小数点数を出力
%s	文字列を出力

エスケープシーケンス

エスケープシーケンスとは特別な文字定数で、printf関数の書式で使用します。主に改行やブザーを鳴らすときなどに使用します。よく使うエスケープシーケンスを次に示します（その他のエスケープシーケンスについては、「文法のまとめ」をご覧ください）。

表記	呼び方（略名）	意　　味
¥0	Null	ヌルコード（値は0）
¥a	Bell(BEL)	ブザーを鳴らす
¥t	Horizontal Tabulation(HT)	水平タブ
¥f	Form Feed	改ページ
¥n	New Line(NL)	改行（＋復帰）

コラム●printf関数で複数の表示を行なう●

二つ以上の変数の内容を表示するときには、次のようにします。

```
printf("モ シ〝 =%c  スウシ〝 =%d¥n", moji, suuji);
```

①複数の変数の内容を表示する位置を指定します。
②表示位置に対応する変数名を指定します。
③最初の「%c」の位置にはmojiの内容を、次の「%d」の位置にはsuujiの内容を表示します。

文字と数値を入力するプログラム

ここで学ぶこと
☆scanf関数
☆整数定数
☆浮動小数点定数
☆文字定数
☆文字列定数

文法のまとめ
◎scanf関数は、文字や数値データを入力します。
◎数値定数は、整数定数と浮動小数点定数があります。
◎整数定数は、10進定数、8進定数、16進定数があります。
◎文字には、文字定数と文字列定数があります。

〔プログラム例4〕 文字や数値を入力するプログラム

```
main()
{
        char moji;
        int  suuji;

        printf("1モジ゛ ヲ ニュウリョク シテ クダ゛サイ=");
①       scanf("%c", &moji);
        printf("スウジ゛ ヲ ニュウリョク シテ クダ゛サイ=");
②       scanf("%d", &suuji);
        printf("1モジ゛ =%c¥n", moji);
        printf("スウジ゛ =%d¥n", suuji);
}
```

実行例 RUN↵A↵5↵

```
1モジ゛=a
スウジ゛=5

>_
```

文字や数値を入力する関数

①scanf関数——文字の入力

scanf関数には、データを読み込む機能があります。次のような引数を記述します。

```
scanf("%c", &moji);
      書式   変数
```

書式：%文字は入力するデータの形式を指定します。「%c」は1文字を入力するときに用います。

変数：読み込んだデータを代入する変数の名前を書きます。また、単純変数の場合は、変数名の前に「&」を付けます。

プログラムの①の「&moji」のように代入変数が単純変数の場合は、変数の前に「&」を付けます。「&」はメモリーの番地(アドレス)を表わす記号で、「&moji」は変数mojiの格納されるメモリーの番地(アドレス)を表わします。このように、scanf関数では、データを格納する変数はその変数の番地(アドレス)で指定します。

書式の「%c」は1文字を入力することを示します。入力されたデータが「1文字としてのデータである」として扱います。すなわち、「1」と入力しても文字として1が入力されたことになり、数値の1ではないと判断されます。

②scanf関数——数値の入力

プログラムの②では数値を入力します。入力された数値データを格納する変数は、int型、long型、float型、double型などの数値を表わす型で宣言しておきます。ここでは、int型を宣言しています。

「%d」は、数値データを入力することを示す書式です。int型に対応しています。

変数suujiの前には、格納されるメモリーの番地を示す「&」を付けます。

ここで入力されるデータは、int型(整数型)としての数値となります。たとえば、「1.23」と入力しても、整数分の「1」だけが認識されて、小数部分は無視されます。

数値の種類

11ページのプログラム例の「suuji=123;」の「123」のようなデータのことを定数といいます。定数にはいくつかの種類があります。「123」は整数定数の10進定数です。C言語では、このほかに8進数(8進定数)や16進数(16進定数)を扱うことができます。

また、小数点数(浮動小数点)も扱えます。

	整数の変数を指定する例	int suu1, suu2, suu3;
	実数の変数を指定する例	float suuf;
③	整数12を10進数で与える例	int suu1=12;
④	整数12を8進数で与える例	int suu2=014;
⑤	整数12を16進数で与える例	int suu3=0xc;
	実数3.1416を10進数で与える例	float suuf=3.1416;

③10進定数

整数の数値を10進数で与えます。特に指定しないと、10進数になります。

④8進定数

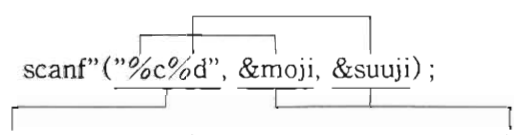
整数の数値を8進数で与えます。0(ゼロ)から始まる数値は、8進数になります。

⑤16進定数

整数の数値を16進数で与えます。0x(ゼロエックス)から始まる数値は、16進数になります。

コラム●複数のデータの入力●

二つ以上の変数を入力するときは、次の例のようにします。



複数のデータを入力する
%文字を指定します。

%文字に対応する変数名を指定します。
入力されたデータを%文字に応じて変換し、指定された変数へ代入します。「%c」が&mojiに、「%d」が&suujiに対応しています。

文字の種類

数値の他に、文字も定数の一種として扱えます。文字を扱うには、次のようにします。

	1文字の変数を指定する例 文字列の変数を指定する例	char ichimoji; char mojiretu[4];
⑦	1文字「A」を与える例	char ichimoji='A'
⑧	文字列「ABC」を与える例1 (ライブラリー関数のstrcpyを使う)	char mojiretu[4]; strcpy(mojiretu, "ABC");
	文字列「ABC」を与える例2 (1文字ずつに分解し、文字定数で与える)	char mojiretu[4]; mojiretu[0]='A'; mojiretu[1]='B'; mojiretu[2]='C'; mojiretu[3]='¥0';
	文字列「ABC」を与える例3 (直接文字列定数で与える)	char mojiretu[4]="ABC";

⑦文字定数

文字を与えます。データを「'」(シングルクォーテーション)で囲むと、1文字の文字定数になります。

⑧文字列定数

文字列を与えます。データを「"」(ダブルクォーテーション)で囲むと、文字列定数になります。

文字列の最後には、データの終わりを表わす「¥0」(NULL)が自動的に付加されます。

mojiretu	0	1	2	3
	'A'	'B'	'C'	'¥0'

定数の種類

	種　類	形　式	例
数値タイプ	整数定数	10進定数	int型　12, −12, 32767 long型　123456789L, 123l
		8進定数	int型　0, 14, 0777 long型　012345L, 0123l
		16進定数	int型　0xb, 0xf4, 0xFFFF long型　0x7FFFFFFFL, 0Xabcl
	実数定数	浮動小数点定数	3.1416, −12.0, 1.0e−5, 5.555E10
文字タイプ	文字定数	一般の文字定数	'A', 'a', '1', '0'
		エスケープシーケンス	'¥n'(改行), '¥a'(ブザー) '¥¥'(¥という文字) '¥''('という文字) '¥"'("という文字)
	文字列定数		"コンニチハ", "ABCDE", "abcde", "123456" "1モジ ニュウリョク シテ シダサイ"

〔プログラム例5〕 いろいろなタイプを扱うプログラム

```
main ( )
{
        int     suui;
        float   suuf;
        char    mojiretu [4] ;

        printf ("セイスウチ ヲ ニュウリョク シテクダ' サイ=") ;
⑨       scanf ("%d", &suui) ;
⑩       printf ("スウチ=%d (10シン)  %o (8シン)  %x (16シン) ¥n", suui,
suui, suui) ;
        printf ("スウチ ヲ ニュウリョク シテクダ' サイ=") ;
        scanf ("%f", &suuf) ;
        printf ("スウチ=%f (フト' ウショウスウテン) ¥n", suuf) ;
        printf ("モジ' レツ (3モジ' )  ヲ ニュウリョク シテクダ' サイ=") ;
⑩       scanf ("%s", mojiretu) ;
⑩       printf ("モジ' レツ=%s¥n", mojiretu) ;
}
```

注1) 下線部は、1行として入力してください。

注2) 配列は配列名自体がメモリーの番地(アドレス)を表わすので、「&」は不要です。

注3) 文字列の最後には、データの終わりを表わす「¥0」(NULL)が自動的に付加されます。

(実行例) RUN↵78↵

```
>run
セイスウチ ヲ ニュウリョク シテクダサイ=78
スウチ=78(10シン) 116(8シン) 4e(16シン)
スウチ ヲ ニュウリョク シテクダサイ=_
```

123.45↵

```
スウチ=78(10シン) 116(8シン) 4e(16シン)
スウチ ヲ ニュウリョク シテクダサイ=123.45
スウチ=123.450000(フトウショウスウテン)
モジレツ(3モジ) ヲ ニュウリョク シテクダサイ=_
```

ABC↵

```
モジレツ=abc

>_
```

⑨10進から8進、16進への変換

入力した数値を、%文字の指定する形に変換して出力します。ここでは、「suuji」という変数の内容を10進、8進、16進の三つの表現で出力します。

「%o」は8進、「%x」は16進になります(「%X」とすると、大文字の16進指定になります)。

⑩文字列の入出力

文字列は、配列という変数を使ってメモリーに記憶されます。配列については、後で解説します。

簡単な計算をするプログラム

ここで学ぶこと
☆変数名の付け方
☆変数の型宣言
☆データ型の種類
☆算術演算子

まとめ
◎int型は16ビットの整数
◎代入演算子で変数に数値を代入します。
◎long型は32ビットの整数。
◎float型は32ビットの実数(浮動小数点)。
◎double型は64ビットの実数(倍精度浮動小数点)。

〔プログラム例6〕 文字や数値を表示するプログラム

```
main ( )
{
①       int  suu1=6, suu2=4, suu3, suu4;

②       suu3=suu1+suu2;
②       suu4=suu1/2;

③       printf ("suu1+suu2=%d¥n", suu3) ;
③       printf ("suu1/2    =%d¥n", suu4) ;
}
```

実行例 RUN↵

```
suu1+suu2=10
suu1/2    =3

>_
```

変数名

C言語では、プログラムで使用する変数名をプログラムの最初に宣言します。
また、変数の型も同時に宣言します。
変数名は、次の規則の範囲で自由に付けられます。

変数名の規則

- 英字で始まる文字と数字を組み合わせてください。
- _(下線)は、英字として使用できます。
- 長さに制限はありませんが、先頭から8文字だけが有効です。
- キーワードと同じ変数名は使えません。
- ライブラリー関数名と同じ変数名は使えません。

- キーワードは、「文法のまとめ」の「キーワード」のページで説明します。
- 普通、変数名は小文字を使います。

変数の型宣言

変数の型には、代表として次のようなものがあります。必ずどれかを宣言しなければなりません。

データ型	扱 う デ ー タ	ビット数
char	文字	8
int, long	整数	16、32
float	浮動小数点	32
double	倍精度浮動小数点	64

①int型の宣言と変数の初期化

int型は、整数を扱う変数の型です。int型変数の扱える数値は、次の範囲です。

−32768 ～ 32767

二つ以上の変数名を続けて宣言することもできます。このときは、「,」(カンマ)で区切って書きます。

変数は、宣言中に初期化できます。宣言文の変数に「＝」を付け、その次に定数を書くと、変数にその定数が初期値として代入されます。

コラム●signed、unsignedとconst●

C言語の整数型（char型、int(short)型、long型）には、符号付き整数型と符号なし整数型の2種類があります。

符号付き整数型はsigned修飾子を使い、たとえばsigned intのような型で宣言します。ただし、一般にはsignedは省略されます。

符号なし整数型はunsigned修飾子を使い、たとえばunsigned intのような型で宣言します。

符号なし整数型には符号がないのでメモリーサイズは同じですが、扱える値の範囲は異なります。

例）signed　int型：−32768～32767
　　unsigned int型：0～65535

したがって、必要に応じてsigned型とunsigned型を使い分ければ、効率の良いプログラムを組むことができます(単にsignedやunsignedのみで型宣言を行なった場合は、それぞれsigned int、unsigned intと見なされます)。

const型指定子を使うと、オブジェクトを変更不可能として宣言できます。constキーワードは、整数型のchar型、int(short)型の修飾子として使えます。ある変数がconst型指定子のみで宣言されている場合は、その型はconst intと見なされます。

```
例）main( )
    {
        const   a=20, b=30;
        a=b;                    ←ここでエラーが発生します。
                                  a=20は変更できません。
```

算術演算子と代入演算子

計算に使用する演算子を、算術演算子といいます。算術演算子には次の種類があります。

演算子	意　味	例注1)	演算内容
＋	加算	suu3＝suu1＋suu2;注2)	10＝ 6 ＋ 4
－	減算	suu3＝suu1－suu2;	2 ＝ 6 － 4
＊	乗算	suu3＝suu1＊suu2;	24＝ 6 ＊ 4
／	除算注3)	suu3＝suu1／suu2;	1 ＝ 6 ／ 4
％	剰余注4)	suu3＝suu1％suu2;	2 ＝ 6 ％ 4

注1）suu1、suu2、suu3はint型で、suu1＝6、suu2＝4とします。

注2）例で使用している「＝」は、代入演算子の一種です。

注3）整数と整数同士の除算（割り算）では、結果の小数部分が切り捨てられます。

注4）剰余(余り)を求めるときには、「％」を使います。「％文字」と同じ記号ですが、意味はまったく違いますので気を付けてください。

②加算・除算と代入演算子

上の表の算術演算子を、プログラムの中に書いて計算させることができます。ここでは、加算と除算の算術演算子を使いました。

算術演算の結果を、代入演算子「＝」を使って代入しています。

③計算結果の表示

suu1＝6、suu2＝4ですから、計算結果は次のようになります。

```
suu3＝suu1＋suu2：10＝ 6 ＋ 4
suu4＝suu1／2　　：3 ＝ 6 ／ 2
```

〔プログラム例7〕 大きな数、小さな数

```
main ( )
{
        long    sul1, sul2, sul3;
④       float   suf1, suf2, suf3;
⑤       double  sud1, sud2, sud3;
⑥
        sul1=30000 , sul2=100500;
        suf1=1.23  , suf2=3.1416;
        sud1=123E10, sud2=123E12;

        sul3=sul1+sul2;
        suf3=suf1+suf2;
        sud3=sud1+sud2;

⑦       printf ("long    :%ld," , sul3) ;
⑦       printf ("float   :%f," , suf3) ;
⑦       printf ("double  :%f¥n", sud3) ;
}
```

実行例 RUN↵

```
long    :130500.float   :4.371600.
double  :12423000000000000.000000

>_
```

大きな数、小さな数

④long型の宣言

long型は、int型と同じように整数を扱う変数の型です。ただしlong型は、int型より大きい数値を扱えます。範囲は、次のようになります。

−2147483648 ～ 2147483647

⑤float型の宣言

float型は、実数(浮動小数点数)を扱う変数の型です。数値に小数桁が必要なときに使います。float型変数の扱える数値は、次のとおりです。

−9.99999e+63 ～ −1e−63, 0, +1e−63 ～ +9.99999e+63

⑥double型の宣言

double型は、float型と同じように実数を扱う変数の型です。ただしdouble型は、float型より大きい数値を扱えます。範囲は、次のとおりです。

−9.999999999e+99 ～ −1e−99, 0, +1e−99 ～ +9.999999999e+99

⑦結果出力

printf関数を使って、sul3、suf3、sud3に代入された数値を出力しています。それぞれの数値の型がわかるように、long、float、doubleの文字列も同時に出力するようにしてあります。

コラム●printf関数の出力形式(出力桁指定)●

```
main ( )
{
      int    sua=700, sub=250;
      long   sul;
      float  suf;

      sul=sua+sub;
      suf=sul/2. 0;

      printf ("sua=%5d, sub=%5d¥n", sua, sub) ;
      printf ("sul=%8 ld, suf=%8. 2f¥n", sul, suf) ;
}
```

注1)「suf=sul/2.0;」のように演算式に実数を書くと、実数の演算結果が得られます。

注2)「%5d」のように%文字を書くと、数値が編集されて出力されます（この例では5桁です)。

表示　sua=␣␣700, sub=␣␣250　　(␣は空白の意味です)

注3)「%8.2f」のように%文字を書くと、数値が編集されて出力されます。小数点には「.」(ピリオド)を使い、小数点も1桁と数えます。

表示　suf=␣␣␣␣␣950, suf=␣␣475.00　　(␣は空白の意味です)

条件分岐を行なうプログラム

ここで学ぶこと
☆ifによる分岐
☆比較演算子
☆論理演算子

文法のまとめ
◎if文が成立すると次に書かれた文が、成立しないとelseの次に書かれた文が実行されます。
◎比較演算子は、式の真偽によって0(偽)または0以外の値(真)を返します。
◎論理演算子は、AND、OR、NOTの論理演算を行ないます。

〔プログラム例8〕 偶数、奇数を判定するプログラム

```
main()
{
        int suuji, hantei;

        printf("ｽｳｼﾞ ヲ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ=");
        scanf("%d", &suuji);
        hantei=suuji%2;
①       if(hantei!=0) {
②           printf("%d ﾊ ｷｽｳ ﾃﾞｽ¥n", suuji);
        }
        else {
③           printf("%d ﾊ ｸﾞｳｽｳ ﾃﾞｽ¥n", suuji);
        }
}
```

実行例 RUN↵12↵

```
ｽｳｼﾞ ヲ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ=120
120 ﾊ ｸﾞｳｽｳ ﾃﾞｽ

>_
```

RUN↵121↵

```
ｽｳｼﾞ ヲ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ=121
121 ﾊ ｷｽｳ ﾃﾞｽ

>_
```

if文による分岐

①if文による条件の判定

変数hanteiの内容が0(ゼロ)以外の数値のときは式が成立し、0(ゼロ)のときは式が成立しません。

②式が成立するとき

式が成立するときには、このprintf関数だけが実行されます。

③式が成立しないとき

式が成立しないときには、このprintf関数だけが実行されます。

if文のまとめ

```
if(式) {
    文1;
}
else {
    文2;
}
```

- 式の値が0以外（真＝成立する）ならば、文1を実行します。
- 式の値が0（偽＝成立しない）ならば、文2を実行します。
- else文は省略できます。
- 文1、文2が単文の場合には、{} は省略できます。
  左ページのプログラムではわかりやすくするために、{}を記述してあります。

コラム●if文のもうひとつの書き方●

上記のif文を、次のように書くこともできます。

```
if (hantei)
    printf ("%d ハ キスウ デ゛ス¥n", suuji) ;
else
    printf ("%d ハ グ゛ウスウ デ゛ス¥n", suuji) ;
```

比較演算子

このプログラムでは「if(hantei != 0)」のように、等価比較を行なっています。if文には、次のような演算子が使えます。

比較演算子	意　味
i＞j	iがjより大なら真、そうでないなら偽
i＜j	iがjより小なら真、そうでないなら偽
i＞＝j	iがjより大または等しいなら真、そうでないなら偽
i＜＝j	iがjより小または等しいなら真、そうでないなら偽
i＝＝j	iとjが等しいなら真、そうでないなら偽
i！＝j	iとjが等しくないなら真、そうでないなら偽

コラム●字下げ●

if文などの{}は、文のおよぶ範囲が一目で分かるように {をifのiと同じ桁に書き、{}内の文1、文2などをifよりも少し下げて書きます。これを「字下げ」といいます。

論理演算子

「偶数、奇数を判定するプログラム」を次のように書き換えると、「suujiは2の倍数でも5の倍数でもない」という条件になります。

〔プログラム例9〕 倍数を判定するプログラム

```
main( )
{
        int  suuji, hantei1, hantei2;

        printf ("ｽｳｼﾞ  ヲ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ=") ;
        scanf ("%d", &suuji) ;
        hantei1=suuji%2;
        hantei2=suuji%5;
        if (hantei1!=0  && hantei2!=0) {
                printf ("%d ﾊ 2 ﾉ ﾊﾞｲｽｳ ﾃﾞﾓ 5 ﾉ ﾊﾞｲｽｳ ﾃﾞﾓ ｱ
ﾘﾏｾﾝ¥n", suuji) ;
        }
        else {
                printf ("%d ﾊ 2 ﾉ ﾊﾞｲｽｳ ｶ 5 ﾉ ﾊﾞｲｽｳ ﾃﾞｽ¥n",
suuji) ;
        }
}
```

注) 下線部は、1行として入力してください。

実行例 RUN↵4↵

```
ｽｳｼﾞ  ヲ ﾆｭｳﾘｮｸ ｼﾃ ｸﾀﾞｻｲ=4
4 ﾊ 2 ﾉ ﾊﾞｲｽｳ ｶ 5 ﾉ ﾊﾞｲｽｳ ﾃﾞｽ

>_
```

RUN↵11↵

```
11 ﾊ 2 ﾉ ﾊﾞｲｽｳ ﾃﾞﾓ 5 ﾉ ﾊﾞｲｽｳ ﾃﾞﾓ
 ｱﾘﾏｾﾝ

>_
```

if(hantei1！=0 && hantei2！=0) は、「hantei1が0でない」かつ「hantei2が0でない」という二つの比較演算の両方の結果がともに真であるときに成立する論理演算子「&&」を使用した条件です。「&&」(論理積)は、両方のオペランドが真(0でない) のときに真となります。

論理演算子には、次の3種類があります。

論理演算子	意　　味
&&(論理積)	両方のオペランドがともに真(0でない)のときに、真となります。
¦¦(論理和)	両方のオペランドのどちらか一方または両方が真(0でない)のときに、真となります。
!(否定)	オペランドが偽(0)のときに真(0でない)を、オペランドが真 (0でない) のときは偽(0)を返します。

※&&、¦¦ ……2項演算子です。
　! ……単項演算子です。

論理積(AND)〈2項演算子〉

記号	第1オペランド	第2オペランド	結　果
&&	真	真	真
	真	偽	偽
	偽	真	偽
	偽	偽	偽

論理和(OR)〈2項演算子〉

記号	第1オペランド	第2オペランド	結　果
||	真	真	真
	真	偽	真
	偽	真	真
	偽	偽	偽

否定(NOT)〈単項演算子〉

記号	オペランド	結　果
!	真	偽
	偽	真

コラム●演算子の優先順位●

よく使う演算子を次に示します。演算子には、実行するときの優先順位があります。その他の演算子については「文法のまとめ」を参照してください。

		優先順位
算術演算子	* / + −	1
比較演算子	< <= > >=	2
比較演算子	== !=	3
論理演算子	&& || !	4
代入演算子	+= = etc	5

繰り返し処理のプログラム

ここで学ぶこと
☆while文による繰り返し
☆for文による繰り返し
☆インクリメント演算子、デクリメント演算子

まとめ
◎while文は条件式が真の間、文を実行します。
◎for文は条件式が真の間、文を実行します。条件は初期値で与え、増分式で更新します。
◎インクリメント演算子、デクリメント演算子は、変数を＋１または－１します。

いろいろな繰り返し

〔プログラム例10〕 和を求めるプログラム

```
main ( )
{
      int  suu1=−1, suu2, suu3;

①     while (suu1 !=0) {
            suu3=0;
            printf ("1 カラ 200 マデ ノ スウ ヲ ニュウリョクシテ クダ サイ=
");
            scanf ("%d", &suu1) ;
            printf ("1 カラ %d マデ ノ スウ ノ ワ ヲ モトメマス¥n", suu
1) ;
            printf ("0ナラ オワリマス¥n") ;
②③          for (suu2=1 ; suu2<=suu1 ; suu2++) {
                  suu3=suu3+suu2 ;
            }
            printf ("%d¥n", suu3) ;
      }
}
```

注）下線部は、1行として入力してください。

実行例　RUN↵55↵

```
0ナラ オワリマス
1540
1 カラ 200 マデ ノ スウ ヲ ニュウリョクシテ クダ サ
イ=_
```

0↵

```
0ナラ オワリマス
0

>_
```

①while文

繰り返し処理を指定するには、while文を使います。式で与える条件が成立している間、｛｝内の処理を繰り返します。while文の範囲は、｛｝で指定します。

②for文

繰り返し処理を指定するもう一つの方法は、for文を使います。式で与える条件が成立している間、｛｝内の処理を繰り返します。for文の範囲は、｛｝で指定します。ただし文が単文の場合は、｛｝を省略できます。ここでは省略可能です。

③インクリメント演算子

インクリメント演算子「＋＋」を使った「suu2＋＋」は、「suu2＝suu2＋1」の演算子をします。

while文のまとめ

while文は指定した条件が成立する間、処理を繰り返し実行します。

```
while(条件式) {
    文;
    ⋮
}
```

繰り返し処理を行なう条件を、式で指定します。条件が真（成立する）ならば、{ } の中の処理を繰り返します。

while文の範囲は、{ } で指定します。ただし文が単文の場合は、{ } を省略できます。

プログラム例のfor文をwhile文に書き換えると、次のようになります。

```
suu2=1;                          ループカウンターの初期化
while(suu2<=suu1) {              繰り返し判断条件が真の間ループを繰り返す
    suu3=suu3+suu2;
    suu2++                       ループカウンターに1を加える
}
```

コラム●do～while文●

繰り返し処理は、この他に次のような種類があります。

```
do {
    文;
    ⋮
} while(条件式);
```

この文では、条件式の判定は文を実行した後に行ないます(必ず1回は文が実行されます)。

コラム●無限ループ●

while文を無条件に繰り返す方法（無限ループ)。

```
while(1) {
    ⋮
}
```

無限ループから脱出する方法は、break文、return文、exit関数、abort関数を使います。

forによる繰り返し

for文は指定した条件が成立する間、処理を繰り返し実行します。

```
for(初期値;条件式;増分式){
        ⋮
}
```

繰り返し処理を行なう初期値、条件式、増分式を指定します。条件が真(成立する)ならば、{} の中の処理を繰り返します。

for文の範囲は、{} で指定します。ただし文が単文の場合は、{}を省略できます。

コラム●for文の変形●

for文は、2重3重のネスト（入れ子）にできます(while文、do～while文も可能)。

```
for(i=1;i<=3;i++){
    for(j=i+1;j<=3;j++){
        ⋮
      n=i*j
        ⋮
    }
}
```

カンマ「,」で区切ることで、初期値や増分式を複数個指定できます。

```
for(i=1,j=10;i<j;i++,j--){
      ⋮
}
```

for文を無条件に繰り返す(無限ループ)には、初期値、条件式、増分式に何も書きません（空文とします)。

```
for(;;){
      ⋮
}
```

無限ループから脱出する方法は、break文、return文、exit関数、abort関数を使います。

インクリメント演算子とデクリメント演算子

「＋＋」をインクリメント演算子といいます。インクリメントは増分という意味です。

```
i＝i＋1;
```

と同じ動作をします。

「－－」をデクリメント演算子といいます。デクリメントは減分という意味です。

```
i＝i－1;
```

と同じ動作をします。

プログラム例10の③の「suu2＋＋」の場合は、suu2＝suu2＋1と同じ意味になります。

コラム●「j＝＋＋i」と「j＝i＋＋」●

「j＝＋＋i」と「j＝i＋＋」では、実行結果が異なります。i＝3の場合で考えてみましょう。

「j＝＋＋i」という式では、iに1を加えてi＝4としてから、jにiを代入します。したがって、j＝4となります。

j＝i＋＋という式では、iに1を加える前にjにiを代入します。したがって、j＝3となります。

iの値はいずれも4となります。

また、if文などでは次のようになります。

```
if(＋＋i＜3){
    ⋮
}
```

i＝i＋1の後、i＜3の判定を行なう

インクリメント算算子、デクリメント演算子を他の演算子と一緒に書くときには、演算子を置く位置に気を付けなければなりません。

配列とポインタ

ここで学ぶこと
☆配列の宣言
☆配列へのデータの代入
☆ポインタ変数の宣言
☆＊(アスタリスク)演算子の使い方

まとめ
◎配列を宣言すると、メモリーに配列の領域が確保されます。
◎配列にはデータが代入できます。
◎「型名　＊変数名」でポインタ変数を宣言します。
◎ポインタ変数には、アドレスが格納されます。
◎＊演算子でアドレスの内容(データ)を参照します。

〔プログラム例11〕　2つの数字を入れ替えるプログラム

```
main()
{
        char  c;
①       int   hairetu[2];
②       int   *a0, *a1;
        int   a;

        printf("ｽｳｼﾞ　ヲ　2ﾂ ﾆｭｳﾘｮｸ　ｼﾃｸﾀﾞｻｲ=");
③       scanf("%d%d", &hairetu[0], &hairetu[1]);
④       printf("ｲﾚｶｴﾏｴ　[0]=%d, [1]=%d¥n", hairetu[0], hairetu[1]);
⑤       a0=&hairetu[0];
⑥       a1=&hairetu[1];
⑦       a=*a0;
⑧       *a0=*a1;
⑨       *a1=a;
④       printf("ｲﾚｶｴｺﾞ　[0]=%d, [1]=%d¥n", hairetu[0], hairetu[1]);
}
```

注）下線部は、1行として入力してください。

実行例　RUN↵10↵

```
>run
ｽｳｼﾞ　ヲ　2ﾂ　ﾆｭｳﾘｮｸ　ｼﾃｸﾀﾞｻｲ=10
```

15↵

```
ｲﾚｶｴﾏｴ　[0]=10, [1]=15
ｲﾚｶｴｺﾞ　[0]=15, [1]=10

>_
```

プログラムの説明

①で配列を宣言します。配列の後の[ ]の中は正の整数で、配列の要素の数を示します。

②でポインタ変数を宣言します。ポインタ変数は、変数名の先頭に＊(アスタリスク)を付けて宣言します。

③はscanf関数で、配列にデータを入力します。

④はprintf関数で、入れ替えの処理の前と後の配列を表示します。

⑤~⑨では、ポインタ変数を使って配列のデータを入れ替えます。配列は［添字］によって参照します。添字の値は、0(ゼロ)と正の整数です。

ポインタ変数は、アドレス（番地）を格納する変数です。

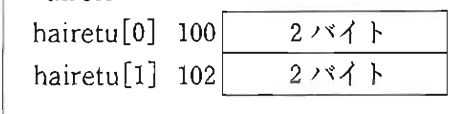
配列の宣言

配列の宣言は、配列の型、配列名、配列の要素の数の順に行ないます。

```
int         hairetu[2];
配列の型   配列名 配列の要素の数
```

上記のint型の配列を例に、配列の宣言で確保される領域を図に示してみます。

宣言した配列の先頭であるhairetu[0]のアドレスを、仮に100番地とします。



int型の配列は、int型の変数を［ ］で指定された要素の数だけ確保します。

配列へのデータの代入

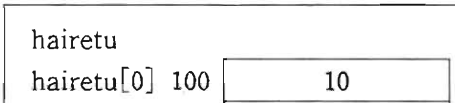
プログラムでは、scanf関数で配列にデータを入力しています。

```
scanf("%d %d", &hairetu[0], &hairetu[1]);
```

この例では、1 ␣ ↵ 2 ␣ ↵と操作します。数字は↵キーで区切って入力します。

最初に入力された10がhairetu[0]に、次に入力された20がhairetu[1]に入ります。なお、配列の先頭は0から始まります。

データが配列に入力された様子を図にしてみます。



hairetu[0]は100番地で内容は10です。

hairetu[1]は102番地で内容は20です。

③のscanf関数では、入力する変数名に「&」を付けます。「&」はアドレスを表わすポインタ演算子で、scanf関数に変数の先頭アドレスを与えています。

④のprintf関数は、配列を表示します。配列の要素は、添字によって参照します。

配列名［添字］

添字には、式や変数や数字を指定できます。そのときの式の値や数字は、０または正の整数になるようにします。添字は０(ゼロ)から始まり、配列の要素の数－１までです。

ポインタ変数の宣言と＊演算子

②では、ポインタ変数を宣言しています。

型名　　＊変数名

ポインタ変数は、変数名の先頭に＊（アスタリスク）を付けて宣言します。

プログラム例のa1とa2は、int型のポインタ変数として宣言されます。

主なポインタ変数の種類

char　＊変数名;	ポインタ変数をchar型として宣言する
int　＊変数名;	ポインタ変数をint型として宣言する
float　＊変数名;	ポインタ変数をfloat型として宣言する

⑤⑥では、ポインタ変数にアドレスを格納します。この例では、

hairetu[0]は100番地ですから、a0には100が、

hairetu[1]は102番地ですから、a1には102が、

入ります。

＊演算子は、ポインタ変数に格納されたアドレスの内容を示します。

⑦⑧⑨では、＊演算子を使って配列のデータの入れ替えを行なっています。

入れ替えには、作業領域として変数ａを使用します。

100番地の内容は10ですから、＊a0には10が、

102番地の内容は20ですから、＊a1には20が、

入ります。次の図を参照してください(ポインタ変数のアドレスは、仮にa0が200番地、a1が202番地とします)。

アドレス		内容
hairetu[0]	100	10
hairetu[1]	102	20

アドレス		内容
a0	200	100
a1	202	102

ポインタ変数の指す番地の内容	
＊a0	10
＊a1	20

(ご注意)

◎ポインタや配列を扱うとき、確保した容量を超えてデータの書き込みを行なったり、ポインタによる不正なアドレスへの書き込みを行なうと、予期しない動作をします。絶対に行なわないようにしてください。

コラム●文字列と配列●

文字列を扱うときにも、配列を使用します。
配列名はアドレスを指すので、「&」は付けません。

```
        ⋮
char mojiretu[10];
        ⋮
scanf("%s", mojiretu);
        ⋮
```

配列名はアドレスを指します。

関数定義

ここで学ぶこと
☆関数定義の方法
☆データの受け渡し
☆関数の型宣言

まとめ
◎定義した関数は、main関数から呼び出して使います。
◎呼び出し元の実引数の値が、呼び出し先の仮引数の値になります。
◎仮引数は呼び出し先で処理されて、結果はreturn文で呼び出し元へ返されます。

〔プログラム例12〕 小文字を入力すると、大文字に変換して画面に表示するプログラム

```
main ( )
{
        char moji;

        printf ("ｺﾓｼﾞ ヲ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ=") ;
        scanf ("%c", &moji) ;
①       printf ("%c¥n", henkan (moji)) ;
}
②   henkan (moji1)
③   char moji1;
{
        char moji2;

        if (moji1>96 && moji1<123)
             moji2=moji1-32;
        else
             moji2=moji1;
④       return (moji2) ;
}
```

注）本機が使用している文字コード「ASCIIコード」では、
大文字のコード＝小文字のコード－32
となっています。詳しくは、『ガイドブック』105ページの「キャラクターコード一覧表」を参照してください。

実行例　RUN↵Y↵

```
ｺﾓｼﾞ ヲ ﾆｭｳﾘｮｸ ｼﾃｸﾀﾞｻｲ=y
Y

>_
```

プログラムの解説

main関数は主プログラムを表わす特別な関数で、プログラム中に一つだけ書けます。

henkan( ) は、ここで定義している関数です。変数の名前と同じように名前を付けることができます。

①では、②で定義しているhenkan関数を呼び出します。( ) の中に変数名を書いて、呼び出す関数にデータを引き渡します。

②では、呼び出されるhenkan関数を定義します。( ) の中に書いた変数名で呼び出し元のデータを受け取ります。

③は、呼び出し元から渡されたデータを受け取る変数を宣言しています。

④では、return文により関数の呼び出し元へ戻ります。( ) の中に変数名を書くと、処理の結果を呼び出し元へ返すことができます。

関数定義の方法

関数は次のように定義します。

```
[型]  関数名 ( [引数] )
      [引数の型宣言]
      {
            文;
            ⋮
       [return( );]
      }
```

- 引数がない場合は、[引数] の項目は省略できます。
- [型]を省略すると、int型になります。
- 引数を使用しないときは、[引数の型宣言] を省略します。
- 結果を返さないときは、[return( );] を省略できます。また、「return moji2」のようにreturn文の( )を省略することもできます。

関数とデータの受け渡し

```
[呼び出し元関数]
  printf(…, henkan(moji) );
                ↓
          moji を moji1に代入する。
[呼び出し先関数]
henkan(moji1)
{
      ⋮
}
```

関数の呼び出し元で指定した引数を、実引数といいます。実引数の値は、呼び出し先の関数の仮引数の値になります。

仮引数は、呼び出す関数名の後で型宣言します。呼び出す関数内で使用される変数宣言となります。{ } の中で宣言すると、仮引数と見なされません。

```
[呼び出し元関数]
    printf(…, henkan(moji) );
                ↑
            henkan関数の値として、moji2
            が呼び出し元に返される。
[呼び出し先関数]
    ⋮
    return(moji2);
}
```

return文は、関数の呼び出し元へ戻る文です(プログラム例の④)。関数での処理結果を返すときは、

```
return(式);
```

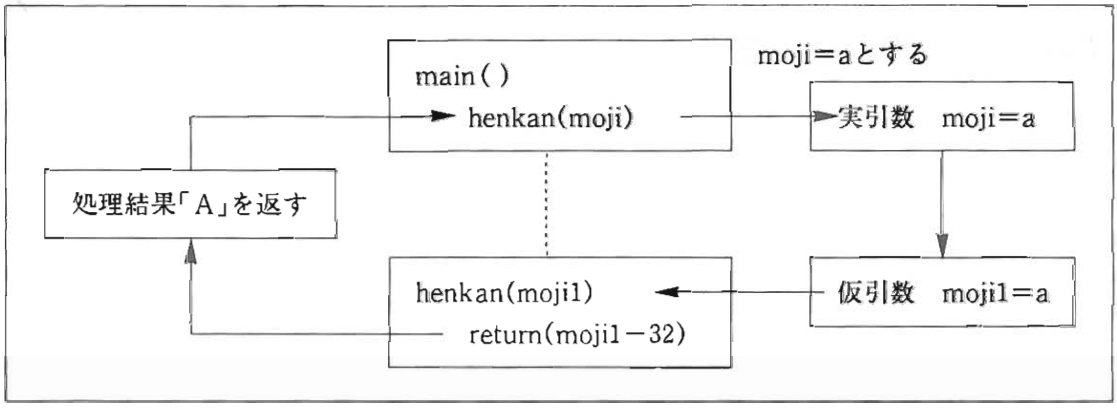
と書きます。

処理結果を返さないときは、

```
return;
```

と書きます。一般には、return文自体を省略します。

このように引数によってデータを渡し結果を返す方法を、「Call by Value」(値による呼び出し)といいます。



〔プログラム例13〕 入力された数を2乗して、表示するプログラム

```
main( )
{
①       float jijou( );
②       float suuf;
        int   suui;

        printf("スウジ ヲ ニュウリョク シテクダ サイ=");
        scanf("%d", &suui);
        suuf=jijou(suui);
③       printf("%d^2=%f¥n", suui, suuf);
}
④   float jijou(suusi)
        int suusi;
{
②       float suusf;

⑤       suusf=(float)suusi*suusi;
⑥       return suusf;
}
```

実行例 RUN↵150↵

```
スウジ ヲ ニュウリョク シテクダ サイ=150
150^2=22500.000000

>_
```

関数の型宣言

①で呼び出すjijyou関数を、float型に宣言します。

②のように、変数も大きなデータが格納できるようにfloat型に宣言します。

③の%文字には、float型用の%fを使います。

④で呼び出されるjijyou関数を、float型で宣言します。

⑤では、float型で計算するため、式の右辺もfloat型にします。

ここではfloat型の例を挙げて説明をしましたが、long型、double型の関数や変数で「Call by Value」するときには、floatをlongまたはdoubleに書き替えます。

記憶クラス

ここで学ぶこと
☆内部変数と外部変数

まとめ
◎外部変数を宣言します。
◎値を返さない関数を宣言します。
◎内部変数を宣言します。
◎シンボル定数を定義します。
◎ローカルな静的変数を宣言します。

〔プログラム例14〕 内部変数と外部変数の定義

```
①  int x=1;
   main()
   {
        void ex1(), ex2();
        x=2;
        printf("main() ノ ヘンスウ x=%d¥n", x);
        ex1();
        printf("ex1()  ｼﾞｯｺｳｺﾞ ノ   x=%d¥n", x);
        ex2();
        printf("ex2()  ｼﾞｯｺｳｺﾞ ノ   x=%d¥n", x);
        getchar();
   }

①  int y;
②  void ex1()
   {
        char x;
        x='A';
        printf("ex1()  ノ ﾅｲﾌﾞ ヘンスウ x=%c¥n", x);
        y=3;
        printf("ex1()  ノ ヘンスウ y=%d¥n", y);
        getchar();
   }

②  void ex2()
   {
①       extern int x;
        float y;
        y=4.56;
        printf("ex2()  ノ ヘンスウ x=%d¥n", x);
        printf("ex2()  ノ ﾅｲﾌﾞ ヘンスウ y=%6.2f¥n", y);
        getchar();
   }
```

実行例　R U N ↵

```
main()  ノ ヘンスウ          x=2
ex1()   ノ ナイフ゛ヘンスウ   x=A
ex1()   ノ ヘンスウ          y=3
_
```

↵

```
ex1()   シ゛ッコウコ゛ノ      x=2
ex2()   ノ ヘンスウ          x=2
ex2()   ノ ナイフ゛ヘンスウ   y=   4.56
_
```

↵

```
ex2()   ノ ナイフ゛ヘンスウ   y=   4.56

ex2()   シ゛ッコウコ゛ノ      x=2
_
```

↵

```
ex2()   シ゛ッコウコ゛ノ      x=2


>_
```

プログラムの説明

①外部変数(大域変数)

main関数や作成した関数の外で宣言する関数を、外部変数といいます。外部変数は、プログラム中のすべての関数で参照できる変数です。

extern宣言は、外部変数として宣言された変数xが参照できるようにするものです。このような変数を、大域変数(グローバル変数)といいます。

大域変数と局所変数は、お互い干渉しません。

②void宣言

プログラムで定義したex1関数、ex2関数は、main関数に実行結果を返しません。
このような関数は、

```
void ex1( ), ex2( );
```

のようにvoid宣言をして、値を返さない関数であることを宣言します。

③内部変数(局所変数)

main関数や作成した関数の中で宣言する関数を、内部変数といいます。ここでは、ex1関数のみでchar型の変数xは有効となります。このような変数を、局所変数(ローカル変数)といいます。

main関数やex2関数では、参照することはできません。

〔プログラム例15〕　5個の数値の和と平均を求めるプログラム

⑤
```
#define MAX 5
int cnt;

main ( )
{
      void keyincmp ( ) ;
      printf ("%dコノ スウシ' ヲ ケイサンシマス¥n", MAX) ;
      printf ("スウシ' ヲ ニュウリョク シテクダ' サイ¥n") ;
      for (cnt=0;cnt<MAX;cnt++)
           keyincmp ( ) ;
}

void keyincmp ( )
{
⑥     static float sum;
             float num, ver;

      printf ("¥n%dカイメ =", cnt+1) ;
      scanf ("%f", &num) ;
      sum += num;
      ver = sum/(float) (cnt+1) ;
      printf ("コ' ウケイ = %6.2f ヘイキン = %6.2f", sum, ver) ;
}
```

(実行例)　R U N ↵

```
5コノ スウシ"ヲ ケイサンシマス
スウシ"ヲ ニュウリョク シテクダ"サイ

1カイメ   =_
```

1 0 ↵ 2 0 ↵ 3 0 ↵
4 0 ↵ 5 0 ↵

```
コ"ウケイ  =  100.00  ヘイキン  =   25.00
5 カイメ =50
コ"ウケイ  =  150.00  ヘイキン  =   30.00
>_
```

プログラムの説明

⑤置換マクロ

置換マクロとは、「#define」というプリプロセッサ擬似命令によって定義されるものです。

置換マクロは、特定の文字列を参照するために使われます。この例では、プログラム中に現れる「MAX」という文字列をシンボル定数として定義し、それを数値の5で置き換えます。

⑥静的変数

staticと宣言した静的変数は、大域変数と同様にプログラムの実行開始時に0で初期化されます。関数の中で記憶クラスstaticを使用すると、その変数は関数が呼び出されるたびに初期化されることはなく、恒久的な変数とすることができます。

この例で、staticの記述を取り去って実行させると、変数sumは実行のたびに初期化されるので、加算ができません。

内部変数の定義

```
[auto]   型   変数名1，変数名2……；   …(a)
static   型   変数名1，変数名2……；   …(b)
```

(a)自動変数

変数を宣言した関数でのみ有効です。他の関数から変数の参照や変更ができないという意味です。

関数が呼び出されたときに、変数領域をメモリー上に確保し、関数の実行が終了したときに、その領域は解放されます。

なお、autoという記述は省略できます。すなわち関数内で何も書かないと、自動変数になるということです。また、autoは内部変数にのみ指定できます。外部変数には指定できません。

(b)静的変数

関数内で宣言した場合は、他の関数から変数の参照や変更はできません。

静的変数は、メモリーの静的（スタティック）領域に作られます。また、関数の外で宣言した場合は、「extern」宣言と同様に外部変数になります。プログラム中のすべての関数で参照できます。

外部変数の定義

```
[extern] 型   変数名1，変数名2，……；
static   型   変数名1，変数名2，……；
```

外部変数は、プログラム中のすべての関数に対して有効です。すべての関数で、この変数を参照したり変更したりできます。したがって、「Call by Value」によるデータの受け渡しは不要です。

コラム●外部変数の使い方●

外部変数はプログラム中のすべての関数で有効な変数ですので一見便利そうですが、乱用すると関数の独立性がそこなわれてしまいます。つまり、プログラムミスで気付かないうちに変数のデータを変えてしまうからです。

「Call by Value」を使用して変数を受け渡す方法に慣れたほうが、よいプログラムを書くことができます。

外部変数は、主に関数間のデータの受け渡しが多い場合や、変数の数が多くて引数欄に書ききれない場合などに、引数の数を減らす目的で使用されます。

複雑な条件分岐を行なうプログラム

ここで学ぶこと
☆高度な制御構造
☆switch文による条件分岐

まとめ
◎switch文とcaseラベルは、セットで使われます。
◎条件式を評価し、その値と一致した定数を持つcaseラベルから実行します。
◎break文は、switch文からの脱出に使います。
◎指定した定数がcaseラベルにない場合は、defaultラベルを実行します。

〔プログラム例16〕 データの種類を分類するプログラム

```
main ( )
{
        int  dig=0, spa=0, oth=0;
        int  keyin;

        printf ("ナニカ ニュウリョク シテクダ゛サイ¥n") ;
        printf ("SHIFT+リターンキーヲ オスト シュウリョウシマス¥n") ;
        while ((keyin=getchar ( )) !=EOF)
①           switch (keyin) {
②                 case '0':
                  case '1':
                  case '2':
                  case '3':
                  case '4':
                  case '5':
                  case '6':
                  case '7':
                  case '8':
                  case '9': dig++;
③                           break;
                  case ' ': spa++;
                            break;
④                 default: oth++;
            }
        printf ("¥nフ゛ンルイケッカ¥n") ;
        printf ("スウジ゛  %d, クウハク %d, ソノタ %d¥n", dig, spa, oth)
;
}
```

注）下線部は、1行として入力してください。

TABキーを押すと、それによって挿入されたスペースの数(8個)が空白として数えられます。

実行例　RUN↵

```
>run
ナニカ ニュウリョク シテクダ"サイ
SHIFT+リターンキーヲ オスト シュウリョウシマス
_
```

ABCD↵CAPSEFGH
IJ0123↵456
SHIFT#SHIFT!789↵

```
abcd
EFGHIJ0123
456#!789
_
```

SHIFT↵
(SHIFTキーを押しながら↵キーを押す)

```
フ"ンルイケッカ
スウシ"  10.クウハク  0.ソノタ  15

>_
```

プログラムの説明

①switch文

switch文は一つの条件式を持ち、その結果をcaseラベルで指定した定数と比較します。

②caseラベル

caseラベルは、条件式の値と定数が一致したときにプログラムを実行する場所を指定します。

③break文

break文は、switch文から抜け出す機能を持っています。

この例でbreak文がないと、数字のデータを入力した時点で、dig、spa、othのすべての変数に1が加算されてしまいます（すべてのcaseラベルとdefaultラベルが実行されます）。

④defaultラベル

caseラベルのどれにも当てはまらないときに、実行する場所を指定します。

switch文の制御構造

```
switch(条件式){
case 定数1： 文;
             ⋮
           break;
case 定数2： 文;
             ⋮
           break;
     ⋮
case 定数n： 文;
             ⋮
           break;
default：文;
}
```

グラフィックスのプログラム

ここで学ぶこと

☆仮想スクリーン上のドットの点灯・消灯

☆仮想スクリーン上の直線描画

☆仮想スクリーン上の直線消去

まとめ

◎グラフィックス関連の関数では、仮想スクリーン上のドットを扱います。

◎グラフィックス関連の関数の引数は、仮想スクリーンのドットの座標です。

◎仮想スクリーンは、192×64ドットで構成されています。

◎座標は、(x1、y1、x2、y2)で表現します。

◎座標の範囲は、次のとおりです。

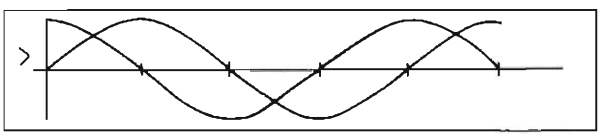
$0 \leqq x1 \leqq 191$、$0 \leqq y1 \leqq 63$

$0 \leqq x2 \leqq 191$、$0 \leqq y2 \leqq 63$

〔プログラム例17〕 グラフィックス機能を使ったプログラム

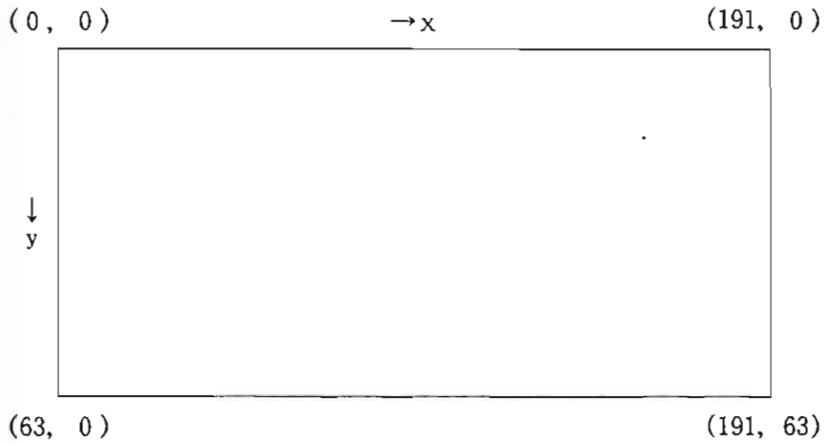
```
main()
{
        int x, ys, yc;

        angle(0);
①       clrscr();
②       line(10, 0, 10, 31);
②       line(10, 15, 180, 15);
        for(x=30;x<180;x+=30)
②           line(x+10, 13, x+10, 17);
        for(x=0;x<150;x++) {
            ys=15.5-15.0*sin(3.0*x);
            yc=15.5-15.0*cos(3.0*x);
②           line(x+10, ys, x+10, ys);
②           line(x+10, yc, x+10, yc);
        }
}
```

実行例　RUN↵



仮想スクリーン

本機のグラフィック機能では、192×64ドットの仮想スクリーンを扱えます。
座標は、仮想スクリーン上の左上隅を原点（0，0）とし、右下隅（191、63）の範囲です。



プログラムの説明

①clrscr関数

表示画面の表示を消去して、カーソルをホームポジションに移動します。この関数は何も戻り値を返しません。

②line関数

line(x1,y1,x2,y2)と記述し、始点(x1,y1)と終点(x2,y2)を結ぶ直線を描きます。

グラフィック関数には、この他にも次の2種類が用意されています。

getpixel関数…getpixel (x, y) と記述し、仮想スクリーン上のドットの点灯・消灯を調べる関数です。引数の範囲はxが0～191、yが0～63となります。
戻り値は、ドットが点灯しているときは1を、点灯していないときは0を返します。

linec関数……linec(x1,y1,x2,y2)と記述し、始点(x1,y1)と終点(x2,y2)を結ぶ直線を消します。

文法のまとめ

ここで学ぶこと
☆文法の復習

まとめ
◎C言語のプログラムは、制御構造になっています。
◎C言語のプログラムは、多くの関数で構成されています。

コメント

プログラムの説明や見出しなどは、コメント（注釈）で表わします。コメントは、プログラムの実行に全く関係がありません。C言語のコメントは以下の例のように、「／＊」と「＊／」でメッセージ（文字列）を囲みます。

```
/* コンニチハ！ ト シュツリョク スル ﾌﾟﾛｸﾞﾗﾑ */
main ( )
{
      printf ("コンニチハ！") ;
}
```

キーワード

キーワードは、C言語のプログラムの文法で使用するためあらかじめ予約されている言葉です。

キーワードの一覧とその機能を次ページの表に示します。これらのキーワードと同じ名前の変数や関数は宣言できません。また、ここで×印の付いているキーワードは、本機のC言語では使用できない機能です。ただしキーワードとして登録されていますので、変数として使用できません。

表に示すキーワード以外に、ライブラリー関数名も変数名として使用できません。

キーワード一覧

	キーワード	意味と用法
	auto	局所変数の記憶クラス指定
	break	while、do～while、for、switch文からの脱出
	case	switch文の名札（ラベル）
	char	文字型データ（8ビット長）の宣言子
	const	定数の宣言
	continue	while、do～while、for文において、次の繰り返しへジャンプ
	default	switch文で該当しないときの飛び先
	do	処理の繰り返し実行〈do {～} while(式); 〉
	double	倍精度浮動小数点型(64ビット長)の宣言子
	else	if文とともに使用〈if(式) {～} else {～}〉
×	enum	列挙型の宣言子
	extern	外部変数や外部定義の記憶クラス指定
	float	単精度浮動小数点型(32ビット長)の宣言子
	for	繰り返し実行〈for(式1;式2;式3) {～}〉
	goto	指定したラベルへのジャンプ
	if	もし式が真ならば実行〈if(式) {～}〉
	int	整数型の宣言子
	long	倍長整数型(32ビット長)の宣言子
	register	レジスタ変数の記憶クラス指定（本機ではautoと同じ）
	return	関数の値を返し、呼び出しへ戻る
	short	短整数型(16ビット長)の宣言子（本機ではintと同じ）
	signed	符号付き整数型の宣言子
	sizeof	データ型の長さ（本機ではsizeof(型)は使用不可）
	static	静的変数の記憶クラス指定
×	struct	構造体の宣言子
	switch	条件による分岐
×	typedef	新しいデータ型の指定
×	union	共用体の宣言子
	unsigned	符号なし整数型の宣言子
	void	値を返さない関数の型宣言子
×	volatile	プログラムの外側から変更できる型の宣言子
	while	繰り返しの実行〈while(式) {～}〉

データ型の長さと精度

本機のC言語で使用できるデータ型と、その長さを次に示します。

型宣言子	本機のC言語
char	8ビット
short	16ビット
int	16ビット
long	32ビット
float	32ビット
double	64ビット

識別子(変数名や関数名)の付け方

変数名や関数名および#define文で定義する定数名は、一般に識別子と呼ばれます。本機の識別子には次のような規則があります。

①識別子の1文字目は、英字(「A」~「Z」、「a」~「z」)、またはアンダーバー「_」でなければならない。また、カナ文字は使えない。
②識別子の2文字目以降は、英字、アンダーバー、数字(「0」~「9」)のいずれかでなければならない。また、英大文字と英小文字は区別する。
③識別子にキーワードおよびライブラリー関数名(次頁の一覧表を参照)を用いてはならない。
④識別子の長さの制限はないが、先頭の8文字までが名前の識別に使われる。たとえば、「abc012345」と「abc012344」という識別子は同じものとして扱われ、「ab012345」と「abc012335」は別のものとして扱われる。

次に識別子として正しくないものを示します。

正しい例
var、My_Name、jugemujugemu、V100、XX、count、q1_2_5、int_32 (識別子の一部がキーワードになっているのはよい)

正しくない例	理　　由
123、3A、#count	第1文字目が英字、アンダーバー以外の文字である。
switch、gets	キーワードかライブラリー関数名である。
C-1	英字、アンダーバー、数字以外の文字(ハイフン)がある。

キーワードとライブラリー関数名

abort	do	if	sinh
abs	double	inp	sizeof
acos	else	inport	sprintf
acosh	enum	int	sqrt
angle	exit	line	sscanf
asin	exp	linec	static
asinh	extern	log	strcat
atan	fflush	log10	strchr
atanh	fgetc	long	strcmp
auto	fgets	main	strcpy
beep	float	malloc	strlen
break	for	out	struct
breakpt	fprintf	outport	switch
call	fputc	pow	tan
calloc	fputs	printf	tanh
case	free	putc	typedef
char	fscanf	putchar	union
clearerr	getc	puts	unsigned
clrscr	getch	register	void
const	getchar	return	volatile
continue	getpixel	scanf	while
cos	gets	short	
cosh	goto	signed	
default	gotoxy	sin	

注）上記のほか、内部システムで使用している記号定数、関数名があります。これらの名前も識別子として使用することはできません。その場合エラーとなりますので、他の名前を使用してください。

ご注意

◎C言語では通常名前のつづりは小文字を用いますが、#define文で検知する記号定数（EOF、NULL、BUFFSIZE、…）には大文字を使用して、混合を避けるようにしています。本機のC言語もこの記述方法に従っています。

データの表現

データ型のメモリーサイズと扱える値の範囲

それぞれのデータ型のメモリーサイズと扱える値の範囲は、以下のようになります。

種類	データ型	サイズ (bit)	範　　囲
文字型	signed char	8	−128 ～ 127
	unsigned char	8	0 ～ 255
整数型	signed short	16	−32768 ～ 32767
	unsigned short	16	0 ～ 65535
	signed int	16	−32768 ～ 32767
	unsigned int	16	0 ～ 65535
	signed long	32	−2147483648 ～ 2147483647
	unsigned long	32	0 ～ 4294967295
実数型	float	32	−9.99999E+63 ～ −1E−63、 0、+1E−63 ～ +9.99999E+63
	double	64	−9.999999999E+99 ～ −1E−99、 0、+1E−99 ～ +9.999999999E+99

※実数型：float(浮動小数点型)、double(倍精度浮動小数点型)

文字定数(char型)

char型の定数は、次のように「'」(シングルクォーテーション)で文字を囲むことによって表わします。char型の定数は、符号付き定数として扱われます。

'C'　'8'　'¥n'

本機では、文字コードとしてASCIIコードを使用しています。

上の例の'¥n'は「エスケープシーケンス」と呼ばれるもので、英字や数字などで表現できない文字を表現するためのものです。

ASCIIコードを例に示します。詳しくは『ガイドブック』の巻末の「キャラクターコード一覧表」を参照してください。

キャラクター	ASCII コード(10進)	ASCII コード(16進)	ASCII コード(8進)
0	48	30	60
9	57	39	71
A	65	41	101
Z	90	5A	132

キャラクターコード一覧表の中の「BS」、「CR」などは制御文字(Control Character)で、特別な意味を持つ文字定数です。C言語では、これらの制御文字の代わりにエスケープシーケンスが用意されています。それらを次に示します。

表記	コード(16進)	呼び名(略名)	意　　味
¥0	0x00	Null(NUL)	ヌルコード(値は0)
¥a	0x07	Bell(BEL)	音を鳴らす
¥b	0x08	Backspace(BS)	バックスペース(1文字後退)
¥t	0x09	Horizontal Tabulation(HT)	水平タブ
¥n	0x0A	New Line(NL)	改行(+復帰)
¥f	0x0C	Form Feed(FF)	改ページ
¥r	0x0D	Carriage Return(CR)	復帰(行の先頭へ移動)
¥"	0x22		「"」という文字
¥'	0x27		「'」という文字
¥¥	0x5C		「¥」という文字
¥ooo		8進表示	oooに8進数を記入すると、その文字コードの意味になる。
¥xhh		18進表示	hhに16進数を記入すると、その文字コードの意味になる。

例を次に示します。これらは文字定数（1バイトの定数）です。

（例）'¥n'　'¥b'　'¥¥'　'¥033'

整数定数（int型、long型）

整数定数には、int型とlong型（倍長整数型）があります。

整数定数の定数を表記する方法は、大きく分けて10進、16進、8進の3種類があります。long型の定数は、最後にLまたはlを付けます。

整数定数

10進表示	例	1121　　−128　　6800　　123456789L(longの場合)
	注意	後述の8進数と区別するため、「0」で始まってはならない。ただし、0だけの場合はよい。
16進表示	例	0x1B　　0xFFFF　　0x09　　0xffffffL(longの場合)
	注意	「0x」または「0X」を先頭に付ける。
8進表示	例	033　　0777　　012　　01234567L(longの場合)
	注意	「0」を先頭に付ける。

本機で扱う整数型定数は次のとおりです。以下の範囲の数値を使うと、それぞれ以下の型定数に指定されます。負の定数は、符号なし定数に単項マイナス演算子が付いたものです。

種　類	振り分けられる範囲	型
10進数定数	0 ～ 32767	int
	32768 ～ 2147483647	long
	2147483648 ～ 4294967295	unsigned long
8進数定数	00 ～ 077777	int
	0100000 ～ 0177777	unsigned int
	0200000 ～ 017777777777	long
	020000000000 ～ 037777777777	unsigned long
16進数定数	0x0000 ～ 0x7FFF	int
	0x8000 ～ 0xFFFF	unsigned int
	0x10000 ～ 0x7FFFFFFF	long
	0x80000000 ～ 0xFFFFFFFF	unsigned long

上記範囲以外は、不定となります。また、ライブラリー関数の「scanf」、「fscanf」、「sscanf」による入力も上記範囲となります。

実数定数(float型、double型)

実数定数には、小数点を含む浮動小数点定数（float型)、倍精度浮動小数点定数(double型)があり、次のような形式で記述します。指数部はEまたはeを用いて記述します。

（例） 3.1416　　−1.4142　　1.0e−4　　1.23456E5

本機で扱える実数定数の範囲は、次のとおりです。

float型	−9.99999E+63 ～ −1E−63、0、+1E−63 ～ +9.99999E+63
double型	−9.999999999E+99～ −1E−99、0、+1E−99～ +9.999999999E+99

文字列定数(char型)

文字列定数は文字定数を並べたような形式で、「"」(ダブルクォーテーション）によって文字の並びを囲みます。文字列の構造は、文字列の最後に「¥0」(NULL）が付けられます。例を次に示します。

"How do you do?"⟶

'H'	'o'	'w'	'␣'	'd'	'o'	'␣'	'y'	'o'	'u'	'␣'	'd'	'o'	'?'	'¥0'

('␣'は空白の意味です)

型宣言

変数や関数をＣ言語で使用する場合は、使用時に必ず型宣言をしなければなりません。本機には、次のように型宣言を行なうための型宣言子があります。

記憶クラス

本機の記憶クラスは、宣言した変数を確保するメモリー領域を指定したり、変数や関数の通用範囲（プログラムのどの範囲で有効か）を指定するものです。

変数の記憶クラスには次のような種類があり、それぞれ変数の使い方によって使い分けます。

記憶クラス	変数の使い方
自動変数(auto)	一時的な内部変数で、宣言した関数内でのみ有効です。関数から出た場合、その変数は無効になります。「auto」の記述は、省略できます。
静的変数(static)	プログラム実行中、常にその領域を保持しておき、プログラムの実行全体を通じて値をアクセス、操作することができる変数です。
レジスタ変数(register)	CPUのレジスタに変数を設定して、処理速度の向上を図るための変数です。本機では、自動変数(auto)と同様に扱われます。
外部変数(extern)	関数外で宣言される大域変数です。外部変数は外部参照変数とも呼ばれ、どの関数でも参照できます。同じ名前の変数のうち、一つはextern宣言のない変数（参照元）がなければなりません。

注意

①auto変数は次のプログラム(ａ)のように、関数に入った直後の先頭で宣言する内部変数です。プログラム(b)のように、関数内の実行の途中で変数の宣言をすると、エラーになります。

auto変数の例

プログラム(a)OK

```
func(a)
double a;
{
  int i, j, x;
  float fx, fy;
  long lx, ly;
  .....
}
```

auto変数は、関数の先頭で宣言します。

プログラム(b)エラー

```
func(a)
double a;
{
  int i, j, x;
  printf("%f", a);
  float fx, fy;
  .....
}
```

文字型・整数型の型宣言

記憶クラス	型修飾子 1	型修飾子 2	型指定子
auto static register extern	const	signed unsigned	char short[int] int long[int]

①基本形

〔記憶クラス〕〔型修飾子 1〕〔型修飾子 2〕型指定子　オブジェクト;

(オブジェクト：変数、関数、ポインタなど)

オブジェクトは、「,」(カンマ) を使って複数指定できます。

記憶クラス、型修飾子 1 、型修飾子 2 は省略することができます。

型修飾子 1 を指定すると、オブジェクトは変更不可能になります(本機では、型修飾子 1 はchar、short〔int〕、intの型指定子で使用できます)。

型修飾子 2 を省略すると、signed指定になります。

②特殊形

〔記憶クラス〕{型修飾子 1 / 型修飾子 1　型修飾子 2 / 型修飾子 2} オブジェクト;

特殊形の宣言は、以下のように扱われます。

const	= const signed int
const signed	= const signed int
const unsigned	= const unsigned int
signed	= signed int
unsigned	= unsigned int

実数型の型宣言

記憶クラス	型修飾子 1	型指定子
auto static register extern	long	float
		double

①基本形

〔記憶クラス〕〔型修飾子１〕型指定子　オブジェクト;

（オブジェクト：変数、関数、ポインタなど）

オブジェクトは、「,」（カンマ）を使って複数指定できます。

記憶クラス、型修飾子１は省略することができます。

型修飾子１は、型指定子がfloatの場合のみ有効です。

long float＝double

その他の型宣言

記憶クラス	型指定子
static extern	void

①基本形

〔記憶クラス〕型指定子 オブジェクト;

（オブジェクト：関数）

voidは型を持たないことを示す指定子です。値を返さない関数の宣言に使用します。

オブジェクトは、「,」（カンマ）を使って複数指定できます。

本機では、次のようなvoid宣言はできません。

●関数の引数なし宣言

エラー

```
a(void)
{
    ⋮
}
```

→

OK

```
a( )
{
    ⋮
}
```

カッコ内には何も書かない。

●型なしポインタの宣言

エラー

```
void *p;
```

→

OK

```
int *p;
```

対応した型宣言を行なう。

変数の初期化

本機では、変数の型宣言時に変数の初期値を与えることができます。ただし、初期値は定数でなければなりません。

（例）
```
static int i=123;
char c='A';
```

コラム●const修飾子(定数と定数ポインタ宣言)●

const指定を行なうと、オブジェクトの値はプログラム実行中に変更できません。変更したり変更される可能性がある場合には、エラーとなります。

例：変数a、bとポインタap、bpを次のように指定したとき

```
const int a=123,*ap;
      int b=456,*bp;
```

実行できる文

```
ap=&a;
ap=&b;
bp=(int *)&a;
bp=(int *)ap;
```

実行できない文(エラーになる)

```
a++;
a=5;
*ap=5;
bp=&a;
bp=ap;
```

また、次のような指定はできません。

```
int const ～    (型指定子と型修飾子の順序が逆)
int *const ～   (「*const」によるポインタ指定はできない)
```

配列とポインタ

配列

配列は、一つの変数名の下に複数のデータを格納するための宣言です。配列は、単純変数とほぼ同様に扱うことができます。

本機では、n次元配列(メモリーが許す限り)まで使用することができます。

(例) int m[2][12][13];

配列の初期化

内部変数の静的変数(static)および外部変数の配列は初期化ができますが、本機では、次のような「{」、「}」を使った配列の初期化はできません。

①char a[3]={'A','B','¥0'};

②int b[3]={1,2,3};

配列の初期化は、型宣言の後に次のように行なってください。

①●文字列で与える

```
char a[3]="AB";
```

●型宣言後に代入演算子(または関数)を使って、データを格納する

```
char a[3];
a[0]='A', a[1]='B',a[2]='¥0';   (またはstrcpy(a,"AB");)
```

②型宣言後に代入演算子を使って、データを格納する

```
int b[3];
b[0]=1, b[1]=2, b[2]=3;
```

ポインタ

ポインタは実際に変数などが格納されているメモリーのアドレスを指す変数で、16ビット長です。ポインタの宣言は、変数名の前に「＊」(アスタリスク)を付けます。

ポインタも配列と同様に初期化ができますが、本機では文字列定数によるポインタの初期化はできません。

× char ＊P="AB";

ポインタは、次のように使用します。

int x, ＊px; x=＊px;	左の例ではpxが指している内容がxに代入されます。
int x, y, ＊px; px=&x; y=＊px;	左の例ではxのアドレスがpxに代入された後、pxが示すアドレスの内容をyに代入します。 すなわち、「y=x;」と同じことになります。

```
main( )
{
  char *p;

  p="Casio";
  printf("%c %s ¥n", *p,p);
}
```

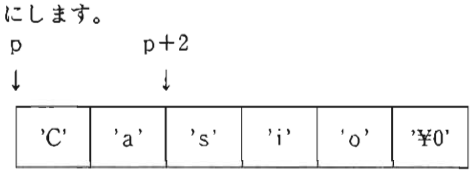
文字列においてポインタを用いると、文字列を単一文字に分解することができます。
実行させると、結果は次のようになります。

C Casio

ポインタpは「Casio」の「C」を指します。

```
main( )
{
  char *p;

  p="Casio";
  printf("%c %c ¥n", *p, *(p+2));
}
```

「Casio」の「C」と「s」を取り出すときは、左のようにします。




```
main( )
{
  int i, a[5], *pa;

  pa=a;
  for(i=0;i<=4;i++)
    *(pa+i)=i;
  for(i=0;i<=4;i++)
    printf("%d" a[i]);
  printf("¥n");
}
```

配列の各要素は、ポインタで示すこともできます。左の例では、配列aの各要素をpaで示しています。

配列名「a」は、配列の最初の要素a[0]を指すポインタを表わします。したがって、配列の各要素を指すポインタは、次のようになります。

ポインタ	配列	配列の値
pa	a[0]	0
pa+1	a[1]	1
pa+2	a[2]	2
pa+3	a[3]	3
pa+4	a[4]	4

```
main( )
{
  char *pc, c[80];

  pc ="abcdefgh";
  strcpy(c, pc);
  printf("%s¥n" ,c);
}
```

ポインタは、変数のアドレスを持っているだけです。したがって、場合によっては格納領域の確保は別に行なう必要があります。

左の例では、ポインタpcと80文字の格納領域を確保するために配列cを宣言し、ポインタpcを配列のポインタcと共通にしています。

その後、strcpy関数によって、文字列「abcdefgh」が配列cにコピーされます。

ポインタを用いると明快で単純なプログラムを作ることができる反面、予想しないアドレスを指すポインタを作ることもありますので、十分に注意してください。

また、ポインタや配列を扱うとき、確保した容量を超えてデータの書き込みを行なったり、ポインタによる不正なアドレスへの書き込みを行なうと、予期しない動作をしますので、絶対に行なわないようにしてください。

演算子

C言語には、BASIC、FORTRANやPascalなど他の言語には見られない多くの演算子があります。

演算子は、「＋」、「－」、「＊」、「／」（加減乗除算）などのように、変数などの値を変化させる処理を行ないます。

また数多くあるC言語の演算子には、優先順位と結合規則というものが決められています。

演算子の種類と機能

一次演算子	( )、func( ) x[ ]、y[ ][ ]	カッコ、関数の引数の演算 配列要素の指定
単項演算子	＊px &x －x ＋＋x、－－x x＋＋、x－－ ～x !x (型)x sizeof(x)	ポインタの示す内容の指定 変数xのアドレス xの負値 変数xを使う前に＋1、－1 変数xを使った後で＋1、－1 xのビットごとのnot(反転) xの論理not(xが0以外なら0、0なら1) xの強制型変換(cast演算) 変数xのバイト長の値。sizeof(型)は使用不可
二項演算子	x＊y、x/y、x％y x＋y、x－y x ≪ y、x ≫ y x＞y、x＜y x＞＝y、x＜＝y x＝＝y x!＝y x&y x＾y x｜y x&&y x‖y	乗算、除算、剰余(xをyで割った余り) 加算、減算 xをy回、左ビットシフト、右ビットシフト xとyの大小比較(真ならば１、偽ならば０) xとyの大小比較(真ならば1、偽ならば0) 等値比較(等しければ1、等しくなければ0) 不等値比較(等しくなければ1、等しければ0) xとyのビットAND xとyのビット排他的OR(XOR) xとyのビットOR xとyの論理AND(xとyが共に0以外なら1) xとyの論理OR(xとyのどちらかが0以外なら1)
三項演算子	x?y：z	xを評価して、xが０以外(真)ならばyの値を、xが０(偽) ならばzの値を返す(条件演算子)。

代入演算子	x＝y	変数xにyを代入
	x＊＝y（x＝x＊y）	xにyをかけてxに代入
	x／＝y（x＝x／y）	xをyで割ってxに代入
	x％＝y（x＝x％y）	xをyで割った余りをxに代入
	x＋＝y（x＝x＋y）	xにyを加えてxに代入
	x－＝y（x＝x－y）	xからyを引いてxに代入
	x≪＝y(x＝x≪y)	xをy回左ビットシフトしてxに代入
	x≫＝y(x＝x≫y)	xをy回右ビットシフトしてxに代入
	x&＝y（x＝x&y）	xとyのビットANDをxに代入
	x^＝y（x＝x^y）	xとyのビット排他的OR(XOR)をxに代入
	x｜＝y（x＝x｜y）	xとyのビットORをxに代入
順序演算子	x、y （例） （a＝b＋c、x＝a／2）	xとyを評価して、結果としてyを返す。 （例）aにb＋cを代入し、次にxにa／2を代入して、xを値とする。

次に、上記に示した演算子の種類と優先順位および結合規則を表で示します。

演算子の種類と優先順位および結合規則

優先順位	演算子		結合規則
高い		「( )」「[ ]」	→
	単項	「!」「~」「++」「--」「(型)」「＊」[注1)]「&」「sizeof」「－」[注2)]	←
	乗除	「＊」[注3)]「／」「％」	→
	加減	「＋」「－」[注4)]	→
	シフト	「≪」「≫」	→
	比較	「＞」「＜」「＞＝」「＜＝」	→
	等価比較	「＝＝」「！＝」	→
	ビットAND	「&」	→
	ビットXOR	「^」	→
	ビットOR	「｜」	→
	論理AND	「&&」	→
	論理OR	「｜｜」	→
	条件	「?:」	←
	代入	「＝」「＋＝」「－＝」「＊＝」「／＝」「％＝」 「&＝」「｜＝」「^＝」「≪＝」「≫＝」	←
低い	順序	「,」	→

注1）間接指定記号の「＊」
注2）負数指定の「－」（マイナス演算子）
注3）乗算記号の「＊」
注4）減算記号の「－」

→左から右に演算
←右から左に演算

型変換(cast演算子)

C言語では異なるデータ型のデータを有する演算を行なう場合に、暗黙のうちに各々のデータ型を変換し、型の統一を行なってから演算が行なわれます。また、cast演算子を使って、強制的にデータ型を変換することもできます。

一般に演算の際に暗黙のうちに行なわれる型変換は、

char ＜ int ＜ long ＜ float ＜ double（右にいくほど上位の型）

というように、上位のデータ型に変換されて演算が行なわれます(正確には、char型とfloat型は、演算時に存在せず、char型はint型に、float型はdouble型に変換されて、演算が行なわれます)。

本機のデータ型変換規則は、次のとおりです。

変換前	変換後	変換方法
char	int	符号拡張
char	unsigned int	符号拡張
char	long	intに変換後、longに変換する。
char	unsigned long	intに変換後、unsigned longに変換する。
char	float	longに変換後、floatに変換する。
char	double	longに変換後、doubleに変換する。
unsigned char	int	ゼロ拡張
unsigned char	unsigned int	ゼロ拡張
unsigned char	long	unsigned intに変換後、longに変換する。
unsigned char	unsigned long	unsigned intに変換後、unsigned longに変換する。
unsigned char	float	unsigned longに変換後、floatに変換する。
unsigned char	double	unsigned longに変換後、doubleに変換する。
int	char	下位バイト保存
int	unsigned char	下位バイト保存
int	long	符号拡張
int	unsigned long	符号拡張
int	float	longに変換後、floatに変換する。
int	double	longに変換後、doubleに変換する。
unsigned int	char	下位バイト保存
unsigned int	unsigned char	下位バイト保存
unsigned int	long	ゼロ拡張
unsigned int	unsigned long	ゼロ拡張
unsigned int	float	unsigned longに変換後、floatに変換する。
unsigned int	double	unsigned longに変換後、doubleに変換する。

変換前	変換後	変　換　方　法
long	char	下位バイト保存
long	unsigned char	下位バイト保存
long	int	下位ワード保存
long	unsigned int	下位ワード保存
long	float	−2147483648～2147483647が有効(有効桁数最大6桁)。
long	double	−2147483648～2147483647が有効(有効桁数最大10桁)。
unsigned long	char	下位バイト保存(有効桁数最大10桁)
unsigned long	unsigned char	下位バイト保存
unsigned long	int	下位ワード保存
unsigned long	unsigned int	下位ワード保存
unsigned long	float	0～4294967295が有効(有効桁数最大 6 桁)。
unsigned long	double	0～4294967295が有効(有効桁数最大10桁)。
float	char	longに変換後、charに変換する。
float	unsigned char	longに変換後、unsigned charに変換する。
float	int	longに変換後、intに変換する。
float	unsigned int	longに変換後、unsigned intに変換する。
float	long	小数点以下を切り捨てる。
float	unsigned long	小数点以下を切り捨てる。
float	double	内部表現を変換する。
double	char	longに変換後、charに変換する。
double	unsigned char	longに変換後、unsigned charに変換する。
double	int	longに変換後、intに変換する。
double	unsigned int	longに変換後、unsigned intに変換する。
double	long	小数点以下を切り捨てる。
double	unsigned long	小数点以下を切り捨てる。
double	float	指数表現の仮数部 7 桁目を四捨五入する。

※ポインタからそれぞれの型への変換は、ポインタを「unsigned int」として扱い、上記データ型変換規則に従います。ただし、float、doubleへの変換はできません。

※ void型からの変換はできません。

制御構造

C言語のプログラムは、実行の単位である文をいくつも書き、それらを順次実行させるという構成になっています。

基本的にプログラムは上から順に一つずつ実行されていきますが、C言語には、実行を制御する次の実行制御命令があります。

①条件によって分岐させる
②繰り返す
③実行の順序を変える

ことです。

文(単文)

文の最小単位は、データや演算子や関数呼び出しを含んだ単一の式を実行させる文です。式の後に「;」を付けます。「;」は文の終わりを表わし、文と文を区切る役目があります。

普通、文は次のような代入か関数呼び出しです。

	(単文の例)
式;	x＝sin(a＋b)－c; printf("x＝%10.4f¥n", x) ; a＝b＋(c＝getchar( ));

また、代表的な

```
int a;
```

のような変数の型宣言も文となります。

複文(ブロック)

複文はいくつかの文を「{」と「}」でくくって、一つの文にまとめたものです。文法的には一つの文と同じと見なして取り扱われます。複文(ブロック)の終わりを示す「}」の後には、「;」を付けません。

```
{
    文1;
    文2;
     ⋮
    文n;
}
```

(復文の例)

```
main( )
{
    int i,j＝0;
    for(i＝1;i＜＝10;i＋＋)
        j＝j＋i;
    printf("コ〝 ウケイ＝%d¥n",j);
}
```

コラム●ブロック内での型宣言●

本機では、以下のようなブロック内での型宣言指定はできません。

```
main( )
{   int i;
    {   int j;
      ⋮
      ⋮
    }
}
```

実行制御命令

次に実行制御命令(条件分岐や繰り返しなどの命令)の説明と代表的な書式を示します。書式中にある文は、単文でも複文(ブロック)でも構いません。

条　件　分　岐	

if (式) {
　⋮
　文
　⋮
}
	式が真ならば、
	
	文を実行します。

if (式1) {
　⋮
　文1
　⋮
}
else {
　⋮
　文2
　⋮
}
	式1が真ならば、
	
	文1を実行し、
	
	それ以外ならば（偽ならば）
	
	文2を実行します。

if (式1) {
　⋮
　文1
　⋮
}
else if (式2) {
　⋮
　文2
　⋮
}
else if (式3) {
　⋮
　文3
　⋮
}
　⋮
else if (式n) {
　⋮
　文n
　⋮
}
else {
　⋮
文n＋1
　⋮
}
	式1が真ならば、
	
	文1を実行し、
	
	それ以外で(偽で)、式2が真ならば、
	
	文2を実行し、
	
	それ以外で(偽で)、式3が真ならば、
	
	文3を実行し、
	
	……
	それ以外で(偽で)、式nが真ならば、
	
	文nを実行し、
	
	それ以外ならば（式1～式nが偽ならば）
	
	文n＋1を実行します。

ご注意

if文などに続く { } のブロック内に文が一つしかない場合は、{ } は省略できます。ここでは見やすくするために、このように表記してあります。

条件分岐	

switch（式）{
　case 定数1：文1
　　⋮
　　break；
　case 定数2：文2；
　　⋮
　　break；
　　⋮
　case 定数n：文n
　　⋮
　　break；
　default：文n＋1
　　⋮
}
	switch文は、最初に条件式を評価し、その値と一致する定数のcaseラベルから実行します。条件式、caseラベルにある定数は、整数型でなければなりません。
	条件式の値がcaseラベルの定数1と一致した場合は、文1から実行します。実行後、break文によってswitch文を抜けますが、このbreak文がないと、その後に続く文（文2～文n＋1）を順次実行していきます。
	
	条件式に一致するcaseラベルの定数がない場合は、defaultラベルから実行します。
	caseラベル、defaultラベルは省略が可能で、条件式による該当ラベルがなければ、エラーにならずにswitch文を抜けます。

繰り返し制御	

while（式）{
　⋮
文
　⋮
}
	式が真の間、
	
	文を実行します。

do { 　　⋮ 　文 　　⋮ } while（式）；	 文を実行します。 式が真ならば、元に戻ります。 以下、式が真の間、文の実行を繰り返します。

for（式1；式2；式3）{
　　⋮
　文
　　⋮
}
	式1を実行して初期設定を行ないます。
	式2が真ならば、文を実行します。
	式3を実行して、条件を更新します。
	式2が真ならば、文を実行します。
	以下、式2が真の間、文の実行、式3による更新を繰り返します。

無限ループ	

while (1) {
　　⋮
　　文
　　⋮
}
	文を繰り返し実行します。
	(while文の条件がいつも真(1))

do { 　　⋮ 　　文 　　⋮ } while (1);	文を繰り返し実行します。 (do ～ while文の条件がいつも真(1))

for (;;) {
　　⋮
　　文
　　⋮
}
	文を繰り返し実行します。
	(for文の初期設定、条件判断、条件更新がない)

break文、continue文	

while (1) {
　　⋮
　　文1
　　⋮
　　if (式) break;
　　⋮
　　文2
　　⋮
}
	while文による無限ループが実行されます。
	式が真ならばbreak文が実行され、無限ループから抜け出します。
	(文1のみが実行され、文2は実行されません)

while (式1) {
　　⋮
　　if (式2) continue;
　　⋮
}
	式1が真の間、{ } の中が実行されます。
	式2が真ならばcontinue文が実行され、while文に戻り、式1が実行されます。

do {
　　⋮
　　if(式1) continue;
　　⋮
} while(式2);
	式1が真ならば、continue文が実行されて、ループの終わりのwhile文に制御を移し、式2が評価されます。

無条件ジャンプ	
goto ラベル; 　　⋮ ラベル: 　　⋮	ラベル位置に、無条件にジャンプします。

関数

一つの仕事を行なう単位が「関数」です。C言語のプログラムは、多数の関数から構成されるのが普通です。Cのプログラムを作るということは、関数を作るということとほぼ同じ意味です。

関数には、あらかじめ用意されている「printf」や「scanf」、「getchar」などの入出力関数、「sin」や「cos」などの数学関数、および文字列を処理する関数などがあります。これらを「ライブラリー関数」と呼びます。ライブラリー関数については、リファレンスの「ライブラリー関数リファレンス」で詳しく説明します。

次に、ユーザーが自分自身で新しい関数を書く場合の注意点を説明します。

main関数

main関数は、主プログラムであることを示す特別な関数です。プログラムの一番始めに起動します。

本機では、右のようなプログラムでコマンド入力ラインを調べることはできません。プログラム中でscanf関数などを使ってキー入力を行なうようにしてください。

```
main(argc, argv)
int argc;
char *argv[ ];
{
      ⋮
}
```

関数の型宣言

int型以外の値を返すには、関数の型宣言が必要です。例として、double型の値を返す関数「dsquare」を作って、1から10までの2乗を出力するプログラムを示します。

```
main( )
{
      double d, dsquare( );

      for (d=1.0; d<=10.0; d+=1.0)                    実引数
            printf ("(%f)^2=%f¥n", d, dsquare(d)); ←呼び出し元
}
                  仮引数
                  ↓
double dsquare(x) ←呼び出し先
double x; ←仮引数はここで宣言します
{
      return (x*x); ←関数での処理の結果を返します
}
```

このような方法を「Call by Value」といいます。

このようにint以外を値として返す関数では、関数の宣言ばかりでなく、使う側(main関数内)でも宣言が必要です。

また、次に示すように、記憶クラスのextern宣言をして使用することができます。

```
extern double dsquare( );

main( )
{
      double d;

      for (d=1.0;d<=10.0;d+=1.0)
            printf ("(%f) ^2=%f¥n", d, dsquare(d));
}

double dsquare(x)
double x;
{
      return(x*x);
}
```

その他の注意点として、次のようなことに留意してください。

・再帰呼び出し：本機のC言語においても、再帰呼び出しを使用することができます。
・void関数宣言が可能です。
・次のような関数へのポインタの宣言はできません。

```
main( )
{
      void pr( ), (*pr_p)( );
      pr_p=pr;
      pr( );
      (*pr_p)( );
}
void pr( )
{
      printf("In pr_func¥n");
}
```

ライブラリー関数は、型指定を宣言せずに使用できます。一般のC言語コンパイラなどでは「stdio.h」や「math.h」、「ctype.h」などのヘッダーファイルを取り込まなければ、ライブラリー関数が使えない場合がありますが、本機ではこのようなヘッダーファイルを取り込まなくてもライブラリー関数を使用できます。

プリプロセッサ

C言語の処理系には、実行部分をコンパイルする前に、プリプロセッサと呼ばれる前処理を行なう機能が備わっています。本機では、#include文と#define文が使用できます。これらは、必ずファイルの第一カラムの位置から書かなければなりません。

●#include文

#include文は、指定したファイルの内容を#include文のある場所にそのまま挿入します。

#include "ファイルエリア名"　("ファイルエリア名":"F0"～"F9"のいずれか)

ただし、以下の点にご注意ください。

①本機ではライブラリー関数使用時に

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

などとして、ヘッダーファイルを取り込む必要がはありません。

②次のような指定はできません。

(1)再帰的な指定

```
「F0」ファイル
#include "F0"
main( )
{
        ⋮
}
```

(2)双方向の呼び出し

```
「F0」ファイル
#include "F1"
main( )
{
        ⋮
}
```

```
「F1」ファイル
#include "F0"
a( )
{
        ⋮
}
```

(3)include先でのinclude指定(カレントファイルからのレベルが2段以上)

```
「F0」ファイル
#include "F1"
main( )
{
        ⋮
}
```

```
「F1」ファイル
#include "F2"
a( )
{
        ⋮
}
```

(4) 2つのファイルにまたがるようなプログラム指定

「F0」ファイル

```
#include "F1"
{
```

「F1」ファイル

```
main( )
{
        :
```

● #define文(マクロ展開)

```
#define  識別子   文字列   (定数の定義や文字列の置換)
                  └── (省略できません)
```

ただし、次の点にご注意ください。

①本機では、#define文によって次のような引数付きマクロ定義をすることはできません。

```
#define isupper(x) ((x>='A'  &&  x<='Z')? 1 : 0)
```

代わりに、次のような関数を作ってください。

```
isupper(x)
int x;
{
  return((x>='A' && x<='Z')?1:0);
}
```

……代わりに使用してください。

②本機では、次の情報があらかじめ登録されています。

```
#define NULL 0
#define EOF -1
#define FILE char
FILE *stdin;
FILE *stdout;
FILE *stdprn;
FILE *stderr;
```

「stdin」、「stdout」、「stdprn」、「stderr」を使用する場合は、extern宣言を行なう必要はありません。

マニュアルコマンドリファレンス

RUN

（書式） RUN　　(run)
RUN＞"PRN：" (run＞"PRN：")
RUN＞"prn：" (run＞"prn：")

（機能） プログラムを実行させます。
"PRN："または"prn："を指定すると、実行結果をプリンタに出力します。
指定を省略すると、画面に出力します。

EDIT

（書式） EDIT　(edit)

（機能） エディタに入ります。
ロード(Load)または実行(Run)でエラーが発生した場合や、ブレーク(Break)がかけられた場合は、その行を表示します。

TRON

（書式） TRON　(tron)

（機能） トレースしながらプログラムを実行するトレースモードを指定します。トレースモードを指定してプログラムを実行すると、プログラムを1行実行するごとに、次に実行する行を表示します。

トレース中のキー操作　[↵]、[T]キー：トレースモードのままで、実行を再開します。
[N]キー　：トレースモードを解除して、実行を再開します。
[D]キー　：トレースモードのままで、変数モードに入ります。
変数モードで[↵]キーのみを押すと、トレースモードに戻ります。
[BRK]キー　：トレースの実行を途中で中断します。

トレースモードは、起動時はOFFになっています。

変数モード

変数モードは、変数名を入力することによって、トレース中のプログラムに使われている変数の型と値を表示するモードです。

このモードに入ると「var＞_」と表示され、次のキー操作が行なえます。

変数モード中のキー操作

変数名[↵]：指定した変数の型とその時点の値を表示します。
[↵]のみ　：トレースモードに戻ります。

TROFF

（書式） TROFF　(troff)

（機能） トレースモードを解除します。

ライブラリー関数リファレンス

ライブラリー関数

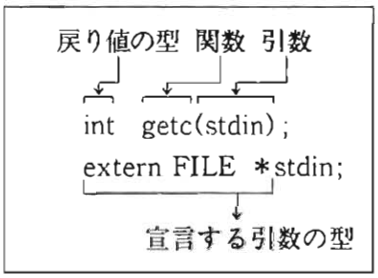
あらかじめ定義されているので、ユーザーが定義しないで呼び出して使える関数が、ライブラリー関数です。

本機のC言語には、次ページ以降に示す入出力関数、実行制御関数、メモリー関数、文字列関数、数値関数、画面制御関数、実行制御関数、その他の関数が用意されています。

ライブラリー関数リファレンスの見方

リファレンスでは、書式、戻り値、機能、例を示します。

(例)



「書式」については、次のように示してあります。

戻り値の型	関数によって得られる戻り値の型を示します。 戻り値の型は、次の型宣言子で記述します。 char int unsigned int(=unsigned) long unsigned long double void
関　　数	関数名です。関数の次の「(」と「)」の中に引数を書きます。 「;」は、文の区切りを意味する記号です。
引　　数	引数を書きます。 引数が二つ以上のときは、「,」(カンマ) で区切ります。 引数のない関数では、「( )」のようにカッコだけを書きます。
宣言する引数の型	関数で使用する引数の型を関数名の後または下に指定します。 引数のない関数では何も指定しません。

getchar

（書式）
int getchar( );

（戻り値）
読み込んだ文字のコード。
int型。

（機能）
キーボードから1文字読み込みます。
入力は、↵キーを押すと行なわれます。
SHIFT ↵と操作すると、EOF(−1)が入力されます。

（例）
```
int c;
    ⋮
c=getchar( );
```

getc

（書式）
int getc(stdin);
extern FILE *stdin;

（戻り値）
読み込んだ文字のコード。
int型。

（機能）
読み込んだ1文字のコードを返します。
入力は、↵キーを押すと行なわれます。
動作は、getchar関数と同じです。
stdin(キーボード)以外の入力指定はできません。

（例）
```
extern FILE *stdin;
int          c;
    ⋮
c=getc(stdin);
```

fgetc

（書式）
int fgetc(stdin);
extern FILE *stdin;

（戻り値）
読み込んだ文字のコード。
int型。

（機能）
キーボードから1文字読み込みます。
入力は、↵キーを押すと行なわれます。
動作は、getchar関数と同じです。
stdin(キーボード)以外の入力指定はできません。

（例）
```
extern FILE *stdin;
int          c;
    ⋮
c=fgetc(stdin);
```

putchar

（書式）
```
int putchar(c);
int c;
```

（戻り値）
出力した文字のコード。
int型。

（機能）
stdout(表示画面)に1文字cを出力します。

（例）
```
main( )
{
  char buffer [64];
  int  i, c;

  gets(buffer);
  for(i=0;buffer [i]!='¥0';i++){
    c=putchar(buffer[i]);
    if(c==EOF) break;
  }
}
```

putc

（書式）
```
int     putc(c, stdout);
int     putc(c, stdprn);
int     c;
extern FILE *stdout, *stdprn;
```

（戻り値）
出力した文字のコード。
int型。

（機能）
stdout(表示画面)またはstdprn(プリンタ)に1文字cを出力します。

fputc

（書式）
```
int     fputc(c, stdout);
int     fputc(c, stdprn);
int     c;
extern FILE *stdout, *stdprn;
```

（戻り値）
出力した文字のコード。
int型。

（機能）
stdout(表示画面)またはstdprn(プリンタ)に1文字cを出力します。
動作は、putc関数と同じです。

（例）
```
extern FILE *stdout;
char        buffer [30];
int         c;
    :
c=fputc(buffer[0], stdout);
```

gets

（書式）

```
char *gets(string);
char *string;
```

（戻り値）

通常は格納されたデータのポインタを返し、EOFの場合はNULLを返します。
char型

（機能）

キーボードから1行読み込み、それをstringに格納します。

文字列は、改行文字またはEOFまで読み込まれます。

SHIFT ↵と操作すると、EOF(−1)が入力されます。

改行文字は、string中では「¥0」(NULL)文字に置き換えられます。

通常はstringの値を返し、1文字も読み込まれずにEOFが検出された場合は、NULLを返します。

（例）

```
char string[30], *result;
        :
result=gets(string);
```

fgets

（書式）

```
char        *fgets(string, count, stdin);
char        *string;
int         count;
extern FILE *stdin;
```

（戻り値）

通常は格納されたデータのポインタを返し、EOFの場合はNULLを返します。
char型。

（機能）

stdin(キーボード)から文字列を読み、それをstringに格納します。

文字は、改行文字またはEOFを読み込むまで、または読み込んだ文字数がcount−1になるまで読まれます。改行文字は「¥0」(NULL)文字に置き換えられませんが、文字列の最後に「¥0」(NULL)文字が付加されます。

stdin(キーボード)以外の入力指定はできません。

通常はstringの値を返し、1文字も入力されずにEOFが検出された場合は、NULLを返します。

（例）

```
entern FILE *stdin;
char        string[30], *result;
        :
result=fgets(string, 30, stdin);
```

puts

(書式)

```
int    puts(string);
char *string;
```

(戻り値)

改行文字コード。
int型。

(機能)

表示画面にstringを出力します。
文字列の終了を表わす「¥0」(NULL)文字は、改行文字に置き換えて書き込みます。

(例)

```
int result;
      :
result=puts("string");
```

fputs

(書式)

```
int          fputs(string, stdout);
int          fputs(string, stdprn);
char         *string
extern FILE *stdout, *stdprn;
```

(戻り値)

書き込んだ最後の文字コード。
int型。

(機能)

stdout(表示画面)またはstdprn(プリンタ)にstringを出力します。
文字列の終了を表わす「¥0」(NULL)文字は書き込みません。
stringがヌル(NULL)の場合は、「¥0」(NULL)文字のコードを返します。
stdout(表示画面)、またはstdprn(プリンタ)以外の出力指定はできません。

(例)

```
extern FILE *stdprn;
int          result;
      :
result=fputs("string", stdprn);
```

printf
fprintf
sprintf

（書式）

```
int     printf(format[, argument……]);
int     fprintf(stdout, format[, argument……]);
int     fprintf(stdprn, format[, argument……]);
int     sprintf(buffer, format[, argument……]);
char    *format;
char    *buffer;
extern FILE *stdout, *stdprn;
```

（戻り値）

出力した文字数。
int型。
（sprintfでの最後の「¥0」（NULL）文字は数えません）
エラーなら、−1を返します。

（機能）

argumentをformatに従って変換し、printfは表示画面に、fprintfはstdout（表示画面）またはstdprn（プリンタ）に、sprintfはbufferにそれぞれ出力します。

sprintfの場合だけ、最後に「¥0」（NULL）文字を出力します。

formatは0文字以上の文字列で、普通の文字、エスケープシーケンス、変換仕様からなります。普通の文字とエスケープシーケンスは、現われる順にそのまま出力されます。

変換仕様が現われると、後続のargumentを順番に取り出し、その仕様に従って変換し、出力します。

argumentが変換仕様よりも多いときは、余分なargumentは無視され、少ないときは結果が不定となります。

（例1）

```
int count=234;
printf("%d%06d%X%x%o¥n",
  count, count, count, count, count);
```

出力結果　234000234EAea352

（例2）

```
int count=234;
printf("¦%d¦%6d¦%-6d ¥n",
  count, count, count);
```

出力結果　¦234¦　234¦234　¦

変換仕様の書式

変換仕様は、「%」で始まる次の文字列です。

%〔フラグ〕〔フィールド幅〕〔.精度〕〔l〕変換文字

ただし、「%」の後に変換仕様として意味を持たない文字が続く場合には、その文字列以降、次の「%」までそのまま出力されます。

「%」を出力するときは、「%%」とします。

変換仕様の文字列

フラグ「－」

変換結果をフィールド内で左寄せにします。デフォルト値は「右寄せ」です。

フィールド幅

最小フィールド幅を符号なしの10進整数で指定します。

変換結果がフィールド幅より小さい場合は、余白はパッド文字で埋められます。フラグに「－」が指定してあるときは右側に、フラグ省略時は左側にパッド文字が埋められます。

パッド文字は、数置変換でフラグに「－」の指定がなく、フィールド幅を指定する10進整数が「０」で始まっている場合は「０」、それ以外は空白文字(スペース)になります。

変換結果がフィールド幅より大きい場合、またはフィールド幅が指定されていない場合は、変換結果がそのまま出力されます。

精度

精度は、ピリオドを前に付けた符号なしの10進整数で指定します。ピリオドのみ書かれ、10進整数が省略されたときは、「０」として扱われます。変換文字により、次のような意味を持ちます。

「d, o, u, x, X」	出力する最小の桁数を指定します。デフォルトは「１」です。
「e, E, f」	小数点以下の桁数を指定します。精度が「０」の場合、小数点も出力されません。デフォルトは「６」です。
「g, G」	小数点以下の桁数を指定します。精度が「０」の場合、小数点も出力されません。デフォルトは「6」です。
「s」	出力する最大の文字数を指定します。デフォルトは、「¥0(NULL)文字が出て来るまですべて」です。
その他	その他の文字が書かれると、精度は無視されます。

l(倍長整数型指定)

変換文字により、次のような意味を持ちます。

「d, o, u, x, X」	引数がlong型であることを指定します。
その他	その他の文字が書かれると、「ｌ」は無視されます。

変換文字

「d」	整数引数が符号付き10進表記に変換されます。
「o」	符号なし８進表記に変換されます。
「u」	符号なし10進表記に変換されます。
「x, X」	符号なし16進表記に変換されます(x：「a」～「f」,X：「A」～「F」)。
「f」	double引数が、「〔－〕dd.dddddd」の形の10進小数点表記に変換されます(dddは１つ以上の10進数)。 ここで、小数点以下の桁は四捨五入されて、精度指定と等しくなります。
「e, E」	doubleの引数が、「〔－〕d.dd…de±dd」(eのとき)または「〔－〕d.…dE＋dd」(Eのとき)の形に変換されます(dは一つ、dd…dは一つ以上の10進数、ddは二つの10進数)。ここで、小数点以下の桁は四捨五入されて精度指定に等しくなります。

「g, G」	doubel引数が、「f」または「e」(「G」ならば「E」)の形に変換されます。精度指定は、有効数字桁数を示します。「e」(または「E」)が選択されるのは、値の指数が−4未満または精度指定よりも大きい場合です。精度指定に伴って付加される小数点以下の後方の「0」は、結果から除かれます。小数点は、後ろに数字が続くときだけに付きます。
「c」	整数引数を1文字として出力されます。
「s」	引数は文字列へのポインタで、文字列終端の「¥0」(NULL)文字の一つ前まで、または精度で指定された文字数に達するまでが出力されます。

注) 実数型(浮動小数点型、倍精度浮動小数点型)以外のargumentに対して、「f」、「e, E」、「g, G」の実数型を表示する変換文字を使用すると、「#NAN」が出力される場合があります。

scanf
fscanf
sscanf

(書式)

```
int          scanf(format,[, argument……]);
int          fscanf(stdin, format[, argument……]);
int          sscanf(buffer, format[, argument…]);
char         *format;
char         *buffer;
extern FILE  *stdin
```

(戻り値)

代入されたargumentの個数(0もあり得ます)を返します。変換に先立って終端記号(scanf、fscanfはEOF、sscanfは文字列終端の「¥0」(NULL)文字)を読み込んだ場合は、EOF(−1)を返します。int型。

(機能)

入力したデータをformatに従って変換し、argumentに代入します。

scanfはキーボードから、fscanfはstdin(キーボード)から、sscanfはbufferからデータを入力します。

argumentは、formatで指定された型に対応する型の変数を指すポインタです。

scanfとfscanfは、↵キーを押すと入力されます。sscanfは、「¥0」(NULL)文字がbufferの終わりと見なされます。

formatは0文字以上の文字列で、空白文字、普通文字、変換仕様からなります。変換仕様が現れると、その変換仕様に従って変換された値が後続のargumentに順番に代入されます。

argumentが変換仕様よりも多いときは、余分なargumentは無視されます。少ないときは、結果が不定となります。

SHIFT ↵と操作すると、EOF(−1)が入力されます。

(例)

```
int     i;
float   f;
double d;

scanf("%d%f%lf", &i, &f, &d);
```

1 2 3 ↵ − 1 . 2 3 E
1 0 ↵ 2 0 3 ↵と入力すると、
iに「123」、fに「−1.23e10」、dに「203.0」が代入されます。

変換仕様の書式

●**空白文字(スペース、改行)**

初めに空白文字でない文字を見つけるまで、入力文字を読み込みます(この文字は、まだ読み込まないままにします)。空白文字でない文字を見つけられない場合は、この関数は終了します。

●普通文字(空白文字以外で、「%」文字以外の文字)

入力文字を読み、この普通文字と一致しなければこの関数は終了し、入力文字はまだ読み込まないままにします。

●変換仕様

変換仕様は、「%」で始まる次の文字列です。

%〔*〕〔フィールド幅〕〔l〕変換文字

ただし、「%」の後に変換仕様として意味のない文字が続く場合は、その文字以降、次の「%」の一つ前までは、普通文字として扱われます。

「%」を入力するには「%%」とします。

変換仕様は、次のステップの順に実行されます。

①入力された空白文字(スペース、改行文字)は、スキップされます。ただし、変換文字「c」の場合を除きます。

②入力フィールドが読み込まれます。入力フィールドは、最初の空白文字まで、または変換仕様で変換できない文字まで、またはフィールド幅が指定されていれば指定されたフィールド幅までのどれかが最初に起きるまでの入力データです。入力フィールドの次の文字は、まだ読まれていないものと見なされます。もし、入力フィールドの長さが「0」ならば、この関数は終了します。

③入力フィールドは、変換文字により定まる型に変換されます。

④代入禁止「*」が指定されていると入力フィールドは読み込まれますが、変換結果はargumentに代入されません。

変換仕様の文字列

＊(代入禁止)

入力フィールドは読み込まれますが、変換結果のargumentへの代入は行なわれません。

フィールド幅

最大フィールド幅を符号なしの10進整数で指定します。省略時は、入力データの長さすべてです。

l(倍長整数型、倍精度浮動小数点型指定)

変換文字により、次のような意味を持ちます。

「d, o, u, x」	引数がlong型であることを指定します。
「e, f, g」	引数がdouble型であることを指定します。
その他	その他の文字が書かれると、「l」は無視されます。

変換文字

「d」	10進整数（符号が付いていてもよい）とマッチします。 対応する引数は、整数型変数へのポインタでなければなりません。
「o」	8進整数（符号が付いていてもよい）とマッチします。 対応する引数は、整数型変数へのポインタでなければなりません。

「u」　符号なしの10進整数とマッチします。
対応する引数は、整数型変数へのポインタでなければなりません。

「x」　16進整数（符号が付いていてもよい）とマッチします。
対応する引数は、整数型変数へのポインタでなければなりません。

「e, f, g」　浮動小数点数（符号が付いていてもよい）とマッチします。
対応する引数は、実数型(浮動小数点型)変数へのポインタでなければなりません。
指数の指数部の数字列の入力が3桁以上のときは、結果が不定となります。

「s」　非空白文字からなるシーケンスにマッチします。
対応する引数は、このシーケンスと終端の「¥0」(NULL)文字を収容するのに十分な大きさを持つ文字配列へのポインタでなければなりません。
終端の「¥0」(NULL)文字は、自動的に付加されます。

「c」　フィールド幅で指定された個数の文字からなるシーケンスにマッチします（フィールド幅指定がない場合は、「1」と見なされます)。
対応する引数は、この文字列を収容するのに十分な大きさを持つ文字配列へのポインタでなければなりません。空白文字は、スキップされないで読み込まれます。

scanf関数などで使われる変換指定子は、printf関数などで使われるものとほとんど同じです。ただし、printf関数などでは、倍精度浮動小数点型（double型）と単精度浮動小数点型（float型）に変換指定子「%f」を使いますが、scanf関数などではdouble型(変換指定子%lf)かfloat型(変換指定子%f)を指定する必要があります。これは、指定したメモリーの場所に正しい大きさのデータを格納するためです。

次に変換指定子を示します。

データ型	変換指定子
char	%c
int	%d
long	%ld
float	%f
double	%lf

getch

（書式）
int getch( );

（戻り値）
読み込んだ1文字のコード。
int型。

（機能）
キーボードから直接1文字読み込み、その文字コードを返します。読み込んだ文字は、表示されません。
入力バッファが空の場合は、次の入力があるまでカーソルが点滅して待機します。

（例）
```
int x;
  ⋮
x＝getch( );
```

kbhit

（書式）
int kbhit( );

（戻り値）
キー入力がある（入力バッファが空でない）場合は0以外を、キー入力がない（入力バッファが空）場合は0を返します。

（機能）
キー入力を調べる関数です。
この関数が呼ばれた時点の入力バッファの状態を調べます。

（例）
```
int kb;
  ⋮
kb＝kbhit( );
  ⋮
```

fflush

（書式）
int fflush(stdin);
extern FILE *stdin;

（戻り値）
正常ならば、0。int型。

（機能）
入力バッファの内容をクリアーします。ただし、入力バッファ内のEOFは、クリアーできません。

（例）
```
extern FILE *stdin;
int          c;
  ⋮
fflush(stdin);
c＝getc(stdin);
```

clearerr

（書式）
void clearerr(stdin);
extern FILE *stdin;

（戻り値）
何も返しません。

（機能）
入力バッファ内のEOFをクリアーします。

beep

（書式）
```
void        beep(n);
unsigned int n;
```

（戻り値）
何も返しません。

（機能）
nの指定により、低い音か高い音を鳴らします。
0：低い音を鳴らします。
1：高い音を鳴らします。

（例）
```
beep(0);
```

inport

（書式）
```
int inport(n);
int n;
```

（戻り値）
読み込んだ値。int型。

（機能）
nで指定されたI/Oポート(30ピンI/O)から1バイトのデータを読み込みます。
nの有効範囲は、0～7です。

outport

（書式）
```
void outport(n, data);
int  n, data;
```

（戻り値）
何も返しません。

（機能）
nで指定されたI/Oポート(30ピンI/O)に1バイトデータのdataを出力します。
nの有効範囲は、0～7です。
dataの有効範囲は、0～255です。

inp

（書式）
```
int         inp(n);
unsigned int n;
```

（戻り値）
I/Oポート(40ピンI/O)から読み込んだ値。int型。

（機能）
nによって指定されたI/Oポート(40ピンI/O)から1バイトデータを読み込みます。
nの有効範囲は、0～65535です。

out

（書式）

```
void          out(n, data, size) ;
unsigned int n ;
unsigned int data ;
int           size ;
```

（戻り値）
何も返しません。

（機能）
nによって指定されたI/Oポート(40ピンI/O)に1バイトデータのdataを出力します。
引数の有効範囲は以下のようになります。
n　：0～65535
size：1または2
data：0～255　(sizeが1の場合)
　　0～65535(sizeが2の場合)

実行制御関数

breakpt

（書式）

```
void breakpt( );
```

（戻り値）
何も返しません。

（機能）
プログラムの実行を停止し、ブレークモードに入ります。

（例）

```
main( )
{
  int i=0, j=10;
  while(i<=10) {
    breakpt( );
    i++;
    j--;
  }
}
```

exit	(書式) void exit( );	(戻り値) 何も返しません。
(機能) プログラムを正常終了させます。		

abort	(書式) void abort( );	(戻り値) 何も返しません。
(機能) プログラムを異常終了させます。「Abort」というメッセージを表示画面に出力します。		

コラム●ブレークモード●

breakpt関数が実行されると「Break ?」というメッセージを表示し、プログラムの実行を停止します。このとき、以下のようなキー操作が行なえます。

[↵]キー：実行を再開します。
[T]キー：トレースモードに入り、実行を再開します。
[N]キー：トレースモードを解除して、実行を再開します。
[BRK]キー：実行を中断します(プログラムが終了します)。
[D]キー：ブレークモードのままで、変数モードに入ります。
(変数モードで[↵]キーのみを押すと、ブレークモードに戻ります)

変数モードは、変数名を入力することによって、ブレークモード指定時のプログラムに使われている変数の型と値を表示するモードです。

このモードに入ると「var>_」と表示され、次のキー操作が行なえます。

変数名[↵]：指定した変数の型と、その時点の値を表示します。
[↵]のみ　：ブレークモードに戻ります。

breakpt関数のプログラム例を実行したときの操作例を示します。

キー操作	表　　示
RUN	Break ?_(プログラムを実行して、ブレークモードに入ります)
D	var>_　(変数モードに入ります)
I [↵]	int：0　(変数 i の型と値を表示します)
J [↵]	int：10　(変数 j の型と値を表示します)
[↵]	Break ?_(ブレークモードに戻ります)

malloc

(書式)

```
char        *malloc(size);
unsigned int size;
```

(戻り値)

メモリーが確保できた場合は、確保したメモリー領域のポインタを返します。確保できなかった場合は、NULLを返します。char型。

(機能)

sizeで指定したバイトサイズのメモリー領域を確保します。

確保されたメモリー領域は、プログラムの実行終了とともに解放されます。

(例)

```
main( )
{
        char *c;
            ⋮
        if((c=malloc(256))==NULL){
          printf("データリョウイキガ
トレマセン¥n");
          exit( );
        }
        ⋮
}
```

calloc

(書式)

```
char        *calloc(n, size);
unsigned int n;
unsigned int size;
```

(戻り値)

メモリーが確保できた場合は、確保したメモリー領域のポインタを返します。確保できなかった場合は、NULLを返します。char型。

(機能)

sizeで指定したバイトサイズのn個の要素を持つ配列をメモリー領域に確保し、「00H」で初期化します。

確保されたメモリー領域は、プログラムの実行終了とともに解放されます。

(例)

```
main( )
{
    int *iarry, i;
        :
    it((iarry=(int*)calloc(1000, 2))
==NULL){
      printf("ハイレツガ　トレマセン
¥n");
      exit( );
    }
    for(i=0;i<1000;i++) iarry[i]
=1;
     :
}
```

free

(書式)

```
int   free(ptr);
char *ptr;
```

(戻り値)

解放されると0を返します。ptrが無効だと(malloc、callocによって確保されたメモリー領域のポインタでないと)、−1を返します。int型。

(機能)

mallocやcallocで確保されたメモリー領域を解放します。

ptrは、malloc、callocによって確保されたメモリー領域のポインタでなければなりません。

(例)

```
char *arry;
     :
arry=malloc(256);
     :
free(arry);
```

文字列関数

strlen

（書式）
```
int    strlen(string);
char  *string;
```

（戻り値）
文字列stringの「¥0」(NULL)文字を含まない長さを返します。int型。

（機能）
文字列stringの「¥0」(NULL)文字の直前までのバイト数を返します。

（例）
```
int length;
    :
length=strlen("abc");
                    /*length=3*/
```

strcpy

（書式）
```
char  *strcpy(dest, source);
char  *dest, *source;
```

（戻り値）
destのポインタ。
char型。

（機能）
文字列sourceの先頭から「¥0」(NULL)文字まで(「¥0」(NULL)文字を含む）の範囲を、文字列destにコピーします。
コピーするとき、オーバーフローのチェックは行ないません。

（例）
```
char  *result, string[64];
    :
result=strcpy(string, "abc");
                    /*string="abc"*/
```

strcat

（書式）
```
char strcat(dest, source);
char  *dest, *source;
```

（戻り値）
destのポインタ。
char型。

（機能）
文字列sourceの先頭から「¥0」(NULL)文字まで(「¥0」(NULL)文字を含む)の範囲を文字列destの最初の「¥0」(NULL)文字の後に付加します。
付加のとき、オーバーフローチェックは行ないません。

strcmp

（書式）

```
int    strcmp(string1, string2);
char *string1 , *string2;
```

（戻り値）

次の整数値を返します。
int型。
string1＜string2→負整数
string1＝string2→ 0
string1＞string2→正整数

（機能）

文字列string1と文字列string2の先頭から1文字ずつ、「¥0」(NULL)文字が現われるまで文字コード比較(ASCIIコード順)をします。
「¥0」(NULL)文字も比較の対象となります。

（例）

```
int   result;
char string1[5];
char string2[5];
    :
strcpy(string1, "abcde");
strcpy(string2, "abcda");
result=strcmp(string1, string2);
                        /*result=1*/
```

strchr

（書式）

```
char *strchr(string, c);
char *string;
int   c;
```

（戻り値）

検索文字cが文字列string中で見つかった場合は、stringのcがある場所のポインタを返します。
見つからなかった場合は、NULLを返します。
char型

（機能）

文字列stringで最初に現われる指定文字cを検索します。
「¥0」(NULL)文字も検索対象となります。

（例）

```
main( )
{
  char instr[64], *ss;

  printf("Input string=");
  gets(instr);
  ss=instr;
  while((ss=strchr(ss,'*'))!=NULL)
  /*アスタリスク ノ ケンサク*/
  *ss='_';
  /*アンダーバー ニ オキカエ*/
  puts(instr);
}
```

数値関数

abs

(書式)

```
int abs(n);
int n;
```

(戻り値)

nの絶対値。
int型。

(機能)

整数の絶対値を返します。

(例)

```
main( )
{
  int i＝－1, ans;
        ⋮
    ans＝abs(i);
        ⋮
}
```

sin cos tan

(書式)

```
double sin(x);
double cos(x);
double tan(x);
double x;
```

(戻り値)

double型。

(機能)

角度xに対する三角関数の値を返します。

演算範囲を超えた場合は、エラーになります。

角度xの有効範囲は、次のとおりです。

|x|＜1440(DEG)
|x|＜8π (RAD)
|x|＜1600(GRA)

(例)

```
main( )
{
  double y;

  angle(0);
  for(;;){
    printf("Angle ?");
    scanf("%lf", &y);
    printf("%11.10g ¥n %11.10g ¥n
%11.10g ¥n", sin(y), cos(y), tan(y));
  }
}
```

asin
acos
atan

（書式）

```
double asin(x);
double acos(x);
double atan(x);
double x;
```

（戻り値）

double型。

（機能）

xに対する逆三角関数の値（角度）を返します。

演算範囲を超えた場合は、エラーになります。

xの有効範囲は、次のとおりです。

$|x| \leqq 1$（asin, acosの場合）

$|x| < 1E+100$（atanの場合）

返される関数の値の範囲は次のとおりです。

（RADの場合）

$\left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$（asin, atanの場合）

$[0, \pi]$（acosの場合）

DEG, GRAもRADに準じます。

（例）

```
double y;
  :
angle(1);
y=asin(1.0);
y=acos(1.0);
y=atan(1.0);
```

sinh
cosh
tanh

（書式）

```
double sinh(x);
double cosh(x);
double tanh(x);
double x;
```

（戻り値）

double型。

（機能）

xに対する双曲線関数の値を返します。

$\sinh x = (e^x - e^{-x})/2$

$\cosh x = (e^x + e^{-x})/2$

$\tanh x = (e^x - e^{-x})/(e^x + e^{-x})$

演算範囲を超えた場合は、エラーになります。

xの有効範囲は、次のとおりです。

$|x| \leqq 230.2585092$（sinh, coshの場合）

$|x| < 1E+100$、$|\tanh(x)| < 1$（tanhの場合）

（例）

```
double y;
  :
y=sinh(1.0);
y=cosh(1.0);
y=tanh(1.0);
```

asinh
acosh
atanh

（書式）

```
double asinh(x);
double acosh(x);
double atanh(x);
double x;
```

（戻り値）

double型。

（機能）

xに対する逆双曲線関数の値を返します。

asinh $x=\sinh^{-1}x=\log_e(x+\sqrt{x^2+1})$

acosh $x=\cosh^{-1}x=\log_e(x+\sqrt{x^2-1})$

atanh $x=\tanh^{-1}x=\{\log_e(1+x)-\log_e(1-x)\}/2$

演算範囲を超えた場合は、エラーになります。

xの有効範囲は、次のとおりです。

|x|＜5E＋99（asinhの場合）

1≦x＜5E＋99（acoshの場合）

|x|＜1　　（atanhの場合）

（例）

```
double y;
    ⋮
y=asinh(1.0);
y=acosh(2.0);
y=atanh(0.5);
```

pow

（書式）

```
double pow(x, y);
double x, y;
```

（戻り値）

double型。

（機能）

xのy乗（x^y）の値を返します。

yが0の場合は、1を返します。

xが0でyが0または負の場合と、xが負でyが整数でない場合は、エラーになります。

その他、演算範囲が超えた場合は、エラーになります。

（例）

```
double x=2.0, y=3.0, z;
    ⋮
z=pow(x, y);
```

sqrt

（書式）

```
double sqrt(x);
double x;
```

（戻り値）

double型。

（機能）

xの平方根（$\sqrt{x}$）を返します。

xが負の場合は、エラーになります。

その他、演算範囲が超えた場合は、エラーになります。

（例）

```
double y;
    ⋮
y=sqrt(2.0);
```

exp

（書式）
double exp(x);
double x;

（戻り値）
double型。

（機能）
xの指数関数(e^x)の値を返します。
x＞230.2585092の場合、エラーになります。
その他、演算範囲を超えた場合は、エラーになります。

（例）
```
double y;
    ⋮
y=exp(1.0);
```

log
log10

（書式）
double log(x);
double log10(x);
double x;

（戻り値）
double型。

（機能）
logは、xの自然対数($\log_e x$)の値を返します。
log10は、xの常用対数($\log_{10} x$)の値を返します。
xが0または負の場合は、エラーになります。
その他、演算範囲を超えた場合は、エラーになります。

（例）
```
double y;
    ⋮
y=log(1000.0);
y=log10(1000.0);
```

angle

（書式）
void angle(n);
unsigned int n;

（戻り値）
何も返しません。

（機能）
三角関数、逆三角関数の角度モードを指定します。nの指定により、次のようになります。
0：DEG（度）
1：RAD（ラジアン）
2：GRA（グラード）

（例）
```
double y;
    ⋮
angle(0);
y=sin(90.0);
angle(1);
y=sin(1.570796327);
angle(2);
y=sin(100.0);
```

画面制御関数

clrscr

（書式）
```
void clrscr( ) ;
```
（戻り値）
何も返しません。

（機能）
表示画面の表示を消去し、カーソルをホームポジションに戻します。

（例）
```
clrscr( ) ;
```

gotoxy

（書式）
```
void        gotoxy(x, y) ;
unsigned int x ;
unsigned int y ;
```
（戻り値）
何も返しません。

（機能）
仮想スクリーン(32桁×８行)上のカーソル位置を、ｘとｙで指定します。
座標は仮想スクリーン上の左上隅を原点(０,０)とし、次の範囲で指定できます。
xの範囲は、０～31です。
yの範囲は、０～７です。

（例）
```
gotoxy(10, 2) ;
```

line

（書式）
```
void        line(x1, y1, x2, y2) ;
unsigned int x1 ;
unsigned int y1 ;
unsigned int x2 ;
unsigned int y2 ;
```
（戻り値）
何も返しません。

（機能）
仮想スクリーン(192×64ドット)上に、始点(x1, y1)から終点(x2, y2)まで直線を描きます。
始点(x1, y1)と終点(x2, y2)が一致した場合は、点指定になります。
座標は仮想スクリーン上の左隅を原点(0, 0)とし、以下の範囲で指定できます。
x1, x2の範囲は、０～191です。
y1, y2の範囲は、０～63です。

（例）
```
line(10, 10, 50, 50) ;
```

linec

（書式）

```
void          linec(x1, y1, x2, y2);
unsigned int x1;
unsigned int y1;
unsigned int x2;
unsigned int y2;
```

（戻り値）

何も返しません。

（機能）

仮想スクリーン（192×64ドット）上に描かれている直線のうち、始点(x1, y1)から終点(x2, y2)までの直線を消します。

始点(x1, y1)と終点(x2, y2)が一致した場合は、点指定になります。

座標は仮想スクリーン上の左上隅を原点(0, 0)とし、以下の範囲で指定できます。

x1, x2の範囲は、0～191です。

y1, y2の範囲は、0～63です。

（例）

```
linec(10, 10, 50, 50);
```

getpixel

（書式）

```
int           getpixel(x, y);
unsigned int x;
uhsigned int y;
```

（戻り値）

指定された位置のドットが点灯している場合は1、消灯している場合は0を返します。int型。

（機能）

仮想スクリーン上のドットの点灯・消灯を調べます。

xの範囲は、0～191です。

yの範囲は、0～63です。

（例）

call

（書式）

```
unsigned long  call(adr, esg) ;
unsigned int   adr ;
unsigned int   seg ;
```

（戻り値）

CPUレジスタDX，AXの内容を、unsigned long型の一つの値（DXはHigh側（2バイト）、AXはLow側（2バイト））にして返します。

（機能）

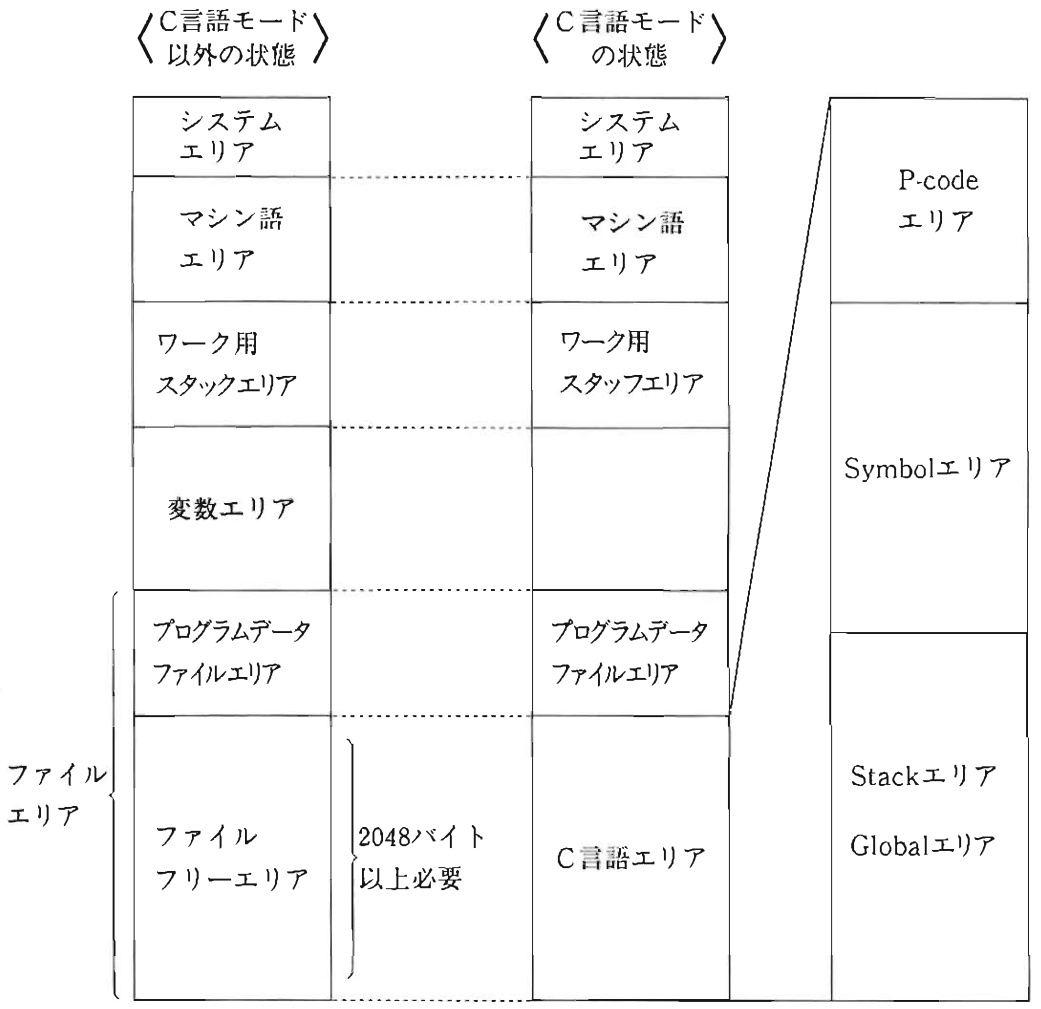
メモリー内に用意されたマシン語サブルーチンに制御します。

マシン語サブルーチンの実行開始番地は、以下のようになります。

実行開始番地＝adr＋seg＊16

マシン語サブルーチンからは、マシン語のIRET命令により戻ります（C言語プログラムに制御が戻ります）。

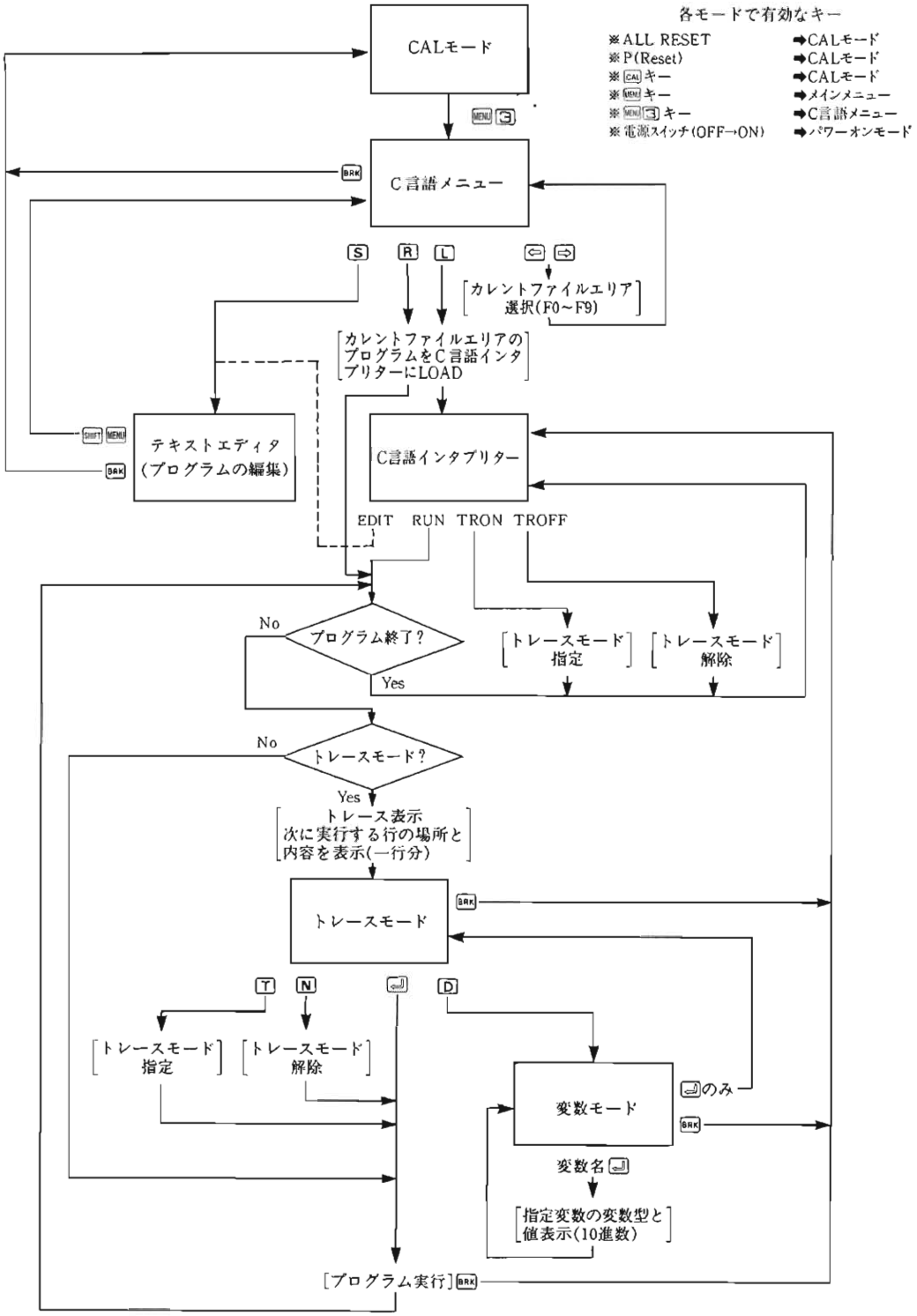
メモリーマップ



●本機のC言語インタプリターへロードできるファイル(プログラム)の大きさは、65520バイト以下です。
また、C言語を動作するには、ファイルフリーエリアが最低2048バイト必要です。

●C言語エリアは、使用時に次のような比率に分けられます。
P-codeエリア：Symbolエリア：Stackエリア＋Globalエリア＝2：3：3

C言語状態遷移図


各モードで有効なキー
※ALL RESET ➡CALモード
※P(Reset) ➡CALモード
※CALキー ➡CALモード
※MENUキー ➡メインメニュー
※MENU 3 キー ➡C言語メニュー
※電源スイッチ(OFF→ON) ➡パワーオンモード
CALモード
MENU 3
C言語メニュー
BRK
S
R
L
カレントファイルエリア
選択(F0～F9)
カレントファイルエリアの
プログラムをC言語インタ
プリターにLOAD
SHIFT MENU
BRK
テキストエディタ
(プログラムの編集)
C言語インタプリター
EDIT RUN TRON TROFF
No
プログラム終了?
Yes
トレースモード
指定
トレースモード
解除
No
トレースモード?
Yes
トレース表示
次に実行する行の場所と
内容を表示(一行分)
トレースモード
BRK
T
N
D
トレースモード
指定
トレースモード
解除
変数モード
のみ
BRK
変数名
指定変数の変数型と
値表示(10進数)
[プログラム実行] BRK

C言語エラーメッセージ一覧表

エラーメッセージ	エラー内容	対処方法
'演算子' illegal lvalue	この演算子のオペランドは、左辺値でなければなりません。	オペランドを確認してください。
'演算子' illegal operand	演算子のオペランドの型が正しくありません。	オペランドの型を確認してください。
'演算子' incompatible types	式中またはreturnの戻り値に、互換性のない型があります。	オペランドの型を確認してください。
'識別子' not a function	関数として宣言されていない識別子を、関数として使おうとしました。	識別子を確認してください。
'識別子' not a variable	変数として宣言されていない識別子を、変数として使おうとしました。	識別子を確認してください。
'識別子' redefined	識別子が二重定義されています。 識別子が正しく定義されていません。	識別子を確認してください。 すでに定義されているものと同じ場合は、別の名前にしてください。
'識別子' undefined	識別子が未定義です。	識別子を確認してください。 定義されていなければ、その定義を追加してください。
'main' not found	main関数が見つかりません。	main関数を含むファイルをロードしてください。
Abort	ライブラリー関数abortにより実行が中断されました。	
Argument list too long	引数リストが、256バイトを超えました。	引数の数を少なくしてください。
Arithmetic overflow	浮動小数点定数が、表現できる値の範囲を超えました。 計算結果が許される範囲を超えました。	浮動小数点定数および計算式を確認してください。
Buffer overflow	256文字以上入力し、[IN]キーを押しました。	1行255文字以下にしてください。
C undefined error	ハードウェア上の不具合、またはC言語プログラムを実行したときのメモリーの異常により発生する可能性があるエラーです。	ALL RESETボタンを押してください。

エラーメッセージ	エラー内容	対処方法
File full	ソースプログラムが容量オーバーです。(ソースプログラムが65521バイト以上です)。	ソースプログラムの容量を減らしてください。
Global area overflow	C言語エリア内のGlobalエリアがオーバーフローしました（外部定義の変数と関数内のauto変数、static変数の合計サイズが大きすぎます)。	clear文を用いるか、RAMを増設しC言語エリアを増やします。
Illegal 'case'	caseラベルが、switch文の外側にあります。	caseラベルの位置を確認してください。
Illegal 'case' constant	caseラベルの定数指定に誤りがあります。	caseラベルの定数を確認してください。
Illegal 'default'	defaultのラベルが、switch文のブロック外にあります。	defaultのラベルの位置を確認してください。
Illegal 'main'	main関数に引数は指定できません。	main関数の引数を取ってください。
Illegal 'switch' value	switch文の式の値が正しくありません。	式の値を正しく直してください。
Illegal argument	ライブラリー関数の引数が許容範囲を超えています。	引数を確認してください。
Illegal argument DCL	関数は仮引数として宣言できません。 関数の仮引数中にない引数が、宣言されています。	関数定義の引数宣言を確認してください。
Illegal array DCL	配列宣言で要素数の指定が整数値でありません。 配列のサイズが大きすぎます(65535バイトを超えています)。 最初の次元以外の次元の要素数の指定がありません。 サイズを取っていない配列を、ローカル変数として宣言しようとしました。	配列宣言を確認してください。
Illegal break	break文がwhile、do～while、for文またはswitch文のいずれかの文中以外で使用されています。	break文はwhile、do～while、for文またはswitch文の文中で使用するようにしてください。
Illegal cast	キャストの型指定が正しくありません。	キャストの型名を確認してください。

エラーメッセージ	エラー内容	対処方法
Illegal char constant	文字定数が1文字の表示可能文字、またはエスケープシーケンス以外です。 文字定数がシングルクォーティションでくくられていません。	文字定数を確認してください。
Illegal command	コマンドの書式に誤りがあります。	コマンドの書式を再確認してください。
Illegal continue	continue文がwhile、do～while文またはfor文のいずれかの文中以外で使用されています。	continue文はwhile、do～while文またはfor文の文中で使用するようにしてください。
Illegal escape sequence	¥の後の文字が、有効なエスケープシーケンスでありません。	エスケープシーケンスを確認してください。使用できるエスケープシーケンスは¥0(00h)・¥a(07h)・¥b(08h)・¥t(09h)・¥n(0Ah)・¥f(0Ch)・¥r(0Dh)・¥"・¥'・¥¥・¥ooo・¥xhhです。
Illegal function DCL	キーワード、ライブラリー関数および内部システムで使用している関数と同じ名前で関数定義や宣言はできません。	関数定義または宣言を確認してください。 別の名前を使用してください。
Illegal function call	関数を呼び出すとき、引数の個数が関数定義の仮引数の個数と一致しません。 または、引数の型を仮引数の型に変換できません。	引数の個数と型を確認してください。
Illegal index	配列の添字の値が負、または定義されている要素数を超えました。	添字の指定を確認してください。
Illegal initialization	関数は初期化できません。 初期化する変数と定数の型が異なります(要素並びの配列の初期化はできません)。	初期化を確認してください。
Illegal pointer DCL	ポインタ宣言は、＊を1つだけしか指定できません。 void型指定子のポインタ宣言はできません。	ポインタ宣言を確認してください。
Illegal preprocessor	プリプロセッサの構文エラーです。	＃include文、＃define文を確認してください。
Illegal sizeof	sizeof演算子のオペランドは、識別子でなければなりません。	sizeofのオペランドを確認してください。

エラーメッセージ	エラー内容	対処方法
Illegal storage class	この部分では、指定の記憶クラスは使用できません。	記憶クラスを確認するか、指定位置を確認してください。
Illegal stream pointer	ストリームポインタが正しくありません。 読み込みのためのストリームに書き込もうとしました。 書き込みのためのストリームから読み込もうとしました。	ストリームポインタを確認してください。
Illegal string	文字定数がダブルクォーティションでくくられていません。 文字列が255文字を超えています。	文字列を確認してください。
Illegal type	指定できない型指定子を使用しました。 データ定義で型指定子にvoidを指定しました。	型指定子を確認してください。
Mathematical error	0による割り算などの数学的エラーです。 ライブラリー関数の引数が演算範囲を超えています。 float/double型のデータのフォーマットエラーです。	計算式または数値を確認してください。
Memory overflow	ファイルエリアがオーバーフローしました。	ファイルエリアの容量を増やしてください。ファイルフリーエリアは最低2048バイト必要です。
P-code area overflow	C言語エリア内のP-codeエリアがオーバーフローしました（プログラムが大きすぎます）。	clear文を用いるか、RAMを増設しC言語エリアを増やしてください。
Stack area overflow	C言語エリア内のStackエリアがオーバーフローしました（再帰呼び出しなどにより、引数リストと内部定義の変数が取られているスタックが足りなくなりました）。	clear文を用いるか、RAMを増設しC言語エリアを増やしてください。
Symbol area overflow	C言語エリア内のsymbolエリアがオーバーフローしました（識別子、定数が多すぎます）。	clear文を用いるか、RAMを増設しC言語エリアを増やしてください。

エラーメッセージ	エラー内容	対処方法
Syntax error	構文エラーです。 配列の要素数に、定数以外を指定しました。	構文および識別子を確認してください。
System stack overflow	複数のレベルが深すぎて、システムスタックがオーバーフローしました。または、関数の再帰呼び出しにより、システムスタックがオーバーフローしました。	ネスティングを浅くしてください。
Too complex expression	式が複雑すぎます。	式を簡単にするか分割してください。
Too many 'case's	caseのラベルの数が多すぎます。	caseのラベルの数を減らしてください。
Too many 'default's	defaultのラベルが複数個存在します。	一つのswitch文内のdefaultのラベルを一つにしてください。
Too many breaks /continues	while、do～while、for文またはswitch文中のbreak文とwhile、do～while文またはfor文中のcontinue文の合計が40を超えました。	break文とcontinue文の合計を40以内にしてください。
Too many initializers	初期化される変数の数が多すぎます。	初期化される変数の数を減らしてください。
Unknown character	識別子などに本機では許されていない文字が使用されました。	使用している文字を確認してください。

第2章
BASIC編

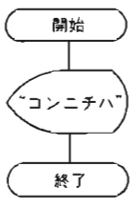
プログラムの構成

ここで学ぶこと
☆BASICプログラムの構成

まとめ
◎BASICプログラムは、入力文、出力文、演算文で構成されます。
◎一つの文は、行番号、コマンド、オペランドで構成されます。
◎プログラムの文には、実行される文と、実行されない文があります。

プログラム例

```
10 REM ﾒREI ｺﾝﾆﾁﾊ ﾉ ﾌﾟﾛｸﾞﾗﾑﾒ
20 PRINT ﾒｺﾝﾆﾁﾊﾒ
30 END
```



プログラムフローチャート

プログラムを作るためには、いろいろな文法や規則があります。文法などの詳しい説明は後にして、まずBASICプログラムがどのような形になっているのかを見てみましょう。

プログラムの基本

実行される命令文は、基本的に三つの文に分けられます。

(1)データを入力するための入力文
(2)そのデータに基づいて演算などの処理をする演算文
(3)実行の結果を出力するための出力文

すなわち、データを入力し、そのデータを演算などの処理で加工し、加工された実行結果を出力する……という一連の作業が、プログラムの基本になるわけです。

文(行)の基本構成

BASICプログラムの各文(行)は、基本的な約束ごとにしたがって構成されています。

プログラムを書いたり実行したりするためには、各文(行)に名前を付けておく必要があります。BASICでは、各文(行)に番号を付けて、それを名前とします。これが「行番号」です。

プログラムは行番号の小さい順から実行されます。行番号は1～65535までの整数です。行番号は10、20、30…のように10きざみで付けると、後でプログラムの行を追加するときに便利です。

また、プログラムの入力は行番号順に行なう必要はありません。入力されたプログラムはコンピュータ内で自動的に行番号順に並べかえられます。

行番号に続いて計算機に命令する「コマンド」に必要な情報を示す「オペランド」を記述します。

実行文と非実行文

BASICプログラムの文には、入出力や演算などの処理をする実行文と、処理をしない非実行文とがあります。プログラム例の行番号10のREM文は非実行文の一つで、注釈(コメント)の役割をしています。

インタプリターの操作

ここで学ぶこと
☆プログラムを入力するための準備

まとめ
◎本機でBASICを使用する前に、使用するプログラムエリアを指定します。
◎使用するプログラムエリアにプログラムが入力されている場合は、NEWコマンドでプログラムを消去します。

プログラムエリア

これからプログラムを入力したり実行したりするための準備として、使用するプログラムエリアを指定します。

操作 スライドスイッチをONにして、本機の電源を入れます。
(モード設定が[CAL]モード時)

```
_
```

操作 MENUキーを押して、メニューを表示させます。

```
                ( MENU )

1:F.COM 2:BASIC 3:C     4:CASL
5:ASMBL 6:FX    7:MODE
```

操作 2キーを押すと、次のような表示になります。

```
P 0 1 2 3 4 5 6 7 8 9      18412B
Ready P0
_
```

これでBASICモードになり、プログラムの作成・編集・実行ができます。

表示の1行目の「P」はプログラムエリアを意味し、「0」から「9」までの数字は各プログラムエリアを示しています。「18412B」は、残りのメモリー容量を示しています(RAMパックを増設している場合や、テキストデータ、BASIC、C言語、CASL、アセンブリ言語のプログラムを記憶している場合は、この数値が異なります)。

2行目の「Ready P0」は、本機が現在「P0」エリアを使用していることを示しています。使用するプログラムエリアは、SHIFTを押しながら2～9の数字キーを押すことにより、変更することができます。

プログラムの消去

1行目の「0」から「9」までの数字の代わりに「＊」が表示されている場合は、そのプログラムエリアにプログラムがすでに入力されています。もし使用したいプログラムエリアが「＊」になっている場合は、「NEW」を指定してプログラムを消去しましょう。

右の表示は、プログラムエリアP0、1にすでに入力されているときのものです。

```
P ＊ ＊ 2 3 4 5 6 7 8 9      17625B
Ready P1
_
```

ここで[N][E][W][↵]と操作すると、「Ready」で示されたプログラムエリア(右の表示ではP1)の内容が消去されます。

```
P ＊ 1 2 3 4 5 6 7 8 9      17992B
Ready P1
_
```

また[N][E][W][A][L][L][↵]と操作すると、すべてのプログラムエリアの内容が消去されます。

プログラムの入力と実行

ここで学ぶこと
☆BASICプログラムの入力方法
☆入力されたプログラムを確認する方法
☆プログラムの実行方法

まとめ
◎プログラムエリアを指定してから、BASICプログラムを入力します。
◎EDITコマンドやLISTコマンドにより入力したプログラムを確認します。
◎RUNコマンドで、入力済みのプログラムを実行します。

簡単なプログラムを使って、プログラムの入力・編集・実行をする方法を説明します。ここではプログラムで使用している命令については考えず、操作方法だけをマスターしてください。

プログラムの入力

前述の「プログラムの構成」で示したプログラムを、プログラムエリア「P0」に入力してみましょう。

操作		表示	
①	1 0 SPACE R E M SPACE SHIFT " R E I SPACE SHIFT カナ K O N N N I T I H A SPACE N O SPACE P U R O G U R A M U SHIFT " SHIFT カナ ↵	①	10 REM "REI コンニチハ ノ プログラム"
②	2 0 SPACE SHIFT PRINT SHIFT " SHIFT カナ K O N N N I T I H A SHIFT " SHIFT カナ ↵	②	20 PRINT "コンニチハ"
③	3 0 SPACE E N D ↵	③	30 END

入力上の注意

行番号20のように、SHIFT PRINTと操作すると、PRINTという文字列を入力することができます。このようにして文字列を入力することを、「ワンキーコマンドを使う」といいます。ワンキーコマンドを使うと、入力されたコマンドの後ろに自動的に1個のスペースが付きます。

アルファベットは、BASICの基本として大文字で入力してください。大文字と小文字のモード切り換えは、CAPSキーを押します。本機のBASICでは、コマンドが小文字で入力されても大文字に変換されます。ただし、変数名や「"」(ダブルクォーテーション)で囲まれた文字列は、入力されたままになります。

プログラムの入力では、1行の最後に必ず↵(リターン)キーを押してください。

プログラムの確認

入力されたプログラムを確かめる方法として、次の2種類があります。

(1)LISTコマンドを使用する……プログラムを見る

SHIFT LIST ↵と操作すると、画面に次々とプログラムが表示されます。

```
20 PRINT "コンニチハ"
30 END
Ready P0
_
```

しかし、これでは速過ぎてプログラムを読むことができません。そこで↓キーや↑キーを使って、現在表示されている行以外の行を確認してください。

また、LISTコマンドで長いプログラムを表示中にSTOPキーを押すと、プログラムリストの表示を一時停止させることができます。これはプログラムを少しずつ見るときに便利です。次の部分を表示させるには↵キーを押します。

本機は8行分の仮想スクリーンを持っているので、現在表示されている行を含めて8行を↑↓キーで確認することができます。

L I S T行番号 ……指定した行番号を表示します。

L I S T行番号 ー 行番号……指定した範囲の行番号を表示します。

(2)EDITコマンドを使用する……プログラムを編集する

SHIFT EDIT ↵と操作すると、画面にプログラムが表示されます。この状態で、↵(前進)、SHIFT ↵(後進)、↑、↓キーを使って、他の行を確認することができます。

```
10 REM "REI コンニチハ ノ ﾌﾟﾛｸﾞﾗﾑ"
20 PRINT "コンニチハ"
30 END
```

プログラムの確認が終了したら、BRKキーを押します。「Ready P0」が表示され、作業が終了します。

プログラムリストを確認中にプログラムの入力ミスを発見した場合は、プログラムの修正を行ないます。↑↓←→キーを使って、カーソル(画面上で点滅している「_」)を修正したい部分に移動させて、修正後に↵キーを押します。このとき、文字と文字の間に文字を挿入したいときは、INSキーを押して間を空けます。逆に、文字を消去して、後ろの文字を前に詰めたいときは、DELキーを押します。修正方法については、次ページの「エディタの操作」を参照してください。

抜けている行を発見したときには、行番号とコマンド、オペランドを入力し、最後に↵キーを押すと、行の挿入ができます。不要な行を発見したときには、行番号を入力後に↵キーを押します。入力した行番号の文が1行分削除されます。

プログラムの実行

「Ready P0」が表示されているときにSHIFT RUN ↵と操作すると、プログラムエリア「P0」に入力されているプログラムが実行されます。

```
RUN
コンニチハ
Ready P0
_
```

エディタの操作

ここで学ぶこと
☆プログラムの修正方法
☆実行エラーの対処方法
☆プログラムのデバッグ方法

まとめ
◎プログラムの入力中またはLISTコマンドやEDITコマンドで確認中は、いずれもカーソルを移動させて修正します。
◎実行中のエラーは、LISTコマンドまたはEDITコマンドでプログラムを表示させてから修正します。
◎エラーメッセージの意味を理解します。
◎STOPコマンドやトレース機能でプログラムの誤りを発見します。

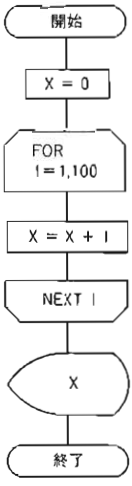
プログラムの修正を行なうのは、次の4つの場合です。
(1)プログラム入力中(↵キーを押す前)に間違いに気付いた。
(2)リスト確認中(↵キーを押した後)に間違いに気付いた。
(3)実行してみたらエラーが出た。
(4)エラーはないが、実行結果がおかしい。

このようにいろいろな場合がありますが、文の訂正、挿入、削除の方法は同じです。いずれの場合も、修正したい位置にカーソル(画面で点滅している「_」)を移動させて修正します。
プログラムリストの修正は、BASICモード(MENU 2)で行ないます。

プログラム例

```
10 X=0
20 FOR I=1 TO 100
30 X=X+I
40 NEXT I
50 PRINT X
60 END
```

上に示したプログラム例を使って、いろいろな修正方法を紹介します。このプログラムは、1から100までの整数の和を計算するものです。プログラムの内容の理解は後回しにして、いろいろな修正方法をしっかりと覚えてください。なお、プログラムエリアP1に「入力中」または「入力後」という設定で修正を行なうものとします。



プログラムフローチャート

入力中のプログラム訂正(↵キーを押す前)

プログラムの入力中(↵キーを押す前)に間違いに気付いたときは、カーソルを訂正部分に移動させ、修正します。

(例1) 30 X=X+Iを30 X=Y+Iと入力してしまった。

```
10 X=0
20 FOR I=1 TO 100
30 X=Y+I_
```

←←←
(カーソルを訂正部分に移動させます。)

```
10 X=0
20 FOR I=1 TO 100
30 X=Y+I
```

X↵
(正しい文字を入力し↵キーを押すと、この行が登録されます。)

```
10 X=0
20 FOR I=1 TO 100
30 X=X+I
_
```

入力後のプログラム訂正(↵キーを押した後)

LISTコマンドで訂正する方法

プログラムリストを入力後(↵キーを押した後)に間違いに気付いたときは、その行を表示させ、カーソルを訂正したい部分に移動させ、訂正します。

(例2) 30 X=X+Iを30 X=XX+Iと入力してしまったことに気付いた。

SHIFT LIST 3 0 ↵
(訂正したい行を表示します。)

```
30 X=XX+I
Ready P1
_
```

↑↑→→→→→
(カーソルを、訂正したい文字まで移動させます。)

```
30 X=XX+I
Ready P1
```

DEL ↵
(余分な文字を削除します。)

```
30 X=X+I
Ready P1
```

ここで、正しく訂正されたかどうかを確認してみましょう。

CLS
(表示されている文字をクリアーします。)

```
_
```

SHIFT LIST 3 0 ↵
(訂正された行を表示します。)

```
30 X=X+I
Ready P1
_
```

正しく訂正されたことが確認できます。
プログラムを訂正した後は、必ず↵キーを押してください。

EDITコマンドで訂正する方法

プログラムをEDITコマンドで表示させ、⇧⇩キーを使って修正したい行を画面の最上行に表示させます。次に⇦⇨キーで訂正したい部分にカーソルを移動させます。

（例３） 30 X＝X＋I を30 X＝＋Iと入力してしまったことに気付いた。

SHIFT EDIT ↵

```
10 X=0
20 FOR I=1 TO 100
30 X=+I
40 NEXT I
```

⇩⇩
（訂正したい行を1行目に表示させます。）

```
30 X=+I
40 NEXT I
50 PRINT X
60 END
```

⇨⇨⇨⇨⇨⇨
（カーソルを訂正したい部分に移動させます。）

```
30 X=±I
```

INS
（一つスペースを空けます。）

```
30 X=_+I
```

X ↵
（正しい文字を入力します。）

```
30 X=X+I
40 NEXT I
50 PRINT X
60 END
```

もし他にも訂正する箇所があれば、このまま⇧⇩⇦⇨キーを使って訂正したい行を画面の最上行に表示させ、カーソルを訂正させたい位置まで移動して、同様に訂正できます。

プログラムを訂正した後は、必ず↵キーを押してください。

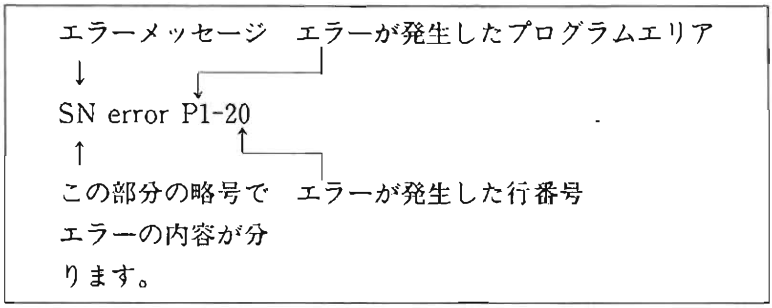
ご注意

◎LISTコマンドでは仮想スクリーン8行の中だけで訂正を行ないますが、EDITコマンドではプログラム全体を見ながら訂正できます。

EDITコマンドでは「まず⇧⇩キーを使って訂正させたい行を１行目に表示させ、次に⇦⇨キーを使ってカーソルを移動させる」ということに注意してください。

実行してエラーが出たときの訂正

実行時のエラーにはいろいろな種類があります。プログラムに不都合な点があると、RUNコマンド(SHIFT RUN ↵)を入力した後、次の例のようなエラーメッセージが表示されます。



この例では、「SN error」すなわち「コマンドまたは文の書式に誤りがある」ということになります(エラーメッセージの内容は、156ページの「エラーメッセージ一覧表」を参照してください)。LISTコマンドやEDITコマンドで間違いのある行を表示し、訂正します。訂正方法は前に説明した方法と同じです。

―コラム●プログラムのデバッグ●―

プログラムの誤りを「バグ」といいます。バグとは虫(BUG)のことで、この虫を取り除く作業のことを「デバッグ(DEBUG)」といいます。

本機ではエラーチェック機能を利用し、プログラムの流れやBASICプログラムの文法上の誤り(記述ミス)がないかを、表示を見ながらデバッグ作業を行なえます。

デバッグ作業には、「エラーメッセージによるデバッグ」、「STOPコマンドによるデバッグ」、「トレースモードによるデバッグ」の3種類があります。

●エラーメッセージによるデバッグ

キー入力やプログラムに誤りがあったり、プログラムの実行中になんらかの不都合が発生すると、エラーメッセージとエラーの発生した場所を表示して実行が中断されます。

ここで、SHIFT LIST .↵または L I S T .↵、SHIFT EDIT .↵またはE D I T .↵と操作すると、エラーの発生した行が表示されます。プログラムをよく検討し、訂正します。

●STOPコマンドによるデバッグ

STOP文をプログラム中に書き込んでおくと、そこでプログラムの実行が停止されます。調べたい変数などのある行の直後にSTOP文を書き込み実行させると、そこで停止しますので、

SHIFT PRINT 調べたい変数名↵またはP R I N T 調べたい変数名↵と操作して変数の内容を表示させます。実行を再開させるときは、SHIFT CONT↵またはC O N T↵と操作します。

●トレースによるデバッグ

トレース機能をONにすると、行番号を表示しながらプログラムを実行することができます。トレースONの指定は、

T R O N↵と操作します。トレース機能は、プログラムの流れが思い通りにいっているかどうかを「追跡」するのに用いると便利です。トレースONの解除は、

T R O F F↵と操作します。

その他、エラー表示が出力されずにプログラムは実行するが、実行結果が期待と違う場合は、原因として、

・引用している変数の間違い
・配列の引数の間違い
・指定すべき分岐先の間違い

などが考えられます。注意深くプログラムを確認し、修正し再実行してください。

条件分岐するプログラム

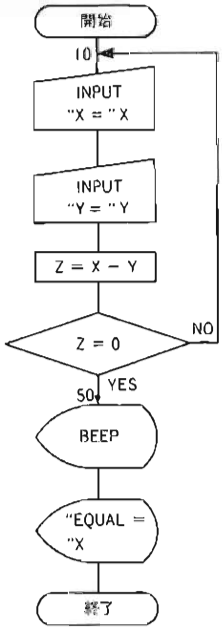
ここで学ぶこと
☆データの入力、表示
☆変数名の付け方
☆IF文による条件式判断

まとめ
◎INPUT文は、データ入力をします。
◎LET文は、代入の指定をします。
◎IF文は、条件判断により異なった実行をさせることができます。
◎BEEP文は、ブザーを鳴らします。
◎PRINT文は、データを表示します。
◎END文は、プログラムを終了します。

プログラム例

```
①  10 INPUT "X=";X
①  20 INPUT "Y=";Y
②  30 LET Z=X-Y
③  40 IF Z=0 THEN 50 ELSE 10
④  50 BEEP
⑤  60 PRINT "EQUAL=";X
⑥  70 END
```

（XとYの二つの数値を入力し、XとYの値が等しければブザーを鳴らしてEQUAL＝とXの値を表示し、XとYの値が等しくなければ再度入力することを求めるプログラム）



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

```
表示される文字列   入力される変数名
        ↓          ↓
INPUT  "X="；  X
```

②LET文

左辺の変数に右辺の値、文字、または式の値を代入します。

```
省略可能  代入される変数名
  ↓        ↓
 LET      Z=X-Y
```

LETコマンドは、「LET」の記述を省略できます。

変数とは、数値や文字列を貯えておく「箱」のようなものです。変数の名前を変数名といいます。変数名の付け方には、次のような規則があります。

(1)先頭の１文字がアルファベット(「Ａ」～「Ｚ」、「ａ」～「ｚ」)またはカナ(「ア」～「ン」)を使い、２文字目からは数字も使うことができます。

(2)変数名の先頭に、予約語(BASICの命令や関数)を含めることはできません。

(例) PRINTER, LETTER

(3)変数名の最後に＄記号を付けることにより、文字変数となります。

(4)変数名の長さは、最大249文字です。

変数は意味のある変数名を付けるとよいでしょう。

(例) TEIHEN、TAKASA、GOUKEI

③IF～THEN～ELSE文

IF以下の条件式が成立したときは、THEN以下の文を実行、または指定された行番号へ分岐します。条件式が成立しなかったときは、ELSE以下の文を実行、または指定された行番号へ分岐します。

```
   条件式   成立しなかったときに分岐する行番号
     ↓                    ↓
IF  Z＝0   THEN  50  ELSE  10
                 ↑
          成立したときに分岐する行番号
```

条件式には次の記号が使用できます。

記号	例	意　味
＝	Ａ＝Ｂ	ＡとＢが等しい
＜	Ａ＜Ｂ	ＡがＢより小さい
＞	Ａ＞Ｂ	ＡがＢより大きい
＜＝ ＝＜	Ａ＜＝Ｂ Ａ＝＜Ｂ	ＡがＢより小さいか等しい
＞＝ ＝＞	Ａ＞＝Ｂ Ａ＝＞Ｂ	ＡがＢより大きいか等しい
＜＞ ＞＜	Ａ＜＞Ｂ Ａ＞＜Ｂ	ＡとＢが等しくない

④BEEP文

ブザーを鳴らします。BEEPまたはBEEP０で低い音を鳴らし、BEEP1で高い音を鳴らすことができます。

⑤PRINT文

文字列、数式、変数などの出力要素を表示します。

```
      表示する文字列   表示する変数
              ↓          ↓
PRINT   "EQUAL＝" ;  X
```

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●省略可能なIF文、LET文、END文●

- IF～THEN～ELSE文の中で、条件式が成立しなかった場合に実行する処理がなかったときは、ELSE以下を省略することができます。
- LET文では「LET」を省略し、計算式や代入文だけを記述できます。
- END文自体を省略することができます。プログラムは実行する文がなくなった時点で、自動的に終了します。

繰り返しのプログラム

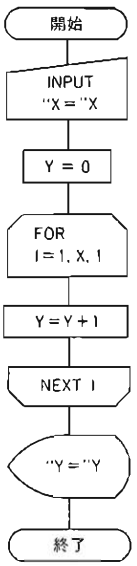
ここで学ぶこと
☆特定の文の実行を、ある回数だけ繰り返す方法

まとめ
◎FOR～NEXT文は制御変数値を変化させながら、ある回数だけ繰り返して実行させることができます。

プログラム例

```
① 10 INPUT "X=";X
② 20 Y=0
③ 30 FOR I=1 TO X STEP 1
④ 40 Y=Y+I
⑤ 50 NEXT I
⑥ 60 PRINT "Y=";Y
⑦ 70 END
```

（1から入力したXの値までの和を求めるプログラム）



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

②代入文(LET文)

左辺の変数に右辺の値、文字、または式の値を代入します。「LET」は省略してあります。行番号20では、合計値の入る変数Yを0にしています。

③FOR～NEXT文

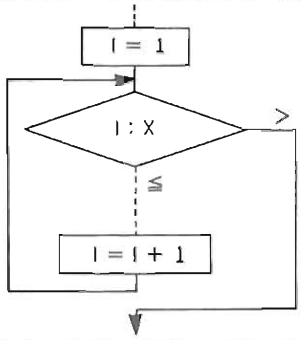
FOR文からNEXT文までの間に書かれた文を、制御変数を初期値から終値まで増分値ずつ変化させながら繰り返します。

```
制御変数名  初期値  終値        増分値
    ↓       ↓      ↓           ↓
FOR  I=    1  TO  X  STEP  1
    繰り返し実行される文
NEXT  I ←この時点でIの値が増分される
```

例では行番号30～50の文に、行番号40の文が挿入されています。この行番号30～50までをFOR～NEXTループと呼び、行番号40の文が繰り返し実行する文になります。

このように、FORとNEXTは常にペアで使用されます。また、初期値、終値、増分値は変数でも数値でも指定することができます。それでは、図を見ながらもう一度FOR～NEXTの動きを確認してみることにしましょう。

説明図　この図は、FOR～NEXTの動きを説明するための図です。プログラムとは一致していません。



……制御変数に初期値をセットします。

……制御変数と終値を比較し、制御変数の値が終値を超えたときは、NEXT文の次の文にジャンプします

……制御変数に増分値を加え、FOR文の終値との比較へ戻ります。

図のように繰り返し実行される文は、制御変数と終値とを比較した後に実行されるため、初期値が終値を超えた場合は、一度も実行されません。

また、FOR～NEXTループはメモリーが許す限り、何重にも指定することもできます。その場合、他のループに重なってはいけません。

```
┌──FOR  J=1  TO  10          ┌─FOR  J=1  TO  10
│ ┌─FOR  I=1  TO  10         │ ┌─FOR  I=1  TO  10
│ │                          │ │
│ └─NEXT  I                  │ └─NEXT  J
└──NEXT  J                   └──NEXT  I
```

正しい例(内側より実行される)　　誤った例

④代入文(LET文)

左辺の変数に右辺の値、文字、または式の値を代入します。「LET」は省略してあります。行番号40では、合計値の入る変数YにIを加算しています。

⑤PRINT文

文字列、数式、変数などの出力要素を表示します。

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●省略可能な増分値●

FOR～NEXT文の中のSTEPの記述は、省略することができます。STEP以下を省略した場合は、増分値は1が指定されたものと見なされます。

無条件ジャンプのプログラム

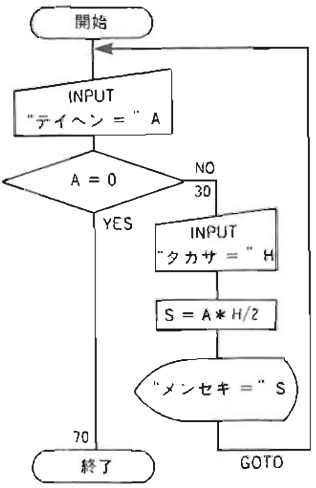
ここで学ぶこと
☆プログラムの流れを変え、特定の行番号へジャンプする方法

まとめ
◎乗算(掛け算)は＊で表わし、除算(割り算)は／で表わします。
◎GOTO文は、指定された行番号へ無条件にジャンプします。

プログラム例

```
①  10  INPUT  〃テイヘン=〃;A
②  20  IF  A=0  THEN  70
①  30  INPUT  〃タカサ  =〃;H
③  40  S=A*H/2
④  50  PRINT  〃メンセキ=〃;S
⑤  60  GOTO  10
⑥  70  END
```

(三角形の底辺と高さを入力してその面積を求め、それを表示し、底辺が0の場合は終了するプログラム)



プログラムフローチャート

プログラムの文法と要点

①INPUT文
キーボードからデータを指定した変数に入力します。

②IF文
IF以下の条件式が成立すると、THEN以下の行(行番号70)へ分岐します。
成立しないと、次の行(行番号30)へ進みます。

③代入文(LET文)
左辺の変数に右辺の値、文字、または式の値を代入します。「LET」は省略してあります。
算術式には次のような記号があり、次の優先順で計算されます。

優先順位	種類	記号
1	カッコ	( )
2	乗算　除算	＊　／
3	加算　減算	＋　－

注) 優先順位が同じ場合は、左から順に計算されます。

④PRINT文

文字列、数式、変数などの出力要素を表示します。

⑤GOTO文

指定された行番号へ無条件でジャンプします。

```
        ジャンプする行番号
        ↓
GOTO  10
```

この例では、行番号10にジャンプします。

これまでに学んだプログラムでは、IF～THEN～ELSE文やFOR～NEXT文のような文を除き、行番号順に実行され、最後の文を実行するとプログラムの実行が終了しました。このようなプログラムでは、同じ処理を何回も繰り返して実行したいとき、もう一度初めからRUNコマンドを操作しなければなりません。このような面倒な操作をしなくても、GOTO文の指定によって特定の行にジャンプして繰り返し処理することができます。

⑥END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●行番号の代わりに用いるラベル●

●GOTO文などのジャンプ先として通常は行番号を使用してますが、以下のように行番号の次にラベルを記述することにより、条件分岐命令の中で行番号の代わりに使用することができます。

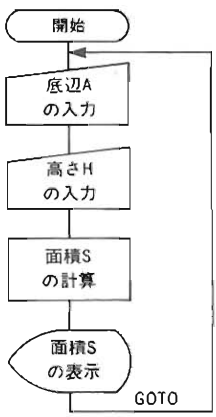
```
10 GOTO *HYOUJI
      ⋮
200 *HYOUJI:PRINT "PROGRAM RUNNING"
```

●ラベルは、先頭に「＊」(アスタリスク)を付けた英数字で始まるラベル名で構成されます。
変数名同様に先頭から予約語のつづりを使用することはできません。

●ラベル名は、１行に収まる長さまで使用できます。

●ラベルは、必ず行の先頭から記述します。続けて文を書く場合には、「：」(コロン)を付けます。

コラム●無限ループ●



例題のプログラムではAの入力値が０のときに処理が終了するように設定されていますが、行番号20のIF文がないと左のフローチャートのような流れになります。

このフローチャートには終了がありません。無限に数値の入力と計算、表示を行なうことができる代わりにいつまで経ってもプログラムは終了しません。

このように終わりがなく、無限に繰り返される繰り返しを、「無限ループ」といいます。実行を止めるためにはBREAKキーを押して、強制的に無限ループから抜け出さなければなりません。

無限ループの存在するプログラムは、あまり好ましいプログラムとはいえません。どうしても無限ループとなってしまう場合は、無限ループのどこかで脱出する方法を考えておく必要があります。

コラム●FOR～NEXTループ内からの脱出●

FOR～NEXTループ内から、GOTO文でループ外へジャンプすることができます。この場合、NEXT文が現われる以前の行にGOTO文などで戻すことにより、再びFOR～NEXTを続行することができます。

またこれとは逆に、FOR～NEXTループ外からループ内に入ることはできません。

サブルーチンの使い方

ここで学ぶこと

☆同じ処理を一つにまとめ、サブプログラム化する方法

まとめ

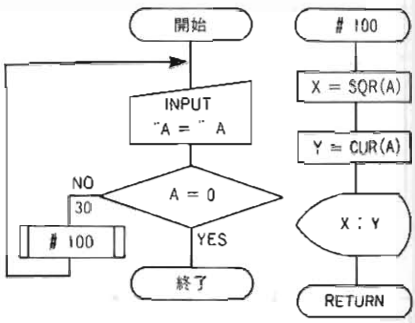
◎GOSUB文は、メインプログラム（メインルーチン）よりサブプログラム（サブルーチン）へ実行を移します。

◎RETURN文は、サブプログラムへ実行を移したメインプログラムのGOSUB文の次の文へ実行を戻します。

プログラム例

```
①   10 INPUT "A=";A
②   20 IF A=0 THEN 50
③   30 GOSUB 100
⑦   40 GOTO 10
⑧   50 END
④  100 X=SQR(A)
④  110 Y=CUR(A)
⑤  120 PRINT X;Y
⑥  130 RETURN
```

（入力したAの値の平方根Xと立方根Yを求め、それを表示し、Aの値が0の場合は終了するプログラム）



プログラムフローチャート

プログラムの文法と要点

①INPUT文

キーボードからデータを指定した変数に入力します。

②IF文

IF以下の条件式が成立すると、THEN以下の行（行番号50）へジャンプします。成立しないと、次の行（行番号30）へ進みます。

③GOSUB文

指定された行番号から始まるサブルーチンへジャンプします。サブルーチンからはRETURN文で戻ります。

```
          ジャンプ先の行番号
              ↓
GOSUB  100
```

④代入文(LET文)

左辺の変数に右辺の値、文字、または式の値を代入します。「LET」は省略してあります。

行番号100では変数Aの平方根を、行番号110では変数Aの立方根をそれぞれ計算し、変数X、Yへ代入しています。本機には内蔵関数とそのための専用キーが用意されています。

内蔵関数には数値関数の他に、文字列の変換などの文字関数などがあります。詳しくは、『ガイドブック』の巻末の「内蔵関数一覧表」を参照してください。

⑤PRINT文

文字列、数式、変数などの出力要素を画面に表示します。

⑥RETURN文

指定された行、またはサブルーチンを呼び出した文の次の文に戻ります。

⑦GOTO文

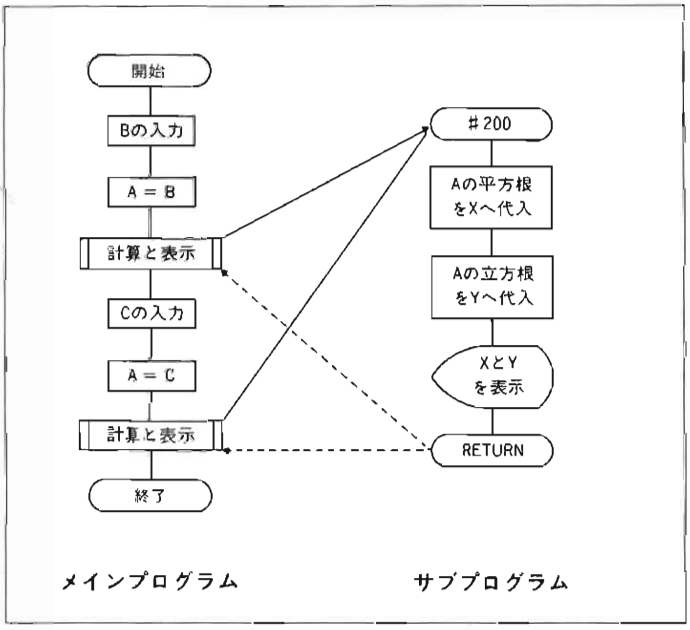
指定された行番号へ無条件でジャンプします。

⑧END

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●サブルーチンの呼び出し●

一つのサブルーチンを、メインプログラムのいろいろな位置から何回も呼び出すことができます。



図のように呼び出す側のプログラムをメインプログラム(メインルーチン)と呼び、呼び出される側のプログラムをサブプログラム(サブルーチン)と呼びます。

また、サブルーチンのフローチャートは、メインルーチンのフローチャートとは別に書きます。さらに、サブルーチンの最初の記号には(サブルーチン)(＃200)(サブルーチン名)のように、サブルーチンの開始であることがわかるような名称を書き、最後の記号には(RETURN)というように、メインプログラムへ戻ることがわかるような名称を書きます。

READ文とDATA文

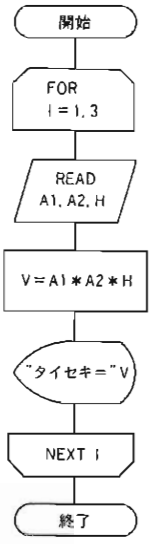
ここで学ぶこと
☆READ文によるデータ入力と、DATA文によるデータの定義

まとめ
◎READ文は、DATA文で指定されたデータを順々に読み込みます。
◎DATA文は、READ文で読み込むデータを設定します。

プログラム例

```
①  10 FOR I=1 TO 3
②  20 READ A1, A2, H
④  30 V=A1*A2*H
⑤  40 PRINT "タイセキ=";V
①  50 NEXT I
⑥  60 END
③  70 DATA 3, 5, 6
③  80 DATA 5, 8, 4
③  90 DATA 3, 4, 10
```

(行番号70から90までのデータをREAD文で読み込み、変数A1、A2、Hに代入し、この変数によって直方体の体積を求めるプログラム)



プログラムフローチャート

プログラムの文法と要点

①FOR~NEXT文

制御変数を初期値から終値まで増分値で変化させながら、FOR文からNEXT文までの間に書かれた文を繰り返します。

②READ文

DATA文で定義されたデータを読み込み、指定した変数に順々に代入します。

```
          代入される変数名
            ↓   ↓   ↓
READ  A1, A2, H
```

③DATA文

READ文で読み込む数値または文字列を設定します。

```
          読み込まれるデータ
            ↓   ↓   ↓
DATA  3, 5, 6
```

これまで学んだプログラムでは、数値を扱う場合に、(1)LET文のように変数に代入する、(2)INPUT文を使用してキーボードから数値を入力する、という2つの方法がありました。

ここではもう一つの方法として、READ文とDATA文を学びます。READ文を使用すれば、データをその度入力する必要がありません。プログラム中にすでに書き込まれていますので、後からデータをチェックするときにも便利です。

```
READ  A1, A2, H
      ↑   ↑   ↑
DATA  3, 5, 6
DATA  5, 8, 4
DATA  3, 4, 10
```

READ文とDATA文は左図のように常に対になって使用され、READ文で読み込まれるデータ数または読み込む回数とDATA文中のデータ数とが対応していなければなりません。

もし、READ文で読み込まれるデータ数がDATA文中に存在するデータ数よりも多い場合は、エラーになります。

```
READ  A, B, C, D, E   ┌ A、B、Cにデータは読み込まれるが、D、Eに読み込まれる
      ↑  ↑  ↑        └ データがない。
DATA  10, 15, 20
```

逆に、READ文で読み込まれるデータ数がDATA文中に存在するデータ数よりも少ない場合は、エラーにはなりません。

```
READ  A, B, C, D, E
      ↑  ↑  ↑  ↑  ↑        ┌── 6、4のデータは余る。
DATA  5, 7, 3, 2, 1, 6, 4
```

④代入文(LET文)

左辺の変数に右辺の値、文字、または式の値を代入します。「LET」は省略してあります。

⑤PRINT文

文字列、数式、変数などの出力要素を画面に表示します。

⑥END

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●読み込むデータの行番号指定●

READ文で読み込もうとするDATA文が複数組ある場合は、RESTORE文により読み込むデータの位置を指定することができます。

この場合、最初のREAD文が実行される前に「RESTORE 行番号」を指定すれば、指定された行番号のデータが読み込まれます。

配列変数の宣言

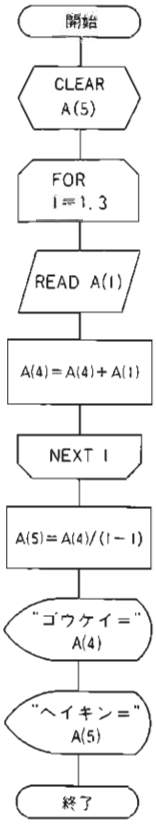
ここで学ぶこと
☆大量のデータを配列変数を使って処理をする

まとめ
◎CLEAR文は、すべての変数をクリアーします。
◎DIM文は、配列変数を宣言します。

プログラム例

```
 10 CLEAR
 20 DIM A(5)
 30 RESTORE
 40 A(4)=0
 50 FOR I=1 TO 3
 60 READ A(I)
 70 A(4)=A(4)+A(I)
 80 NEXT I
 90 A(5)=A(4)/(I-1)
100 PRINT "ゴウケイ=";A(4)
110 PRINT "ヘイキン=";A(5)
120 END
130 DATA 3,2,10
```

(行番号100で指定した数値を配列に代入し、合計と平均を求めるプログラム)



プログラムフローチャート

プログラムの文法と要点

①CLEAR文

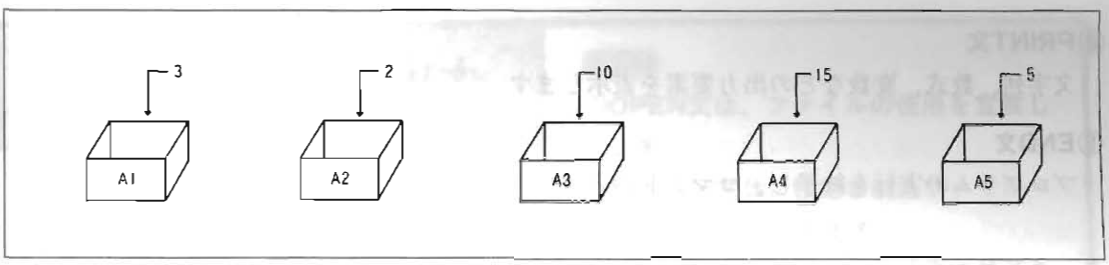
すべての変数をクリアーします。

②DIM文

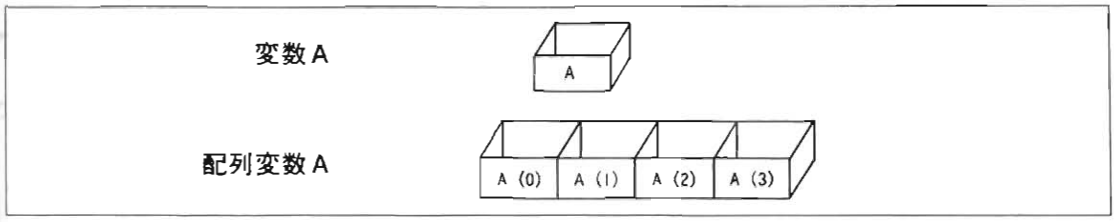
配列変数の宣言には「DIM」文を使います。

```
配列変数名　添え字の最大値
      ↓  ↓
   DIM  A(5)
```

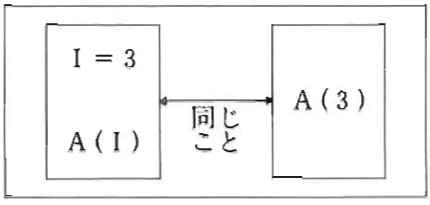
これまでに学んだ変数では、一つの変数に対して一つの値しか代入できませんでした。ですから、数多くの値を扱うときには、いくつもの変数を用意しなければなりませんでした。



配列を使うと、一つの配列変数に対していくつもの値を代入することができます。
変数を箱に例えると、配列変数はいくつもの箱をひとまとめにしたものに例えられます。



一つ一つの箱を区別するために、配列変数名に続く( )の中の数字で何番目の箱であるかを指定しています。この数字を「添え字」といいます。( )の中の添え字には、数値でも変数名でも指定することができます(右の図を参照)。



例題のプログラムでは、配列は3を宣言しています。配列の中にFOR～NEXT文で代入するよりも、READ文を1度実行して順に代入した方が良いと思う人がいるかも知れません。データ数が少ないときは、それでも大きな違いはありませんが、扱うデータが100個というような場合には、1度のREAD文によるプログラムではものすごい長さになってしまいます。

③FOR～NEXT文

FOR文からNEXT文までの間に書かれた文を、制御変数を初期値から終値まで増分値で変化させながら繰り返します。

④RESTORE文

READ文で読み込むDATA文の開始行を指定します。ここでは、開始行が指定されていないので、プログラムの中で最初にあるDATA文が指定されます。

⑤READ文

DATA文で定義されたデータを読み込み、指定した変数に順々に代入します。

⑥DATA文

READ文で読み込む数値または文字列を設定します。

⑦代入文(LET文)

左辺の変数に右辺の値、文字、または式の値を代入します。

行番号60の代入文では平均を求めています。なぜ変数Iより1を減算してから割るのかというと、FOR～NEXT文では制御変数が終値を超えたときに実行を終えるからです。

⑧PRINT文

文字列、数式、変数などの出力要素を表示します。

⑨END文

プログラムの実行を終了し、コマンドの入力待ちとなります。

コラム●配列の0番目？●

本機では配列を0番目から使用することができます。したがって、A(5)という配列を宣言すると、0～5番目の計6個のデータを記憶することができます。

周辺機器との入出力

ここで学ぶこと

☆フロッピーディスクへの書き込み方

☆フロッピーディスクからの読み込み方

まとめ

◎OPEN文は、ファイルの使用を宣言します。

◎PRINT＃文は、指定されたファイルへデータを書き込みます。

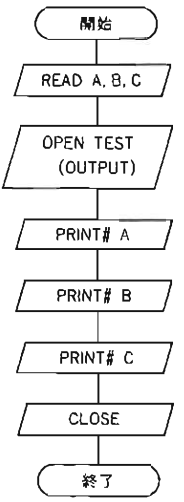
◎CLOSE文は、ファイルの使用の終了を宣言します。

◎INPUT＃文は、指定されたファイルからデータを読み込みます。

プログラム例 1

```
①  10 READ A, B, C
②  20 DATA 17, 14, 21
③  30 OPEN "0:TEST" FOR OUTPUT AS
       #1
④  40 PRINT #1, A
④  50 PRINT #1, B
④  60 PRINT #1, C
⑤  70 CLOSE
⑥  80 END
```

（行番号20で指定した数値を変数に代入し、フロッピーディスクにオープンした"TEST"というファイルに変数の値を書き込むプログラム）

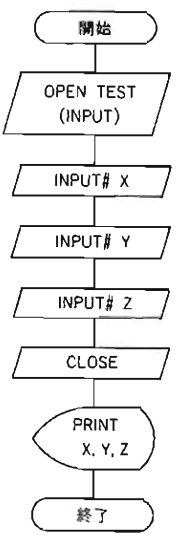


プログラムフローチャート

プログラム例 2

```
③  10 OPEN "0:TEST" FOR INPUT AS
       #1
⑦  20 INPUT #1, X
⑦  30 INPUT #1, Y
⑦  40 INPUT #1, Z
⑤  50 CLOSE
⑧  60 PRINT X;Y;Z
⑥  70 END
```

（フロッピーディスクから"TEST"というファイルをオープンして、データを読み込み表示するプログラム）



プログラムフローチャート

プログラムの文法と要点

①READ文

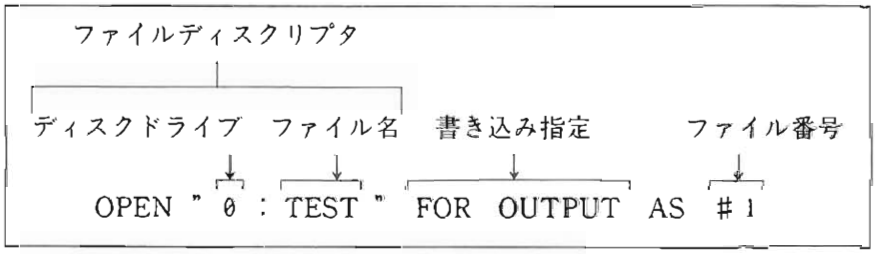
DATA文で定義されたデータを読み込み、指定した変数に順々にデータを代入します。

②DATA文

READ文で読み込む数値定数または文字定数を設定します。

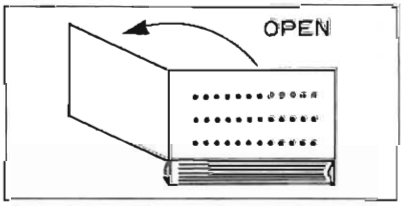
③OPEN文

ファイルの使用を宣言します。プログラム例1、プログラム例2とも同じような宣言をしています。



フロッピーディスクに記録されたデータ、またはデータの集まりをファイルといいます。このファイルを使用するための命令としてOPENがあります。OPEN文の実行は、これからデータを読み書きするデータファイルを「開く」という意味です。

ファイルを開く(OPEN)とは、これからデータを記録するためノートやデータを読むための本などの表紙を「開く作業」と同じようなものと考えるとよいでしょう。



ファイルのOPENのイメージ

プログラム例1の行番号30のOPEN文では、ディスクドライブの「TEST」という名前を付けたファイルに書き込みを行ないます。「FOR OUTPUT」という部分によって、OPENする目的がファイルへの書き込みに指定されます。

OPENする目的は、次のようにして指定します。

	目 的	備 考
OPEN～FOR OUTPUT～	ファイルへの書き込み	プログラム例1 行番号30
INPUT	ファイルから読み込み	プログラム例2 行番号10
APPEND	ファイルへの追加書き込み	

OPEN文では、開いたファイルに番号を付けています。プログラム例1、プログラム例2ではともにOPEN文により、ファイル番号として「AS ＃1」を指定しています。その後の出力するための命令(例1の行番号40～60)と、入力するための命令(例2の行番号20～40)により、同じくファイル番号として「＃1」を指定します。

④PRINT＃文

指定されたファイルに、出力要素を順番に出力します。

```
ファイル番号　出力要素(例では変数A)
          ↓  ↓
PRINT　＃1，A
```

PRINT＃文の実行によって、フロッピーディスクにデータが書き込まれます。書き込まれたデータを読み込むときは、INPUT＃文を使用します。プログラム例1で書き込まれたTESTというファイルを、プログラム例2で読み込んでいます。

フロッピーディスクに書かれたデータを読み込むときは、次のように指定してください。

注　意	書き込み	読み込み
数値型データは数値変数に読み込む	PRINT　＃1，A	INPUT　＃1，P
文字型データは文字変数に読み込む	PRINT　＃1，B$	INPUT　＃1，Q$
数値型データを文字変数に読み込んではいけない	PRINT　＃1，C	INPUT　＃1，R$
文字型データを数値変数に読み込んではいけない	PRINT　＃1，D$	INPUT　＃1，S
変数は繰り返し書き込み、繰り返し読み込める	PRINT　＃1，E，F，G	INPUT　＃1，T，U，V

コラム●BASICで扱うファイル媒体とファイルディスクリプタ●

本機で扱うことのできるファイルの媒体には、フロッピーディスク、通信回線があります。これらを区別するために次のように指定します。

●フロッピーの場合　OPEN　"0：ファイル名" FOR～

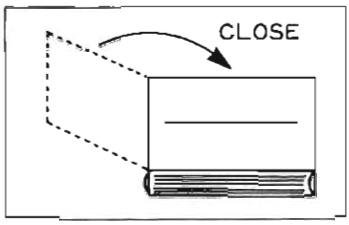
"　"で囲まれた部分をファイルディスクリプタといいます。ファイルディスクリプタには、次の2種類があります。

対象となるファイル	ファイルディスクリプタ
フロッピーの指定	"0：ファイル名"
通信回線の指定	"COM 0：通信パラメーター"

⑤CLOSE文

ファイルを閉じ、入出力バッファ使用の終了を宣言します。CLOSE文では、ファイル名、出力モード、ファイル番号などは指定しません。

先ほどは、書き込みそして読み込むための準備として、OPEN文を使ってファイルを開く作業をしました。書き込む、読み込むなどの作業が終了すれば、ファイルを閉じるという作業を行ないます。CLOSE文の実行は、データを読み書きしたデータファイルを「閉じる」という意味です。



ファイルのCLOSEのイメージ

ファイルを閉じる(CLOSE)とは、データを記録し終わったノートやデータを読み終わった本などの表紙を「閉じる作業」と同じようなものと考えるとよいでしょう。

⑥END

プログラムの実行を終了し、コマンドの入力待ちとなります。

⑦INPUT＃文

指定されたファイルよりデータを読み込み、変数にデータを代入します。

```
  ファイル番号 代入する変数名
          ↓  ↓
INPUT   #1, X
```

⑧PRINT文

文字列、数式、変数などの出力要素を表示します。

コラム●データの保存●

本機では電源を切ってもメモリー保護用電池によってデータが保存されます。しかし、次の場合には、SAVE文を使用してフロッピーディスクにデータを保存してください。

(1)本体の記憶容量が不足したとき

(2)本体に記憶されているデータと同じものを、バックアップとして保存しておきたいとき

コラム●テキストデータを使う●

ファイルエリアに書き込んだデータをBASICプログラム中で利用するためのものとして、READ＃文、RESTORE＃文、WRITE＃文が用意されています。

READ文は、プログラム中のDATA文の内容を読み込みます。一方、READ＃文は、ファイルエリア内のデータを指定する変数に読み込みます。

RESTORE文は、READ文で読み込むDATA文の位置を指定します。一方、RESTORE＃文は、データをサーチ(検索)したり、READ＃で読み込むデータの順番を変えたりします。

また、ファイルエリアのデータの書き換えや削除するための命令として、WRITE＃が用意されています。

文法のまとめ

ここで学ぶこと

☆文、演算子、変数の復習

まとめ

◎文は、行番号とコマンドとオペランドで構成されます。

◎演算子には、比較演算子と文字演算子があります。

◎変数には、数値変数と文字変数と配列変数があります。

文の構成

各文(行)は、次のように構成されます。

行番号	コマンド(命令)	オペランド

比較演算子(プログラム中にのみ有効)

二つの数値または二つの文字列の比較を行ない、結果を真(−1)または偽(0)で表わします(演算子については「条件分岐するプログラム(IF〜THEN〜ELSE文)」を参照してください)。

	文	結　果	備　考
数値の比較	PRINT　123>45	−1(真)	数値の大小を比較します。
	PRINT　123<45	0(偽)	
文字列の比較	PRINT　"DEF"<"ABC"	0(偽)	先頭から順に文字コードの大きさを比較します。
	PRINT　"ABC"<"ABD"	−1(真)	
	PRINT　"ABC">"ABCD"	0(偽)	短い文字列(「ABC」の3文字)で比較し、等しいときは短い方が小と判断します。

文字演算子

文字列の演算は＋(プラス)だけが有効です。結果が、256文字以上の長さになってはいけません。

"A"＋"B"→"AB"
　　↑
255文字以下

変数の種類

- 変数
  - 数値変数
    - 固定数値変数(A～Z)
    - 登録数値変数
  - 文字変数
    - 登録文字変数
  - 配列変数
    - 配列数値変数
    - 配列文字変数

※配列変数は登録変数です。

変数名と配列名の付け方

変数名と配列名の付け方には、次のような規則があります。

①英大文字、英小文字、カナで始まる文字列である。
②英大文字、英小文字、カナ、数字で構成される。
③先頭から予約語を含んではならない。
④長さは、255文字以内である。

配列

①配列は、DIM文で宣言します。
②配列の添え字は0以上の整数で、小数部は切り捨てられます。
③配列の次元は、表記上許される範囲です。
④添え字の最大量は、記憶容量の許す範囲です。

変数、配列の取り扱い

①変数、配列は、すべてのプログラムに対して共通に用いられます。
②固定変数は、AからZまでの英大文字1字で構成されます。
③登録変数は、初めて使用したときにその場所が確保されます。
④配列宣言をしないと、配列変数は使えません。

コラム●変数の使用バイト数●

固定数値変数は、あらかじめその場所が確保されています。

登録数値変数と登録文字変数は、初めて使用したときにその場所が確保されます。また、配列変数は、DIM文で定義したときにその場所が確保されます。

固定数値変数以外の変数が確保されるときに使用されるバイト数は、次のとおりです。

●ワークエリアの使用バイト数

数値、文字、配列の別に関係なく

変数名長が偶数のとき…(変数名長＋6)バイト

変数名長が奇数のとき…(変数名長＋7)バイト

が確保されます。

●変数データ領域の使用バイト数

数値変数…8バイト

文字変数…文字長＋1(奇数のときは、さらに＋1)バイト

配列数値変数…|配列の大きさ×8＋(次元＋2)×2|バイト

配列文字変数…|配列の大きさ×2＋(次元＋2)×2|バイト

マニュアルコマンド一覧表

注）・マニュアルコマンドは、プログラム内で実行できません。
・{　}内はいずれか一つを示します。{　}そのものは書き込みません。
・〔　〕内は省略可能です。ただし〔　〕そのものは書き込みません。
・※の付いたコマンドは、CALモードでも使えます。

コマンド名	書　式	機能および使用例
LIST	LIST {〔開始行番号〕〔－〔終了行番号〕} / . / ALL / ＊ラベル}	プログラムの内容の全部または一部を画面に表示します。 LISTをLLISTとすると、画面への出力からプリンタへの出力になります。 ⇨①LIST　（先頭から表示） ②LIST30　（行番号30を表示） ③LIST20－80（行番号20～80を表示） ④LIST20－　（行番号20以降を表示） ⑤LIST－90　（先頭行～行番号90までを表示） ⑥LIST.　（最後に扱った行を表示） ⑦LIST ALL（すべてのプログラムエリアのプログラムを表示） ⑧LIST＊INIT（ラベル＊INIT以降を表示）
※LIST＃	LIST＃	ファイルエリア「F0」に書き込まれているテキストデータをすべて表示します。LISTをLLISTとすると、画面への出力からプリンタへの出力になります。 ⇨LIST＃
EDIT	EDIT {〔行番号〕 / . / ＊ラベル}	現在指定されているプログラムエリアのプログラムを表示し、編集モードに入ります。 ⇨①EDIT　（最初の行より編集を行ないます） ②EDIT 30　（行番号30の編集を行ないます） ③EDIT .　（最後に扱った行の編集を行ないます） ④EDIT ＊INIT（ラベル＊INITの行の編集を行ないます）
RENUM	RENUM 〔新行番号〕〔,〔旧行番号〕〔,増分〕〕	行番号を一定間隔で付け直します。 ⇨RENUM 100,10,10（行番号10を行番号100とし、以降10間隔で付け直します）
NEW	NEW 〔ALL〕	現在指定されているプログラムエリアのプログラム、またはすべてのプログラムエリアのプログラムを消去します。 117ページ参照

コマンド名	書　式	機能および使用例
DELETE	DELETE {〔開始行番号〕 〔－〔終了行番号〕〕 / ＊ラベル}	プログラムの一部を行単位で削除します（パラメーターは両方とも省略することはできません）。 ⇨①DELETE50　（行番号50を削除） ②DELETE20－80（行番号20～80を削除） ③DELETE20－　（行番号20以降を削除） ④DELETE－90　（先頭行～行番号90までを削除） ⑤DELETE＊INIT（ラベル＊INIT以降を削除）
RUN	RUN {〔開始行番号〕 / ＊ラベル}	プログラムを先頭行または指定した行から実行します。 119ページ、123ページ参照
※CONT	CONT	STOP文またはSTOPキーで停止されたプログラムの実行を再開します。 ⇨CONT
※SYSTEM SYSTEMP SYSTEMF	SYSTEM SYSTEMP SYSTEMF	プリンタのON/OFF設定、トレースモードのON/OFF設定、CLEAR文の設定、テキストエリア（FREE）、変数領域（V）のフリーエリア容量を表示します。 SYSTEMPとSYSTEMFは、それぞれプログラムエリアとファイルエリアの使用状況を表示します。 ⇨SYSTEM
※PASS	PASS"パスワード"	全プログラムエリアおよび全ファイルに対するパスワードを、設定または解除します。 ⇨PASS"CASIO"
SAVE	SAVE"ファイルディスクリプタ"〔,A〕	現在指定されているプログラムエリアのプログラムを、ファイルディスクリプタで指定されたファイルへ出力します。 「,A」を付けたときは、アスキー形式で出力します。 ⇨SAVE"0:DEMO1.BAS"（フロッピーディスク内のファイル"DEMO1.BAS"へ出力）
LOAD	LOAD"ファイルディスクリプタ"	現在指定されているプログラムエリアに、ファイルディスクリプタで指定されたファイルよりプログラムを読み込みます。 ⇨LOAD"0:DEMO1.BAS"（フロッピーディスク内のファイル"DEMO1.BAS"より読み込みます）

コマンド名	書　式	機能および使用例
MERGE	MERGE "ファイルディスクリプタ"	現在指定されているプログラムエリアに、ファイルディスクリプタで指定されたファイル上のプログラムを混合します。 ⇨①MERGE"0:TEST.BAS"(フロッピーディスク内のファイル"TEST.BAS"と混合します。"TEST.BAS"はアスキー形式)
BSAVE	BSAVE "ファイルディスクリプタ" ,開始番地,長さ〔,実行開始番地〕	マシン語プログラムをファイルへ出力します。 ⇨BSAVE"0:DEMO.EXE",0,&H100(フロッピーディスク内のファイル"DEMO.EXE"へ出力します)
BLOAD	BLOAD "ファイルディスクリプタ" 〔,〔格納開始番地〕〔,R〕〕	指定したファイルよりマシン語プログラムを読み込みます。 ⇨BLOAD"0:DEMO.EXE"(フロッピーディスク内のファイル"DEMO.EXE"より読み込みます)
MON	MON	モニターを起動します。 ⇨MON

プログラムコマンド一覧表

注）・{　}内はいずれか一つを示します。{　}そのものは書き込みません。
・〔　〕内は省略可能です。ただし〔　〕そのものは書き込みません。

コマンド名	書　式	機能および使用例
CLEAR	CLEAR〔変数領域サイズ〕〔,マシン語サイズ〕〔,ワーク領域サイズ〕	すべての変数をクリアーし、パラメーターにしたがってメモリーの領域を確保します。 ⇨CLEAR　136ページ参照
DEFSEG	DEFSEG＝セグメントアドレス	PEEK関数、POKE文、CALL文を実行するときのセグメントベースアドレスを設定します。 ⇨DEFSEG＝&H2000
MODE	MODE　数式	角度単位やプリンタ出力の指定をします。 ⇨①MODE 4（DEG：度） ②MODE 5（RAD：ラジアン） ③MODE 6（GRA：グラード） ④MODE 7（プリンタ出力ON） ⑤MODE 8（プリンタ出力OFF）
ANGLE	ANGLE　角度指定	角度単位を0、1、2の数値によって指定します。 ⇨①ANGLE 1（DEG：度） ②ANGLE 2（RAD：ラジアン） ③ANGLE 3（GRA：グラード）
SET	SET {F, E} 0～9の1文字 N	数値データの出力形式を指定します。Fは小数点以下の桁数を、Eは有効桁数を指定し、Nは指定を解除します。 ⇨SET F3
DIM	DIM　配列名(添え字の最大値〔,添え字の最大値…〕)	配列の変数を宣言します。 ⇨①DIM A(5)（数値変数Aを1次元の配列として宣言） ⇨②DIM B$(2,5)（文字変数B$を2次元の配列として宣言） 136ページ参照
ERASE	ERASE　配列名　〔,配列名〕	指定した配列を変数名単位で消去します。 ⇨ERASE A,B

コマンド名	書　式	機能および使用例
IF～ (THEN / GOTO)～ ELSE	IF 条件文 { THEN { 文〔:文〕 / 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 } / GOTO { 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 } } 〔 ELSE { 文〔:文〕 / 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 } 〕	条件式が成立したときは、THEN以下の文を実行するか、THEN、GOTO以下の分岐先行番号、ラベル、または指定されたプログラムエリアの先頭の行番号へジャンプします。 また条件式が成立しないときは、ELSE以下の文を実行するか、ELSE以下の分岐先行番号、ラベル、または指定されたプログラムエリアの先頭の行番号へジャンプします。 ⇨①IF A＞100 THEN 50 ELSE 80 ②IF B＝0 THEN X＝B ELSE Y＝B ③IF C＝1 THEN *CHK1 ELSE *CHK2 ④IF D＜＞50 THEN ＃9
FOR～TO～ STEP ⋮ NEXT	FOR　変数＝初期値　TO　終値 〔STEP増分値〕 ⋮ NEXT 〔変数名〕	FOR～NEXT間の処理を、変数が終値を超えない範囲で繰り返し行ないます。 ⇨FOR I＝1 TO 10 ⋮ NEXT I
GOTO	GOTO { 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 }	指定された分岐先行番号、ラベル、または指定されたプログラムエリアの先頭の行番号へ、無条件でジャンプします。 ⇨①GOTO 80 ②GOTO ＊JMP1 ③GOTO ＃7　129ページ参照
ON～GOTO	ON　数式　GOTO { 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 } 〔 , { 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 } 〕	ON以下の数式の値に対応して、GOTO文を実行します。分岐先は、先頭から順に式の値が1の場合、2の場合、…と割り当てられます。 ⇨①ON A GOTO 100, 200, , 300 ②ON B GOTO *JMP1, *JMP2, *JMP3 ③ON X＋Y GOTO 100, ＃6, ＃7
GOSUB	GOSUB { 分岐先行番号 / ＊ラベル / ＃プログラムエリア番号 }	指定された分岐先行番号またはラベルから始まるサブルーチンを呼び出します。または指定されたプログラムエリアの先頭から始まるサブルーチンを呼び出します。 ⇨①GOSUB 100 ②GOSUB ＊INIT ③GOSUB ＃5　132ページ参照

コマンド名	書　式	機能および使用例
ON～GOSUB	ON　数式　GOSUB {分岐先行番号 / ＊ラベル / ＃プログラムエリア番号} [, {分岐先行番号 / ＊ラベル / ＃プログラムエリア番号}]	ON以下の数式の値に対応して指定された分岐先行番号またはラベルから始まるサブルーチンを呼び出します。または、指定されたプログラムエリアの先頭から始まるサブルーチンを呼び出します。分岐先は、先頭から順に式の値が1の場合、2の場合、…と割り当てられます。 ⇨①ON A GOSUB 1000, 1200, , 1500 ②ON B GOSUB ＊PROG1, ＊PROG2, ＊PROG3 ③ON X+Y GOSUB100, ＃5, ＃8
RETURN	RETURN {戻り先行番号 / ＊ラベル / ＃プログラムエリア番号}	サブルーチンから指定された行番号、ラベル、または指定されたプログラムエリアの先頭に復帰します。戻り先行番号の指定がないときは、サブルーチンを呼んだ文の次の文へ復帰します。 ⇨①RETURN ②RETURN 20 ③RETURN ＊PRT ④RETURN ＃1　132ページ参照
INPUT	INPUT 〔"メッセージ文"{;}〕 変数名 〔,〔"メッセージ文"{;}〕 変数名…〕	キーボードからデータを入力し、指定した変数に代入します。また、メッセージ文を表示し、変数を入力することもできます。 ⇨①INPUT A, B, C ②INPUT "X＝";X　124ページ参照
CLS	CLS	ディスプレイ画面を消去(クリアー)します。 ⇨CLS
LOCATE	LOCATE　X座標, Y座標	仮想スクリーン上のカーソルの位置を指定します。 ⇨LOCATE 10, 0

コマンド名	書　式	機能および使用例
PRINT	PRINT {REV / NORM / TAB(タブ指定) / 数式 / 文字列 / 変数} [; {REV / NORM / TAB(タブ指定) / 数式 / 文字列 / 変数}]	変数の内容、式の値、文字列などの出力要素を表示します。PRINTをLPRINTとすると、画面への出力からプリンタへの出力になります。 ⇨①PRINT　(改行) ②PRINT A, B, C　(変数A, B, Cの内容表示) ③PRINT "X=";X　(文字列と変数Xの内容表示) ④PRINT REV;"カシオ"　(文字列を反転表示) ⑤PRINT TAB(5);"カシオ"(文字列を左端より5つずらして表示) 86ページ参照
PRINT USING	PRINT USING "書式指定";出力要素	書式指定にしたがって出力要素を表示します。なお、USING以下はLPRINT、PRINT#にも指定することができます。 ⇨①PRINT USING "&　　&";A$ (&　　&の長さ分のみ表示されます) ②PRINT USING "###.##";X (###.##に合わせて数値を表示し、整数部分の無効数字は空白に置き換えられます)
TAB	TAB(数式)	指定された位置までスペースを表示します。また、プリンタにスペースを出力します。 ⇨PRINT TAB(10);"ABC"
DRAW	DRAW [−][(x座標, y座標) [−({(x座標, y座標)})]	画面に点や線を描きます。 ⇨①DRAW(0,0)−(50,50) ②DRAW−(100,50)
DRAWC	DRAWC [−][(x座標, y座標) [−({(x座標, y座標)})]	画面に描かれた点や線を消します。 ⇨①DRAWC(0,0)−(50,50) ②DRAWC−(100,50)
REV	PRINT REV ;[出力要素]	文字を反転して表示させます。 ⇨PRINT REV;"ハンテン"
NORM	PRINT NORM ;[出力要素]	通常の表示に戻します。 ⇨PRINT NORM;"ヒョウジ"
OPEN	OPEN "ファイルディスクリプタ" [FOR {INPUT / OUTPUT / APPEND} AS[#]ファイル番号]	ファイルの使用を宣言します。 ⇨OPEN "DATA1.DAT" FOR INPUT AS #1

コマンド名	書　式	機能および使用例
CLOSE	CLOSE	現在使用しているファイルを閉じて、I/Oバッファ使用の終了を宣言します。 ⇨CLOSE
PRINT＃	PRINT＃ファイル番号,〔{REV / NORM / TAB(タブ指定) / 数式 / 文字式}〕 〔{, / ;}{REV / NORM / TAB(タブ指定)… / 数式 / 文字式}〕	シーケンシャルファイルへ、変数の内容、式の値、文字列などの出力要素を出力します。 ⇨PRINT＃1, A$ 139ページ参照
INPUT＃	INPUT＃ファイル番号, 変数名 〔, 変数名…〕	指定されたファイル番号のシーケンシャルファイルからデータを読み込みます。 ⇨INPUT＃1, A　139ページ参照
LINE INPUT＃	LINE INPUT＃ファイル番号, 文字変数名	シーケンシャルファイルから1行文のデータを読み込みます。 ⇨LINE INPUT＃1, A$
DATA	DATA　データ1〔, データ2…〕	READ文で読むデータを設定します。 ⇨DATA 10, 20, 15　134ページ参照
READ	READ　変数1〔, 変数2…〕	DATA文で設定されたデータを読み込み、変数に順次格納します。 ⇨READ A, B, C　134ページ参照
RESTORE	RESTORE〔行番号〕	READ文で読むデータ文の開始行を指定します。 ⇨RESTORE　（データ文の先頭行を指定） RESTORE 100（100行のデータ文を指定）
READ＃	READ　＃変数名〔, 変数名…〕	ファイルエリア「F0」に書き込まれているテキストデータを変数に読み込みます。 ⇨READ＃ A$, X
WRITE＃	WRITE＃〔テキストデータ〕 〔, テキストデータ…〕	ファイルエリア「F0」のデータの書き換え、削除を行ないます。 ⇨①WRITE＃　（削除） ②WRITE＃"カシオ Z−1"（書き換え）

コマンド名	書　式	機能および使用例
RESTORE＃	RESTORE＃〔("ファイルエリア名")〕 〔"検索文字列"〕 [,[0/1][,GOTO 分岐先行番号/＊ラベル/＃プログラムエリア番号]]	READ＃、WRITE＃の対象になるファイルエリアを切り換えます。また、ファイルエリア内のデータを検索し、READ＃文で読むデータの位置を変更します。 ⇨①RESTORE＃("F1") ②RESTORE＃"START"
REM	{REM/'} 注釈文	注釈(コメント)を表わします。何も実行しません。 ⇨REM サイダイチヲ モトメル プログラム
LET	LET 変数名＝{代入値/計算値}	左辺の変数名に右辺の値、または計算式の値を代入します。代入文は、LET自体を省略することができます。 ⇨①LET A＝10 ②A$＝"カシオ" ③X＝Y＊Z/2　124ページ参照
BEEP	BEEP {〔0〕/1}	ブザーを鳴らします。 ⇨①BEEP　(低音で鳴らします) ②BEEP 0 (低音で鳴らします) ③BEEP 1 (高音で鳴らします) 124ページ参照
WAIT	WAIT 時間	指定された時間だけプログラムの実行を停止します。時間は、1のとき約1/10秒です。 ⇨WAIT 100 (約10秒間プログラムを停止します)
STOP	STOP	プログラムの実行を一時停止します。 CONT文で実行を再開します。 ⇨STOP　123ページ参照
END	END	プログラムを終了します。 ⇨END
DEFCHR$	DEFCHR$(コード)＝"文字形"	表示する文字の字形を定義します。 ⇨DEFCHR$(252)＝"0F0F0F0F0F0F"
PEEK	PEEK(数式)	数式で指定されたアドレスの内容を与えます。実際のアドレスは、DEFSEG文で指定されたセグメントベースアドレスにPEEK文のアドレスを加えたアドレスになります。 ⇨A＝PEEK(&H1000)

コマンド名	書　式	機能および使用例
POKE	POKE アドレス,データ	指定されたアドレスにデータを書き込みます。実際のアドレスは、DEFSEG文で指定されたセグメントベースアドレスにPEEK文のアドレスを加えたアドレスになります。 ⇨POKE &H7000,0
TRON	TRON	BASICプログラムのトレースモードを設定します。 ⇨TRON　123ページ参照
TROFF	TROFF	BASICプログラムのトレースモードを解除します。 ⇨TROFF　123ページ参照
VARLIST	VARLIST	現時点で存在するすべての変数名、配列名を表示します。 ⇨VARLIST
ON ERROR GOTO	ON ERROR GOTO {分岐先行番号 / ＊ラベル}	エラー発生時の分岐先行番号を指定します。
RESUME	RESUME [{NEXT / 戻り行番号 / ＊ラベル}]	エラー処理ルーチンより復帰します（NEXTまたは戻り番号を省略すると、エラーの発生した文へ戻ります）。 ⇨①RESUME NEXT （エラーの文の次の文へ復帰） ②RESUME 100
FORMAT	FORMAT [{/6 / /9 / /M}]	フロッピーディスクをフォーマット(初期化)します。 ⇨①FORMAT/6 ↵ (640Kバイトのフォーマットを行ないます) ②FORMAT/9 ↵ (720Kバイトのフォーマットを行ないます) ③FORMAT/M ↵ (1Mバイトのフォーマットを行ないます)
FILES	FILES "ファイルディスクリプタ"	フロッピーディスク内のファイルディスクリプタで指定されたファイル名、属性、使用容量などを表示します（＊、？ワイルドカード使用可）。 ⇨①FILES ②FILES "0:TEST.DAT" ③FILES "0:＊.DAT"

コマンド名	書　式	機能および使用例
CHAIN	CHAIN "ファイルディスクリプタ"	現在のプログラムエリアにファイルディスクリプタで指定されたプログラムを読み込み実行します。 ⇨①CHAIN "0:TEST. BAS"
KILL	KILL "ファイルディスクリプタ"	フロッピーディスク内のファイルディスクリプタで指定されたファイルを消去します（＊、？ワイルドカード使用可）。 ⇨①KILL "0:TEST. DAT" ②KILL "0:＊. DAT"
NAME	NAME "旧ファイルディスクリプタ" AS "新ファイルディスクリプタ"	フロッピーディスク内のファイル名を変更します。 ⇨NAME"0:TEST.BAS"AS"0:ABC.BAS"
STAT	STAT Xデータ〔,Yデータ〕〔;度数〕	統計データを入力します。 ⇨STAT 1,3:10
STAT CLEAR	STAT CLEAR	統計処理機能をクリアー(初期化)します。 ⇨STAT CLEAR
CALL	CALL アドレス	マシン語サブルーチンを呼び出します。DEFSEG文で指定されたセグメントベースアドレスにCALLで指定したアドレスを加えた番地が、マシン語プログラムの実行開始番地となります。 ⇨CALL &H0
OUTPORT	OUTPORT　ポート番号,データ	指定されたポートに1バイトのデータを出力します。 ⇨OUTPORT(0,8)
OUT	OUT ポート番号,データ $\left[,\left\{\begin{matrix}1\\2\end{matrix}\right\}\right]$	I/Oデバイスに1バイトまたは1ワードのデータを出力します。⇨OUT0,15

エラーメッセージ一覧表

エラーコード	エラーメッセージ	エラー内容	対処方法
1	OM error	①メモリーオーバーまたは、システムオーバーフロー。 ②CLEAR文でメモリーの確保できない値を設定した。	①プログラムを短かくする。配列の次元を検討する。 ②CLEAR文での値を検討する。 ③RAM増設していない場合は増設する。
2	SN error	コマンドまたは文の書式に誤りがある。	①命令のつづりを再確認する。 ②プログラムの入力を確認する。
3	ST error	文字長が255文字を超えた。	文字列の長さを255文字以内にする。
4	TC error	式が複雑すぎる。	式を分けて記述する。
5	BV error	①入出力バッファがオーバーフローした。 ②1行256バイト以上。 または256文字以上入力した。	①RS-232Cのボーレートを遅くする。 ②1行255文字以内で入力する。
6	NR error	①I/Oが入出力可能な状態になっていない。 ②オープンされていないファイルをアクセスしようとした。	①I/O接続および電源の確認をする。 ②フロッピーディスクをMD-120にセットする。 ③ファイルを正しくオープンする。
7	RW error	I/Oデバイスの動作でエラーになった。	I/Oデバイスを確認する。
8	BF error	ファイル名指定に誤りがある。	ファイル名の確認をする。
9	BN error	ファイル番号指定に誤りがある。	ファイル番号の指定を確認する。
10	NF error	指定されたファイル名が見つからない。	ファイル名を再確認する。 ファイルの属性を確認する。
12	FL error	①フロッピーディスクに書き込むスペースがないのに、書き込もうとした。 ②1つのプログラムファイルが約64Kバイトを超えた。 配列の全体の大きさが64Kバイトを超えた。	①不必要なファイルをKILL文で消し、空きエリアを大きくする。 ②FORMATした新しいフロッピーディスクを利用する。 ③1つのファイルの大きさを小さくする。 配列の大きさを小さくする。
13	OV error	計算結果や入力された数値が、許される範囲を超えた。	計算の対象になる数値を検討する。
14	MA error	①0による除算など、数学的エラー。 ②関数の引数が演算範囲を超えている。	計算式、数値を検討する。
15	DD error	同一配列を二重定義しようとした。	同じ配列を使わない。 一度ERASE命令で配列をクリアーしてから再度定義する。
16	BS error	添字またはパラメーターが規定の範囲を超えた。	①添字パラメーターを検討する。 ②配列を大きくする。

エラーコード	エラーメッセージ	エラー内容	対処方法
17	FC error	①関数やステートメントの呼び方に誤りがある。 ②ダイレクトモードで使用できない文を実行しようとした。またはその逆。 ③CALモードで実行できない文を実行しようとした。 ④定義されていない配列を使用しようとした。	①引数の値とステートメントを検討する。 ②プログラムモード、ダイレクトモードしか使えないものがあるので文法を確認する。 ③文の確認をする。 ④DIM文で配列を定義してから使う。
18	UL error	①GOTO、GOSUB等で指定された行番号がない。 ②BASICのEDITモードで行番号を入力せずに文を入力した。	①行番号を確認する。 ②行番号を必ず入れる。
19	TM error	①式の右辺、左辺、関数の引数などで、変数の型が一致していない。 ②READ文で数値変数に文字データを読み込もうとした。 ③INPUT＃文で数値変数に文字データを読み込もうとした。	式の右辺、左辺の型を確認する。
20	RE error	エラー処理ルーチンに制御を移さなかったのにRESUME文がある。	RESUME文を使う位置を検討する。
21	PR error	①PASS設定時に、実行できないコマンド、操作をした。 ②ライトプロテクトをしてあるフロッピーディスクに書き込みを行なおうとした。	①PASSを解除する。 ②ライトプロテクトを解除して書き込み状態にする。
22	DA error	読むべきデータがないのにREAD文が実行された。	DATA文の確認をする。 READ文の確認をする。
23	FO error	①NEXT文に対するFOR文がない。 ②FOR~NEXTループ中にCLEAR文、ERASE文が入っている。	①NEXT文とFOR文の組み合わせを確認する。 ②ループ中のCLEAR文、ERASE文を削除する。
24	NX error	FOR文に対するNEXT文がない。	FOR文とNEXT文の組み合わせを確認する。
25	GS error	①GOSUB文とRETURN文が正しく対応していない。 ②飛び先にCLEAR文が入っている。	①GOSUB文とRETURN文の対応を確認する。 ②飛び先のCLEAR文を削除する。
26	FM error	フロッピーディスクのフォーマットが行なわれていない。 または、フォーマットが壊れている。	新しいフロッピーディスクは必ずフォーマットを行なう。
28	OP error	OPENされていないファイルを参照しようとした。 または、二重にOPENしようとした。	ファイルは必ずOPEN文を実行してから行なう。OPENしてあるファイルを再びOPENするには、一度CLOSEする。
29	AM error	インプットオープンに対してアウトプット系のコマンドを使おうとした。またはその逆。	インプット系のコマンドとアウトプット系のコマンドは、正しい使い方をする。

エラーコード	エラーメッセージ	エラー内容	対処方法
30	FR error	RS-232Cポートがフレーミングエラーを検出した。	RS-232Cの接続、データ転送方法を確認。
31	PO error	RS-232Cポートがパリティエラーまたはオーバーランエラーを検出した。	①RS-232Cの接続、データ転送方法を確認する。 ②転送速度を遅くする。
32	DF error	①FDDに対し未定義コマンドを送った。 ②ドライブ装置に異常がおこった。	①FDDに対するコマンドを確認する。 ②フロッピーディスクの内容は保証されません。 何度か試してもこのエラーが表示される場合は、弊社までご連絡ください。

※I/Oとは、Input/Outputの略で、各装置のデータをやりとりする部分を言います。

※I/Oデバイスとは、データの入力を行なう装置(インタフェースボックス、フロッピーディスクドライブ、プリンタなど)を指定します。

第3章
CASL編

CASLの紹介

ここで学ぶこと
☆コンピュータシステム
☆COMET
☆CASL

まとめ
◎コンピュータシステムには、ハードウェアとソフトウェアがあります。
◎COMETとは、通産省の情報処理技術者試験用に考えられた仮想コンピュータです。
◎CASLは、COMETで動くアセンブラ言語です。

コンピュータシステム

コンピュータシステムには、「ハードウェア」と「ソフトウェア」があります。

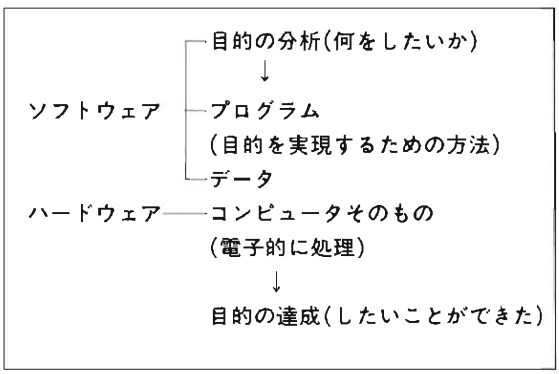
ハードウェアは、「コンピュータそのもの」、「コンピュータの構成要素」、「コンピュータを形作るLSI(大規模集積回路)やその他の物理的構成要素」といった幅広い意味を含む言葉です。ここでは「コンピュータの構成要素」として解説します。

コンピュータは、ハードウェアだけでは何も実行してくれません。ハードウェアに対して、「何々をしなさい」という指令をして、初めてコンピュータは仕事をします。この指令のことをソフトウェアといいます。

ソフトウェアも幅広い意味を持つ言葉で、「指令データ」そのものを意味することもあれば「指令データを作成し、コンピュータを利用する技術」を意味することもあります。

ハードウェアとソフトウェアを統合してコンピュータを使うとき、これらをまとめて「システム技術」といいます。

コンピュータの概念を右に示します。



COMET

情報技術者試験のアセンブラ言語CASLの動く仮想コンピュータCOMETは、ハードウェア構成について特に指定はなく、各処理系に委ねられています。本機のCASLモードは、試験の案内書に掲載されている仕様書を実現したものです。

CASL

仮想コンピュータCOMETで動くアセンブラ言語がCASLです。それでは、これからCASLについて説明していきましょう。

簡単なプログラム

ここで学ぶこと
☆ソースプログラム
☆ソースプログラムの入力

まとめ
◎CASLで書かれたプログラムを、「ソースプログラム」といいます。
◎ソースプログラムを作成・編集するには、CASLのメニュー画面でS キーを押します。

ソースプログラム

アセンブラ言語で書かれたプログラムのことを「ソースプログラム」といいます。このソースプログラムを機械語に翻訳するものが「アセンブラ」です。

［プログラム例］挨拶のプログラム

ラベル	命令コード	オペランド	説　明
REI	START	STAT	プログラムをSTATから開始
STAT	OUT	CHR, N5	CHR番地から(N5)＝5文字分の出力
	EXIT		プログラムの実行を終了
N5	DC	5	出力する文字数5を格納
CHR	DC	'コンニチハ'	出力する文字列を格納
	END		プログラムの終了

CASLのプログラムはラベル、命令コード、オペランドで構成されます。
ラベルはプログラムの開始番地やデータの格納番地を表わします。命令コードでコンピュータに命令を与え、オペランドには命令の実行に必要な情報を与えます。

このプログラムは、CHR番地から格納されている文字列「コンニチハ」を画面に出力するプログラムです。
では、このプログラムを入力して実行してみましょう。

プログラムの入力

ここで学ぶこと
☆ファイルエディタ
☆ファイルエディタの操作

まとめ
◎プログラムの入力はファイルエディタで行ないます。
◎新しいプログラムは、1行ごとに↵キーを押します。

ファイルエディタ

CASLのプログラムを作成したり修正したりする作業を「プログラムの編集」といい、編集を行なう機能をエディタといいます。また、作成したプログラム（ソースプログラム）は、本機のF0－F9のファイルエリアに記憶されます。

そこで、CASLのプログラムを編集するための機能をファイルエディタといいます。

本機を使ってCASLを学ぶには、ファイルエディタの操作に習熟することが大切です。

ファイルエディタの使い方

ファイルエディタの起動

プログラムを作成したり修正したりする場合には、まずファイルエディタを起動します。エディタの起動方法には、次の2通りがあります。

①CASLのメニュー画面で、[S]キーを押してプログラム入力状態にします。

②F.COMメニューから編集するファイルエリアをカーソルキー（↑↓←→）で指定した後、[E]キーを押します。

[操作方法]

電源をONにして、CASLモードに入ります。

操作　スライドスイッチをONにします。
（モード設定が[CAL]モード時）

```
_
```

操作　[MENU]
（メニューを表示させます）

```
                ( MENU )

1:F.COM  2:BASIC  3:C      4:CASL
5:ASMBL  6:FX     7:MODE
```

操作　[4]

```
                ( CASL )

F 0 1 2 3 4 5 6 7 8 9      184128
F2>Assemble/Source
```

CASLのメニュー画面になりました。このメニューから次のような機能が選択できます。

Aキー(Assemble)…ソースプログラムのアセンブル Sキー(Source) ……ソースの作成・編集

まずソースプログラムを入力するため、Sキーを押します。

操作　S

```
■
                                        (      1)
```

CASLエディタに入り、入力待ちの状態になりました。

操作　(1行目)　R E I TAB S T A R T TAB S T A T ↵
(2行目)　S T A T TAB O U T TAB C H R ' N 5 ↵
(3行目)　TAB E X I T ↵
(4行目)　N 5 TAB D C TAB 5 ↵
(5行目)　C H R TAB D C TAB SHIFT ' SHIFT カナ K O N N N I T I H A SHIFT カナ SHIFT ' ↵
(6行目)　TAB E N D ↵

これでプログラムが入力できました。

コラム●TABキーでプログラムを読みやすく●

CASLでは、「ラベル」「命令」「オペランド」の間に1文字以上の空白を空ける必要があります。また、「ラベル」を省略する場合は行頭に1文字以上の空白を空けなければなりません。

このとき、TABキーを利用して空白をつくると便利です。これにより、「ラベル」、「命令」、「オペランド」の先頭がそろった見やすいプログラムになります。

ソースプログラムの編集・訂正

ここで学ぶこと
☆プログラムの編集
☆文字(列)、行番号の検索

まとめ
◎入力状態によってカーソルの形が異なります。
◎SRCH、SHIFT LINEキーで、文字(列)、行の検索ができます。

カーソルについて

挿入モードと上書きモード

ファイルエディタでは、CALモードおよびBASICモードとは異なり、文字を入力すると、カーソルの位置に文字が挿入され、それ以降の文字は後に送られます。この状態を「挿入モード(インサートモード)」といいます。

挿入モードでINSキーを押すと、「上書きモード(オーバーライトモード)」に変わります。この上書きモードでは、文字を入力すると、カーソルの位置に文字が入力され、それまでカーソル位置にあった文字が消えます。

「上書きモード」でINSキーを押すと、挿入モードに戻ります。このように、INSキーを押すと、

挿入モード→上書きモード→挿入モード→上書きモード→……

のように順次切り替わります。

カーソルの形

ファイルエディタでは、カーソルは次のような形になります。

挿入モード		上書きモード	
英字	カナ	英字	カナ
■	▬	_	▬

カーソルの移動

ファイルエディタでは、作成したプログラムの先頭から末尾までのすべてをスクロールさせて見ることができます。

カーソルの移動には、↑↓←→キーを使う以外に次の操作ができます。

操作
①SHIFT L.TOP……カーソルのある行の行頭に移動します。
②SHIFT L.END……カーソルのある行の行末に移動します。
③SHIFT F.TOP……ファイルの先頭に移動します。
④SHIFT F.END……ファイルの末尾に移動します。

プログラムの編集

プログラムの修正

まず、修正したい箇所に↑↓←→キーでカーソルを移動します。修正の操作方法には、次の4通りがあります。

操作

①挿入する文字を入力……文字を挿入します。改行したいときはENTキーを押します。
②DEL……文字を削除します。
③INS……キーを押して上書きモードにします。正しい文字を入力します。
④SHIFT+LCAN……1行全部を削除します。

修正が終わったら、BRKキーを押します。

プログラムの中の文字(列)、行番号の検索

プログラムの中の文字(列)や行番号による検索ができます。プログラムの修正に便利な機能です。

文字列「DC」を検索する場合

操作　SRCH

```
REI       START     STAT↵
STAT      OUT       CHR.N5↵
          EXIT↵
search?_                          ( 1 )
```

操作　D C ↵
(検索したい文字列を入力)

```
N5        DC        5↵
CHR       DC        'コンニチハ'↵
          END↵
                                  ( 4 )
```

操作　NEXT(次の「DC」にカーソルが移動します)

```
CHR       DC        'コンニチハ'↵
          END↵
                                  ( 5 )
```

行番号３を検索する場合

操作 SHIFT LINE

```
REI       START     STAT↵
STAT      OUT       CHR.N5↵
          EXIT↵
line?_                          (     1)
```

操作 3 ↵

（検索したい行番号を入力）

```
■         EXIT↵
N5        DC        5↵
CHR       DC        'コンニチハ'↵
                                (     3)
```

３行目が表示されます。

ファイルエディタを使うときの注意

ファイルエディタによりF0～F9のファイルを編集できます。その際、１行で256文字以上のデータを使えます。しかし、BASICでは１行で扱える文字数は255文字までなので、１行に256文字以上の文字データを扱う場合は、次のような操作はしないでください。

また、ファイルエディタでは↵キーを押さなくてもデータが入力できますが、このようなデータが最後にあるようなファイルをBASICで扱う場合も同様となります。

①F0～F9エリアに、このようなデータがある場合
READ#、WRITE#などのデータバンク用命令を実行しないでください。
②I／Oに、このようなデータがある場合
BASICエリアにロードできません。
③F0～F9エリアのこのようなデータは、BASICのP0～P9エリアにファイルコピーしないでください。

注）「I／O」とは、入出力機器を表わします。

ソースプログラムのアセンブル

ここで学ぶこと
☆ソースプログラムの訂正
☆プログラムのアセンブル

まとめ
◎メニューに戻りたいときは、SHIFT 終了 と操作します。
◎ソースプログラムをアセンブルすることにより、実行可能なオブジェクトプログラムができあがります。

ソースプログラムのアセンブル

CASLで書かれたソースプログラムをコンピュータが実行できるオブジェクトプログラムに翻訳することを、「アセンブル」といいます。

アセンブルするには、メニュー画面でAキーを押します。

操作　A

```
〈 CASL 〉

Go/Dump/Source/Print/Trace
PC:0000 REI      [off] [off]
```

アセンブルが正常に終了すると、上の画面が表示されます。エラーが発生した場合は170ページを参照してください。

オブジェクトプログラム

ここで学ぶこと
☆オブジェクトプログラム
☆アセンブル後のメニュー
☆オブジェクトプログラムの実行

まとめ
◎ソースプログラムをアセンブルしたものを、「オブジェクトプログラム」といいます。
◎アセンブルが正常に終了すると、画面に実行のために必要なメニューが現われます。
◎オブジェクトプログラムを実行するためには二つの方法があります。

オブジェクトプログラム

ソースプログラムをアセンブラによって機械語に翻訳したものを、オブジェクトプログラムといいます。オブジェクトプログラムとは、コンピュータが直接理解できるプログラムのことです。

アセンブル後のメニュー

ソースプログラムをアセンブルして、エラーを生じることなしに正常に終了した場合は、画面は右のように変わります。

```
( CASL )

Go/Dump/Source/Print/Trace
PC:0000 RET      [off] [off]
```

それぞれのキーには、次のような機能が割り当てられています。

Gキー(Go)	オブジェクトプログラムの実行画面に移ります。
Dキー(Dump)	ダンプのメニュー画面に移ります。
Sキー(Source)	エディタに入り、ソースプログラムを表示します。
Pキー(Print)	プリンタのON/OFFを指定します。
Tキー(Trace)	トレースのON/OFFを指定します。
↵キー(実行)	オブジェクトプログラムを実行します。

オブジェクトプログラムの実行

オブジェクトプログラムを実行するには、次の二つの方法があります。ここでは161ページのプログラムを実行しましょう。

(a) ↵キーによる実行

↵キーを押すと、右のように画面が変わります。この画面で↵キーを押すと、メニューの画面に戻ります。

```
コンニチハ
```

(b) Gキーによる実行

Gキーを押すと、右のように画面が変わります。ここで実行開始アドレスを入力して↵キーを押すと、実行を開始します。

```
PC:0000   REI

GO?_
```

実行開始アドレスを省略した場合は、最上行に表示されているアドレスから実行されます。また、アドレス入力は10進、16進、ラベル入力の3種類が可能です。

エラーメッセージ

ここで学ぶこと
☆アセンブル時のエラー
☆オブジェクトプログラム実行時のエラー

まとめ
◎ソースプログラムに誤りがあると、アセンブルしたときにさまざまな種類のエラーが発生します。
◎オブジェクトプログラムを実行中にも、メモリーの容量の不足などでエラーが生じます。

アセンブル時のエラー

ソースプログラムに誤りがあってアセンブルできないときは、エラーを検出して次のようなエラーメッセージを表示します。

エラーメッセージ	エ ラ ー 内 容
OM error	オブジェクトを作成するためのメモリー容量が確保できない
LA error	ラベルの二重定義などラベル指定の誤り
OC error	オペコードエラー（命令語の誤り）
OR error	オペランドエラー
SO error	STARTやENDがない、または書式が違うなどのソースエラー

オブジェクトプログラム実行時のエラー

実行中にエラーが生じたときは、エラーを検出して次のようなエラーメッセージを表示します。

エラーメッセージ	エ ラ ー 内 容
Bad code	オブジェクト中に実行できないコードが入ってきた
Out of address	確保していないアドレスにアクセスしようとした
Stack over	スタックエリアがオーバーした

エラーメッセージが表示されたときは、[ESC]キーを押してメニュー画面に戻り、[S]キーを押してCASLエディタに入ってから、ソースプログラムを修正してください。

割り算のプログラム

ここで学ぶこと
☆割り算のプログラム

まとめ
◎ここでは実用的なプログラムの例として、割り算のプログラムを挙げます。

［プログラム例］ 割り算のプログラム（A÷B＝SHO余りAMARI）

ラベル	命令コード	オペランド	説　明
JOZAN	START		割り算(除算)プログラムを開始
	LEA	GR1, 0	GR1に商を入れる、初期値0
	LD	GR0, A	GR0にA番地の内容を入れる
LOOP	SUB	GR0, B	GR0からB番地の内容を引く
	JMI	ANS	結果が負ならば、ANS番地に飛ぶ
	LEA	GR1, 1, GR1	GR1に1を加える
	JMP	LOOP	LOOP番地に飛ぶ
ANS	ADD	GR0, B	GR0にB番地の内容を加える
	ST	GR0, AMARI	GR0を余りとしてAMARIに格納
	ST	GR1, SHO	GR1を商としてSHOに格納
	EXIT		プログラムの実行終了
A	DC	13	割られる数13
B	DC	5	割る数5
SHO	DS	1	商を格納する番地
AMARI	DS	1	余りを格納する番地
	END		プログラムの終了

割り算のアルゴリズム

CASLには割り算を直接行なう命令コードがないため、割り算を行なうためには引き算を繰り返し行ないます。

このプログラムはA÷Bを行ない、商と余りを出すプログラムです。

上の表の例では割られる数13がA番地に格納されており、割る数5がB番地に格納されています。したがって、13÷5を行ない、結果として商が2、余りが3になるわけです(13÷5＝2余り3)。

このプログラムでは、割られる数をGR0に入れ、LOOP行からの4行で割られる数13から割る数5を順次引いていきます。引けなくなったら（引いた結果が負になってしまったら）ANS番地にジャンプし、最後に結果をSHO番地とAMARI番地に格納します。

プログラムを入力して、アセンブルし実行してみてください。エラーが発生した場合は、CASLエディタでソースプログラムを修正します。

ご注意
◎このプログラムを次ページ以降の「プログラムのダンプ」、「プログラムのトレース」のプログラム例にします。

プログラムのダンプ

ここで学ぶこと
☆ダンプ
☆オブジェクトプログラムのダンプ
☆ブレークポイントの設定

まとめ
◎主記憶装置の内容やレジスタの内容を出力することを、ダンプ(Dump)といいます。
◎ダンプのメニュー画面でOキーを押すと、オブジェクトプログラムのダンプができます。
◎ダンプのメニュー画面でRキーを押すと、レジスタのダンプができます。
◎ダンプのメニュー画面でBキーを押すと、ブレークポイントの設定ができます。

ダンプ

主記憶装置の内容やレジスタの内容を出力することを、ダンプ(Dump)といいます。これにより、オブジェクトプログラムを直接目で見たり、レジスタの内容を確認することができます。ダンプの指定は、アセンブルした後のメニュー画面で行ないます。

この画面でDキー(Dump)を押すと、ダンプのメニューが表示されます。

アセンブル後のメニュー

```
〈 CASL 〉

Go/Dump/Source/Print/Trace
PC:0000 JOZAN  [off] [off]
```

ダンプメニュー

```
〈 CASL 〉

Object/Register/Break point
```

オブジェクトプログラムのダンプ

ダンプのメニュー画面でOキーを押すと、オブジェクトプログラムのダンプができます。ここでアドレス値を入力して↵キーを押すと、そのアドレス値からオブジェクトの内容を表示します。アドレス入力を省略して↵キーを押すと、0番地から表示します。

ダンプのキー操作

↵または↓キー	下スクロール
↑キー	上スクロール

オブジェクトのダンプ

```
0000      JOZAN     1210
0001                0000
0002                1000
0003                0014
```

レジスタのダンプ

ダンプのメニュー画面でRキーを押すと、レジスタのダンプができます。

下の画面はレジスタの初期値を示しています。ここで割り算のプログラムを実行して、レジスタの変化を見てみましょう。

レジスタの初期値

```
GR0：0000  0          GR1：0000  0
GR2：0000  0          GR3：0000  0
GR4：FFFF  65535      FR  ：0
GR0?_
```

まず、SHIFT 英数 キーでCASLのメニュー画面に戻った後アセンブルし、G↵と操作してプログラムを実行します。続けてDキーの後にRキーを押すと、ダンプの変化を見ることができます。

レジスタの変化値

```
GR0：0003  3          GR1：0002  2
GR2：0000  0          GR3：0000  0
GR4：FFFF  65535      FR  ：0
GR0?_
```

ブレークポイントの設定

ダンプのメニュー画面でBキーを押すと、ブレークポイントの設定ができます。ブレークポイントを設定することによって、オブジェクトプログラムの実行を中断させ、その時点でのレジスタとフラグの値を直接見ることができます。

ブレークポイントの解除は、アドレス入力時に−1↵と操作します。

ご注意

◎このページの説明は、前ページの「割り算のプログラム」をアセンブルし、ダンプしたものです。

コラム●オブジェクトとレジスタのダンプの訂正●

オブジェクトとレジスタのダンプを訂正して、再実行させることができます。この方法では、ソースプログラムの内容に関わらず実行されます。ダンプを訂正したときは、ソースプログラムの予想される実行結果と異なる結果が現われますのでご注意ください。

レジスタダンプの訂正

レジスタダンプを行なうと、画面の最下行に「GR0?_」というメッセージが現われます。ここで16進数を入力し↵キーを押すと、GR0にデータを与えることができます。訂正しないときは、↵キーだけを押します。このように、次々とGR1～GR4までにデータを与えることができます。

オブジェクトの訂正

オブジェクトのダンプを表示しているときに⇨キーまたは⇦キーを押すと、画面の最下行に「data?_」というメッセージが現われます。ここで、16進数を入力し↵キーを押すと、画面の最上行に表示されているアドレスのオブジェクトを訂正することができます。訂正が終わったらBRKキーを押してメニュー画面に戻って再実行すると、訂正後のオブジェクトが実行されます。

プログラムのトレース

ここで学ぶこと
☆トレースの方法

まとめ
◎オブジェクトプログラムの実行過程のレジスタ値とフラグの値を、1ステップずつ見ることが「トレース」です。

トレース

オブジェクトプログラムの実行過程のレジスタ値とフラグの値を、1ステップずつ見ることを「トレース」といいます。本機のCASLでは、表示画面またはプリンタにトレースの結果を出力することができます。

トレースの指定は、CASLのメニュー画面で行ないます。

操作 Tキーを押します。

```
( CASL )

Go/Dump/Source/Print/Trace
PC:0000 JOZAN  [off] [on]
```

トレースの指定がされると、上のような画面になります。この状態でオブジェクトプログラムを実行すると、1ステップずつ実行過程を表示画面で見ることができます。

なお、このときPキーを押してPrintを〔on〕としておけば、トレースの内容はプリンタにも出力されます。

ここでは、「割り算のプログラム」を実行したトレース結果の一部を次に示しておきます。

操作 G↵

```
0000  JOZAN  LEA  GR1,0000
      0000 0000 0000 0000 FFFF 1
0002          LD   GR0,0014
      000D 0000 0000 0000 FFFF 1
0004  LOOP   SUB  GR0,0015
      0008 0000 0000 0000 FFFF 0
0006          JMI  000C
      0008 0000 0000 0000 FFFF 0
0008          LEA  GR1,0001,GR1
      0008 0001 0000 0000 FFFF 0
000A          JMP  0004
      0008 0001 0000 0000 FFFF 0
0004  LOOP   SUB  GR0,0015
      0003 0001 0000 0000 FFFF 0
                ⋮
      以下、表示は省略します。
```

表示の見方

```
PC  ラベル   命令コード オペランド
 ↓     ↓       ↓       ↓
0000  JOZAN   LEA   GR1,0000
     0000  0000  0000  0000  FFFF  1
      ↑     ↑     ↑     ↑     ↑    ↑
     GR0   GR1   GR2   GR3   GR4   FR
```

GR0
〜 } レジスタの内容
GR4

表　示	本機	仕様書
FR　フラグの値	0	00
	1	01
	2	10

仕様書とは試験センターのCOMET仕様書のことです。

操作のまとめ

ここで学ぶこと

☆CASLの基本操作

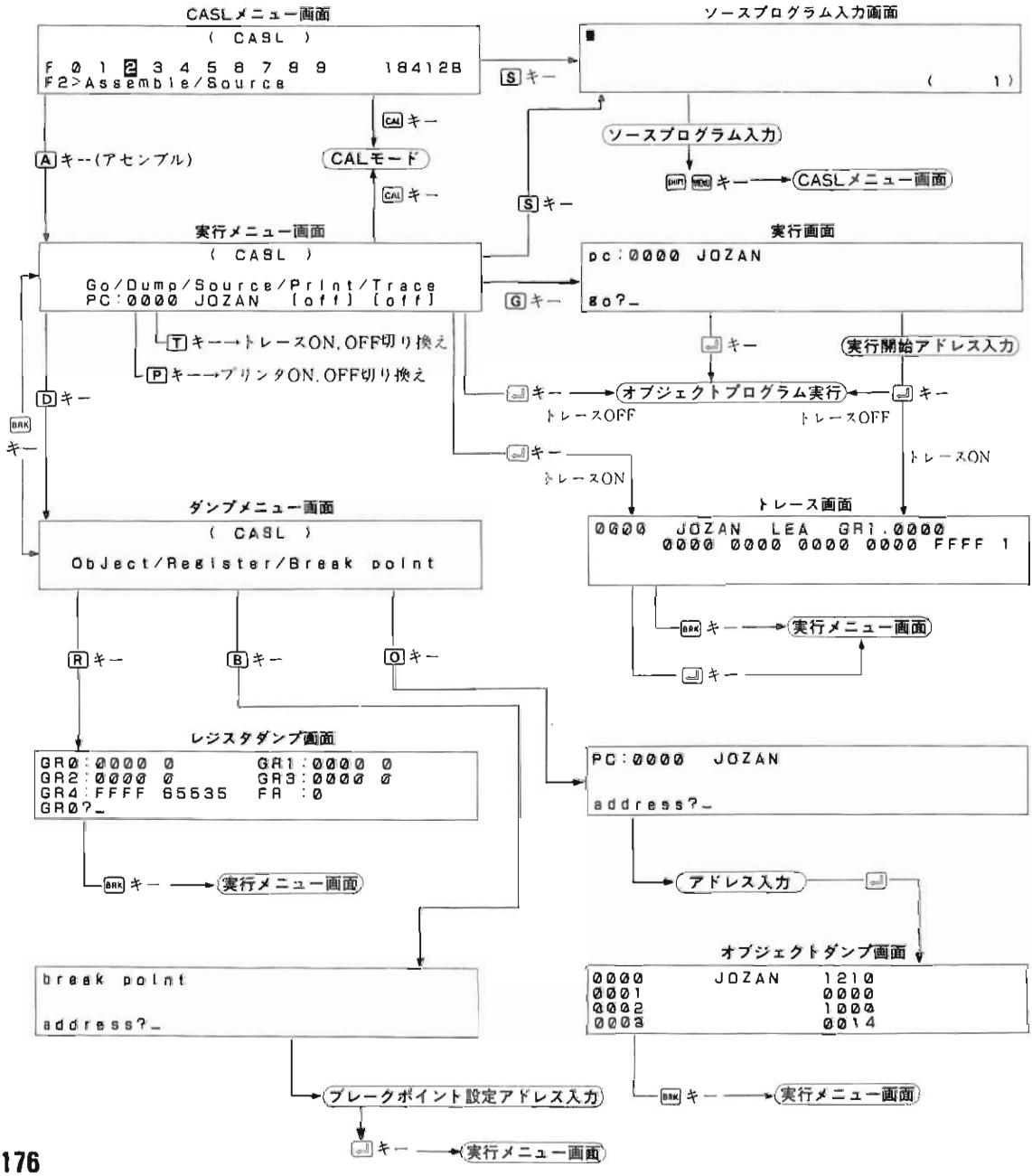
まとめ

◎CASLモードの手順は、以下のようになります。

(1)まずソースプログラムを作ります。

(2)デバッグして、アセンブルします。

(3)オブジェクトプログラムを実行させます。



COMETの構成

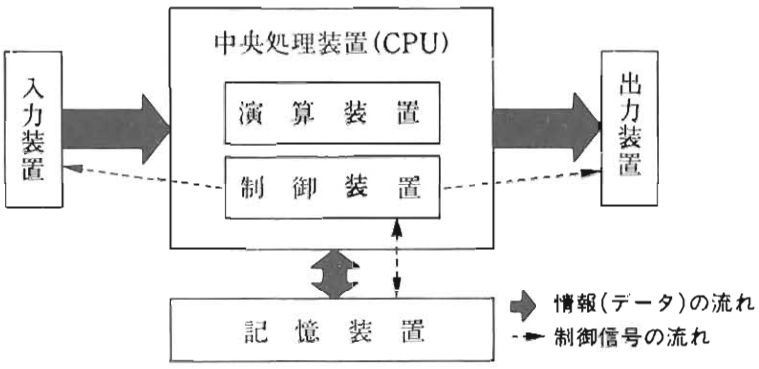
ここで学ぶこと
☆コンピュータの構成
☆中央処理装置の構成
☆記憶装置
☆入出力装置

まとめ
◎一般にコンピュータは、中央処理装置、入出力装置、記憶装置から構成されています。
◎中央処理装置では、実際の計算を行ないます。
◎記憶装置には、プログラムとデータが保存されます。
◎中央処理装置は、入出力装置を通して外部と対話します。

コンピュータの構成

コンピュータは、一般に中央処理装置、出力装置、記憶装置から構成されています。そして、中央処理装置は、演算装置、レジスタ、制御装置から構成されています。次にコンピュータの構成を図示します。このようなハードウェア構成を持つものを、COMETと呼びます。

コンピュータの構成



中央処理装置の構成

中央処理装置はCPU (Central Processing Unit) と呼ばれ、コンピュータシステムの中核を構成します。中央処理装置は、制御装置、演算装置、レジスタから構成されています。レジスタについては後のページで説明します。

制御装置	記憶装置に記録されている命令を読み出し解読して、演算装置や主記憶装置、入出力装置に信号を送ります。このとき、各装置のタイミングを取って動作の制御を行ないます。
演算装置	制御装置によって解読されたプログラムの指令にしたがって、判断や演算を行ないます。

記憶装置

記憶装置には、主記憶装置と補助記憶装置の2種類があります。

主記憶装置は実行しているプログラムやデータを記憶する場所で、それらプログラムやデータを中央処理装置とやりとりします。本機では、メインメモリーに当たります。

補助記憶装置は、限界のある主記憶装置の容量を補助する役割を持っています。

入出力装置

プログラムやデータ、演算結果などを入出力するための装置です。

本機では、本体のキーボードが入力装置に本体の表示画面や外部に接続できるプリンタなどが出力装置に当たります。

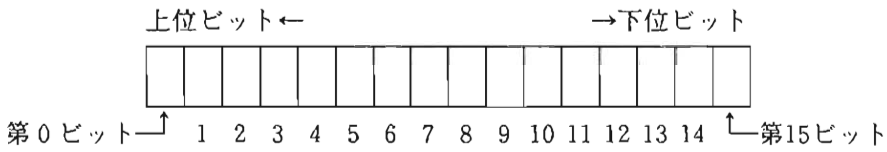
データの構成とレジスタ

ここで学ぶこと
☆データの構成
☆COMETのレジスタとカウンタ
☆汎用レジスタ
☆フラグレジスタ
☆プログラムカウンタ

まとめ
◎COMETでは、データは1語＝16ビット単位で扱われます。
◎COMETには、3種類のレジスタが用意されています。
◎汎用レジスタは、様々な計算用に使われます。
◎フラグレジスタは、演算結果の符号を保持します。
◎プログラムカウンタは、次に実行すべき命令の先頭アドレスを示します。

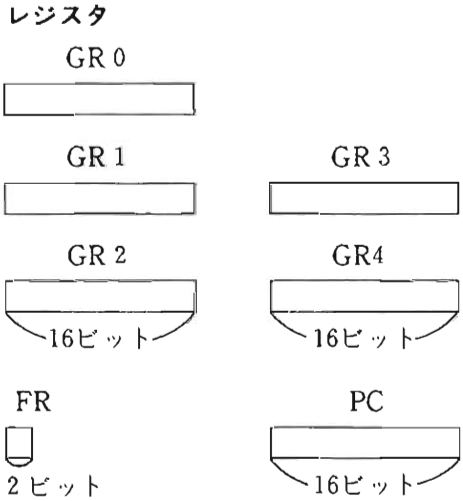
データの構成

COMETは、1語16ビット単位のコンピュータです。1語は、次のような構成になっています。



COMETのレジスタとカウンタ

COMETには、汎用レジスタ、フラグレジスタ、プログラムカウンタの3種類のレジスタが用意されています。



汎用レジスタ(General Resister 略称GR)

汎用レジスタは、16ビットの大きさを持つレジスタです。すべてでGR0, GR1, GR2, GR3, GR4の5個があります。

汎用レジスタには以下のような機能があり、それぞれ使用できるレジスタが決められています。

演算用レジスタ(GR)	算術演算、論理演算、比較演算、シフト演算などに用います。GR0, GR1, GR2, GR3, GR4のすべてが使えます。
インデックスレジスタ(XR)	アドレス修飾(後述)に使います。GR1, GR2, GR3, GR4の4つが使えます。
スタックポインタ(SP)	スタック(後述)のアドレスを指定します。スタックポインタとしては、GR4だけが使えます。

フラグレジスタ(Flag Register 略称FR)

FRはCOMET内に1個だけあり、2ビットで構成されており、演算結果の符号を保持しています(右表)。また、比較演算命令(CPA, CPL)のとき、FRはGRの内容と比較対象を比べた結果によって決まります。

演算結果 比較結果	正 >	ゼロ =	負 <
FRの値(2進)	00	01	10
Z-1の表示	0	1	2

プログラムカウンタ(Program Counter 略称PC)

PCは、次に実行すべき命令の先頭アドレスを格納する16ビットの大きさのレジスタです。COMET内に1個あります。

命令語の構成

ここで学ぶこと
☆命令語の構成

まとめ
◎命令語は、一つが32ビットの大きさを持っています。

命令語の構成

命令語は32ビットの大きさ、すなわち２語の長さを持っています。その構成は以下のようになっています。

命令語の構成

ビット位置

	0 1 2 3	4 5 6 7	8 9 10 11	12 13 14 15
第１語	主OPフィールド	副OPフィールド	GRフィールド	XRフィールド
第２語	adrフィールド			

OPフィールド	８ビットで構成されていて、命令コードの種類を表わします。
GRフィールド	４ビットで構成されていて、命令の対象となる汎用レジスタを指定します。
XRフィールド	４ビットで構成されていて、アドレス修飾するためのインデックスレジスタの番号を指定します。
adrフィールド	16ビットで構成されていて、命令の対象となる主記憶装置上のアドレスを指定します。また、定数を表わすこともあります。シフト命令の時には、シフトするビット数を示します。

下記の表は、各命令語を、アセンブリ言語から機械語に変換した例の一部です。試験センターの仕様書では変換方法を規定していませんので、本機では独自に定めています。

アセンブリ言語	第１語（16進）			第２語（16進）
	OP	GR	XR	adr
LD GR0, 10, GR1	1	0	0	1	0 0 0 A
SUB GR1, -1, GR2	2	1	1	2	F F F F
AND GR0, 10	3	0	0	0	0 0 0 A
CPL GR1, -1, GR2	4	1	1	2	F F F F
SLA GR0, 10, GR1	5	0	0	1	0 0 0 A
JPZ 10, GR1	6	0	0	1	0 0 0 A
PUSH -1, GR3	7	0	0	3	F F F F
POP GR3	7	1	3	0	0 0 0 0
CALL 10, GR1	8	0	0	1	0 0 0 A
RET	8	1	0	0	0 0 0 0

アドレス修飾と実効アドレス

ここで学ぶこと
☆アドレス
☆実効アドレス

まとめ
◎主記憶装置の1語1語に付けられた通し番号をアドレス(番地)と呼びます。
◎命令語のXRの内容とadrをアドレス加算したものが、実効アドレスです。

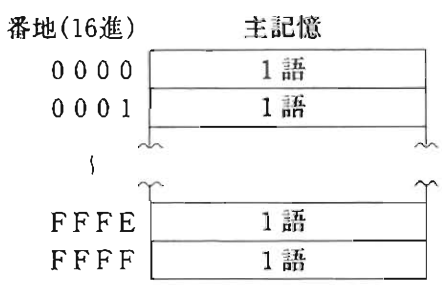
アドレス

主記憶装置は、16ビット1語を最大$2^{16}=65536$語まで装備できます。

アドレスは、0から65535までの整数値で与えます。アドレスの表記は、通常16進数で0000からFFFFまでとなります。

COMETのアドレス空間を右図に示します。

COMETのアドレス空間



中央処理装置は主記憶装置内の命令を解読し、①データのやりとりを行なう、または②実行制御を移す（ジャンプする）ためにアドレスを指定しなければなりません。

実効アドレス

命令を解読して実行するとき、その対象となるアドレスは、命令語のXRフィールドとadrフィールドで指定します。これをアドレス修飾といい、指定されたアドレスを「実効アドレス」といいます。

実効アドレスは、XRレジスタの値とadrの値のアドレス加算で計算されます。

例　　LD　GR0, 10, GR1

この例ではXRがGR1、adrが10になっています。したがって、この場合は「GR1の内容+10」が実効アドレスとなります。

もしGR1の内容が3ならば、実効アドレスは3+10で13になります。

ところが、GR1の内容がFFFA（16進）の場合はどうでしょうか。実際に加算してみると、

FFFA+000A（10進数で10）=10004

となりますが、5桁目の1が無視されて0004が実効アドレスになります。このように5桁目にあふれた数を無視する加算を、「アドレス加算」といいます。

XRフィールドが省略されているときは、adrの値がそのまま実効アドレスになります。

スタック

ここで学ぶこと
☆スタック
☆スタックポインタ

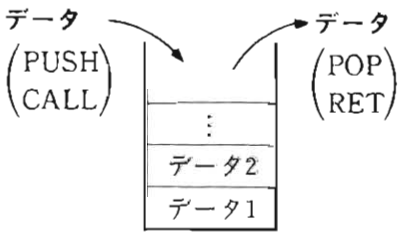
まとめ
◎スタックは、レジスタやメモリー内容の一時退避や、呼び出したサブルーチンから復帰するための戻り番地の保存に使います。
◎スタックに保存されているデータのトップ番地を示しているのが、スタックポインタです。

スタック

スタックは、レジスタやメモリー内容の一時退避や、呼び出したサブルーチンから復帰するための戻り番地の保存に使います。

スタックはデータの井戸のようなもの（下図参照）で、スタックにデータを入れると、それがどんどん積み上がっていく形になります。したがって、最後に入れたデータを最初に出すことができ、最初に入れたデータが最後に出されることになります。

スタックにデータを入れる命令としてPUSHとCALL、それに対応してスタックからデータを出す命令としてPOPとRETがあります。



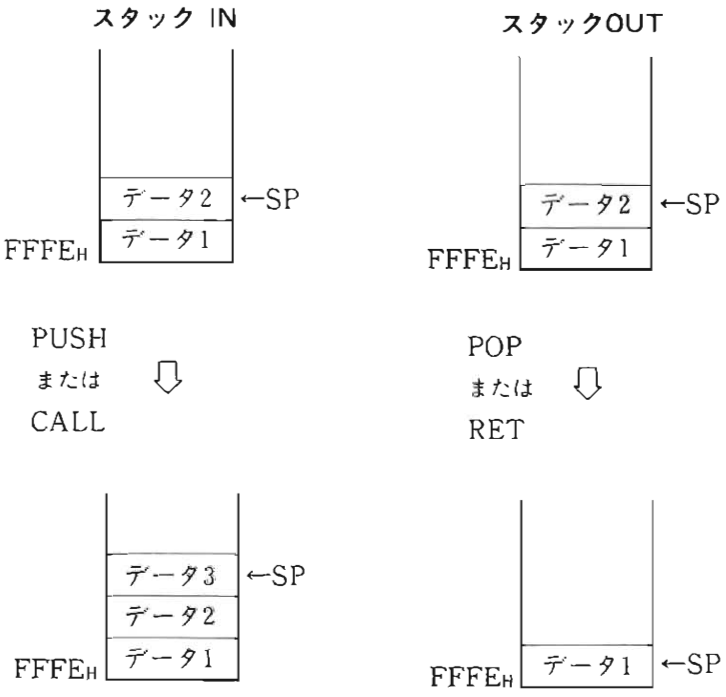
スタックポインタ(SP)

汎用レジスタGR4は、スタックポインタとして使われます。

スタックポインタは、スタックのデータトップの位置を示しています。その初期値はFFFF(16進)になっています。

スタックポインタの内容は、PUSH命令、POP命令、CALL命令、RET命令によって変化します。

スタックポインタの模式図



キャラクターセット

ここで学ぶこと
☆COMETのキャラクターセット

まとめ

◎COMETでは英数字、カナ文字、図形文字が使用でき、それぞれコード番号を持っています。

キャラクターセット

COMETは、JIS X0102ローマ字・カタカナ用8単位符号で規定するキャラクターセットを持っています。

本機では、下表のキャラクターが使用できます。

情報処理試験においては、JIS X0102の表の一部だけが示されています。試験問題において、表にない文字が必要なときは、その文字のビット構成が問題文の中で与えられます。

8単位符号の構成

b8	b7	b6	b5	b4	b3	b2	b1	行＼列	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
				0	0	0	0	0			8P								10/0					
				0	0	0	1	1																
				0	0	1	0	2																
				0	0	1	1	3																
				0	1	0	0	4																
				0	1	0	1	5																
				0	1	1	0	6	32個の機能キャラクターの領域		94個の図形キャラクターの領域						32個の機能キャラクターの領域		94個の図形キャラクターの領域					
				0	1	1	1	7																
				1	0	0	0	8																
				1	0	0	1	9																
				1	0	1	0	10																
				1	0	1	1	11																
				1	1	0	0	12																
				1	1	0	1	13																
				1	1	1	0	14																
				1	1	1	1	15																15/15

備考：図中のb_1～b_8は、8単位符号ビットの組み合わせを示す。

ローマ字・片仮名用8単位符号

列 / 行	00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
0	NUL	TC_7(BLE)	(SP)	0	@	P	'	p	↕	↕	未定義	ー	タ	ミ	↕	↕
1	TC_1(SOH)	DC_1	!	1	A	Q	a	q	↕	↕	。	ア	チ	ム	↕	↕
2	TC_2(STX)	DC_2	〃	2	B	R	b	r	↕	↕	「	イ	ツ	メ	↕	↕
3	TC_3(ETX)	DC_3	#	3	C	S	c	s	↕	↕	」	ウ	テ	モ	↕	↕
4	TC_4(EOT)	DC_4	$	4	D	T	d	t	↕	↕	、	エ	ト	ヤ	↕	↕
5	TC_5(ENQ)	TC_8(NAK)	%	5	E	U	e	u	↕	↕	・	オ	ナ	ユ	↕	↕
6	TC_6(ACK)	TC_9(SYN)	&	6	F	V	f	v	未	未	ヲ	カ	ニ	ヨ	未	未
7	BEL	TC_{10}(ETB)	'	7	G	W	g	w	定	定	ァ	キ	ヌ	ラ	定	定
8	FE_0(BS)	CAN	(	8	H	X	h	x	義	義	ィ	ク	ネ	リ	義	義
9	FE_1(HT)	EM	)	9	I	Y	i	y	↕	↕	ゥ	ケ	ノ	ル	↕	↕
10	FE_2(LF)	SUB	*	:	J	Z	j	z	↕	↕	ェ	コ	ハ	レ	↕	↕
11	FE_3(VT)	ESC	+	;	K	[	k	{	↕	↕	ォ	サ	ヒ	ロ	↕	↕
12	FE_4(FF)	IS_4(FS)	,	<	L	¥	l	¦	↕	↕	ャ	シ	フ	ワ	↕	↕
13	FE_5(CR)	IS_3(GS)	−	=	M	]	m	}	↕	↕	ュ	ス	ヘ	ン	↕	↕
14	SO	IS_2(RS)	.	>	N	^	n	~	↕	↕	ョ	セ	ホ	゛	↕	↕
15	SI	IS_1(US)	/	?	O	_	o		↕	↕	ッ	ソ	マ	°	↕	未定義

オペレーティングシステム

ここで学ぶこと
☆オペレーティングシステム

まとめ
◎命令の実行に関するシステムを、「オペレーティングシステム」といいます。

オペレーティングシステム

情報処理技術者試験のアセンブラ言語仕様書には、オペレーティングシステムについて次のように記載されています。本機のCASLもそれに準拠しています。

オペレーティングシステム

プログラムの実行に関して、利用者プログラムとオペレーティングシステムとの間に、次の取り決めがあります。

(1)プログラムの起動

プログラムはオペレーティングシステムにより起動されます。プログラムが格納される番地は不定とするが、(2)で述べるスタック領域も含めてプログラムの実行に支障を与えないものとします。

(2)スタック領域

オペレーティングシステムは、プログラムの起動時にスタック領域を確保し、スタック領域の最後のアドレスに1をアドレス加算した値をSPに設定します。この領域は、プログラムでスタックとして利用されます。

スタック領域は、試験問題のプログラムで使用するのに十分な容量が確保されているものとします。

(3)入出力装置の割り当て

IN命令に対応する入力装置、OUT命令に対応する出力装置の割り当ては、プログラムの実行に先立ってオペレーターが行ないます。入出力装置には、コンソール表示装置、タイプライター、カード読み取り装置、カードせん孔装置などがあります。

(4)入出力処理

各種の入力装置からのデータの入出力はIN、OUTマクロ命令で行ないますが、媒体や装置による入出力手続きの違いはすべてオペレーティングシステムが吸収し、システムの標準手続き（異常処理を含む）、標準形式で入出力を行ないます。したがって、このマクロ命令の使用者は、入出力装置の違いを意識する必要はありません。

(5)未定義ラベル

アセンブラは、機械語命令のオペランド中のラベル、DC命令のアドレス定数のラベルの中で、そのプログラム内で定義されていないラベルを、他のプログラムのSTART命令のラベル（他のプログラムの入口名）と解釈します。

この場合、アセンブラはアドレスの決定を保留し、その決定をオペレーティングシステムに任せます。オペレーティングシステムは、実行に先立って他のプログラムのSTART命令のラベルとの結合処理を行ない、アドレスを決定します（プログラムの連結）。

命令の形式

ここで学ぶこと
☆ラベル
☆命令コード
☆オペランド
☆注釈(コメント)

まとめ
◎命令語は、ラベル欄、命令コード欄、オペランド欄、注釈欄から構成されています。
◎ラベルは、6文字以内の英数字です。
◎命令コードは、命令の種類を表わします。
◎オペランドは、命令コードの修飾語です。
◎コメントは、プログラムをわかりやすくします。

以下にソースプログラムのコーディング例を示します。

ラベル欄	命令コード欄	オペランド欄	注釈(コメント)欄
REI	START	STAT	;プログラム　ノ　ハジメ
STAT	OUT	CHR, N5	;モジレツ　ノ　シュツリョク
	EXIT		;ジッコウシュウリョウ
N5	DC	5	;テイスウ5
CHR	DC	'コンニチハ'	;モジレツ
	END		;プログラム　ノ　オワリ

ラベル

ラベル欄にはラベルを書きます。必要がなければ省略できます。

ラベルは、6文字以内の英数字で書かれなければなりません。

行の第1文字目から書かれている文字列は、ラベルと見なされます。

ラベルの先頭の文字は、英文字の大文字でなければなりません。第2文字以降は、英大文字、数字のいずれでもかまいません。

計算機の内部では、命令に付けられたラベルは、その命令語または領域の先頭のアドレス番地を表わしています。

命令コード

命令コード欄には、命令コードを書きます。

ラベルを付けたときには、ラベルのあとに少なくとも1文字分以上の空白が必要です。

ラベルを付けないときは、第2文字以降の任意の位置から書き始めます。

オペランド

オペランド欄には、オペランドを書きます。

命令コードのあとには、少なくとも１文字分の空白が必要です。最大72文字目までがオペランド欄として使用できます。次の行に続けて書くことはできません。

注釈

注釈欄には処理系の許す任意の文字でコメントを書くことができます。本機では、英文字とカナが使えます。

セミコロン「;」以降が注釈行となります。第１文字目に「;」があるときと、「;」以降に空白しかないときは、その行全体が注釈行となります。

命令の種類と表記法

ここで学ぶこと
☆命令の種類
☆命令の表記法

まとめ
◎擬似命令は、アセンブラに指示を与える命令です。
◎CASLの機械語命令は23種類あり、翻訳されると機械語は逐一対応されます。
◎マクロ命令は、入出力と終了のための命令です。
◎命令の表記には、さまざまな決まりがあります。

命令の種類

CASLの命令には大きく分類して、擬似命令、機械語命令、マクロ命令の3種類があります。

擬似命令

擬似命令は、アセンブラの制御、定数の定義、プログラムの連結に必要なデータ生成などを行ないます。擬似命令にはSTART、END、DC、DSの4種類があります。

機械語命令

機械語命令は、ハードウェアCOMETの命令で中央処理装置によってそのまま認識されて実行されます。機械語命令はLD、ST、LEAなど23種類あります。

マクロ命令

マクロ命令には、入出力(IN, OUT)と終了(EXIT)のための3種類の命令があります。

命令の表記法

次ページ以降の各命令の解説では、次の表記法を用います。

LABEL	ラベル欄に書くラベルです。
GR	汎用レジスタ(0≦GR≦4)、またはGRn(0≦n≦4)。
XR	インデックスレジスタ(1≦XR≦4)、またはGRx(1≦x≦4)。
SP	スタックポインタGR4。
adr	ラベル名または10進定数。 ラベル名のときは、ラベルに対応する番地(数値)を示します。 10進定数は、次の範囲で使用できます。 -32768≦adr≦65535
実行アドレス	「adr」と「XRの内容」とのアドレス加算値。またはその値が示す番地。
(X)	X番地の内容。Xがレジスタの時はレジスタの内容です。
〔 〕	〔 〕に囲まれた部分が省略可能であることを示します。 XRを省略した場合、インデックスレジスタによる修飾は行なわれません。

命令一覧

START、END、DC、DS 擬似命令

命令名	開始　START	書　　式
		LABEL START〔adr〕
機能	プログラムの先頭に必ず書かなければなりません。adrのラベル名の番地から実行され、省略されるとプログラムの先頭から実行されます。	
命令名	終了　END	書　　式
		END
機能	プログラムの終了を表わします。プログラムの最後やサブルーチンの最後には、必ず書かなければなりません。ラベル、オペランドはありません。	

メインプログラムの記述例

```
MAIN START
    プログラム
        EXIT
        END
```

サブルーチンの記述例

```
SUBR START
    プログラム
        RET
        END
```

メインプログラムを記述するときは、まず先頭にSTART命令を書きます。次にプログラムの内容を書き、最後に処理を終了させるEXITとENDを書きます。

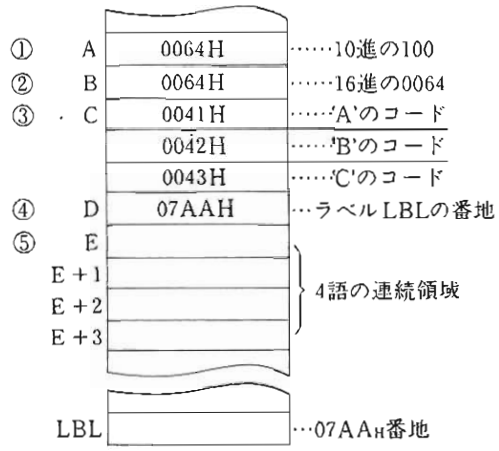
サブルーチンも同様に、先頭にSTARTを最後にENDを書きます。

命令名	定数定義　Define Constant	書　　式
		〔LABEL〕DC 定数
機能	メモリーに定数データを格納します。定数には、10進(-65535≦n≦65535)、16進(4桁)、文字定数、アドレス定数の4種類が使用できます。	
命令名	領域定義　Define Storage	書　　式
		〔LABEL〕DS n
機能	n(≧0)によって指定した語数だけ連続領域を確保します。nが0のときは領域は確保されませんが、ラベルは有効です。	

DC、DS命令でのメモリー構成

DC、DS命令には、下のように5種類の使い方があります。それぞれ以下のようなメモリー構成となります。

	ラベル	命令	オペランド
①	A	DC	100
②	B	DC	＃0064
③	C	DC	'ABC'
④	D	DC	LBL
⑤	E	DS	4

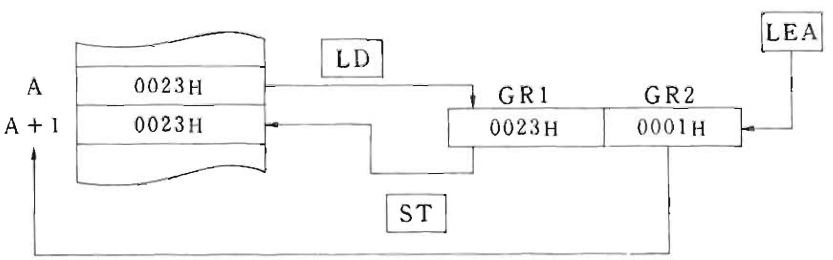


LD、ST、LEA 機械語命令:ロードストア命令、ロードアドレス命令

命令名	ロード Load	書　　式
		〔LABEL〕 LD GR, adr 〔, XR〕
機能	実効アドレスによって示された番地の内容を、レジスタ(GR) に設定します。	
命令名	ストア STore	書　　式
		〔LABEL〕 ST GR, adr 〔, XR〕
機能	レジスタ(GR)の内容を、実効アドレスによって示された番地に格納します。	
命令名	ロードアドレス Load Effective Address	書　　式
		〔LABEL〕 LEA GR, adr 〔, XR〕
機能	実効アドレスによって示される番地の内容または10進定数を、レジスタに設定します。 また、このときの値によってFRの値を設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に格納されている数値がGR1に格納されます(LD)。次に、GR2に定数1が格納されます(LEA)。最後に、A＋GR2の内容の番地、すなわちA＋1番地にGR1の内容35が格納されます(ST)。
	LD	GR1, A	
	LEA	GR2, 1	
	ST	GR1, A, GR2	
	EXIT		
A	DC	35	
	DS	1	
	END		



コラム●試験のCASL仕様書と本機のCASLの違い

本機のCASLは、試験センターのCASL仕様書に準拠して作成されていますが、実際にプログラムをシミュレートする都合のため下記の点で追加、変更しています。

項　目	試験センターのCASL仕様書	本機のCASL
アドレス空間	COMETは1語16ビットの計算機であって、アクセスできるアドレスは0番地から65535番地までです。	メモリーに制限があるため、オブジェクトで確保された範囲内となります。この範囲を超えたアクセスは、Out of addressエラーとなります。
プログラムの起動	プログラムは、オペレーティングシステムにより起動されます。プログラムが格納される番地は不定とします。	オブジェクトプログラムは、アドレス0番地から格納されます。
スタック領域	プログラムの起動時にオペレーティングシステムはスタック領域を確保し、スタック領域の最後のアドレスに1をアドレス加算した値をSPに設定します。この領域は、プログラムでスタックとして利用されます。 スタック領域は、試験問題のプログラムで使用するのに十分な容量が確保されているものとします。	スタック領域は、アドレスFFFFH番地からFF80H番地まで確保しています。ただし、使用可能領域は、FFFEH番地からFF80H番地までです。
入出力装置の割り当て	IN命令に対応する入力装置、OUT命令に対応する出力装置の割り当ては、プログラムの実行に先立ってオペレーターが行ないます。入出力装置には、コンソール表示装置、タイプライター、カード読み取り装置、カードせん孔装置などがあります。	IN命令に対応する入力装置は、キーボードと指定されています。また、OUT命令に対応する出力装置は、表示画面およびプリンタと指定されています。
DC定数	定数で指定した10進数値を1語の2進数データとして格納します。ただし、定数が−32768～32767の範囲にないときは、その下位16ビットを格納します。	定数として10進数値を用いる場合、−65535～65535が格納可能です。

ADD、SUB 機械語命令：算術演算命令

命令名	算術加算 ADD arithmetic	書　　式
		〔LABEL〕 ADD GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術加算し、その結果をGRに設定します。演算結果によって、FRを設定します。	
命令名	算術減算 SUBtract arithmetic	書　　式
		〔LABEL〕 SUB GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術減算し、その結果をGRに設定します。演算結果によって、FRを設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に格納されている16進数値＃0029がGR1に設定されます。次に、この値にB番地の内容である＃000Eを算術加算した値＃0037が、GR1に格納されます(ADD)。同様に、C番地の内容である＃001AをGR1から算術減算した値＃001Dが、GR1に格納されます(SUB)。最後に、GR1の内容をANS番地に格納します。 0029 +) 000E (ADD) 0037 −) 001A (SUB) 001D
	LD	GR1, A	
	ADD	GR1, B	
	SUB	GR1, C	
	ST	GR1, ANS	
	EXIT		
A	DC	＃0029	
B	DC	＃000E	
C	DC	＃001A	
ANS	DS	1	
	END		

AND、OR、EOR 機械語命令： 論理演算命令

命令名	論理積 AND	書　式
		〔LABEL〕 AND GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの論理積を、GRに設定します。演算結果によって、FRを設定します。	

命令名	論理和 OR	書　式
		〔LABEL〕 OR GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの論理和を、GRに設定します。演算結果によって、FRを設定します。	

命令名	排他的論理和 Exclusive OR	書　式
		〔LABEL〕 EOR GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容のビットごとの排他的論理和を、GRに設定します。演算結果によって、FRを設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1に16進定数＃5555が格納されます。次に、それとA番地の内容である＃137Fと論理積を取った結果＃1155が、GR1に格納されます。 ＃5555 = 0101 0101 0101 0101 AND) ＃137F = 0001 0011 0111 1111 ＃1155 = 0001 0001 0101 0101 (このプログラム例でのANDと同様にORとEORも使用できます)
	LD	GR1, A	
	AND	GR1, B	
	EXIT		
A	DC	＃5555	
B	DC	＃137F	
	END		

コラム●論理演算●

論理積（AND）では二つの値がともに1のときのみ1になり、論理和（OR）ではどちらか一方が1ならば1になり、排他的論理和（EOR）では二つの値が異なるときのみ1になります。これを図で表わすと右のようになります。

演算値		AND	OR	EOR
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

CPA、CPL　機械語命令：比較演算命令

命令名	算術比較 ComPare Arithmetic	書　　式
		〔LABEL〕CPA GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を算術比較し、その結果によってFRの値を設定します。	

命令名	論理比較 ComPare Logical	書　　式
		〔LABEL〕CPL GR, adr 〔, XR〕
機能	GRの内容と実効アドレスによって示される番地の内容を論理比較し、その結果によってFRの値を設定します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1に定数－32が格納されます。次に、CPA命令によりA番地の内容＃20＝$(32)_{10}$と算術比較されます。この場合は、GR1$(-32)_{10}$＜A番地の内容$(32)_{10}$なので、FRのビット値は$(10)_2$になります。 同様に、次のCPL命令でも比較が行われますが、ここでは以下に示したようなビット構成による論理比較が行なわれます。この場合は、GR1$(FFE0)_{16}$＞A番地の内容$(0020)_{16}$なので、FRのビット値は$(00)_2$になります。 GR1＝$(-32)_{10}$の ビット構成　1111 1111 1110 0000 A番地の内容＃20＝$(32)_{10}$の ビット構成　0000 0000 0010 0000
	LEA	GR1, －32	
	CPA	GR1, A	
	CPL	GR1, A	
	EXIT		
A	DC	＃0020	
	END		

SLA、SRA、SLL、SRL　機械語命令:シフト演算命令

命令名	算術左シフト Shift Left Arithmetic	書　式 〔LABEL〕 SLA GR, adr 〔, XR〕
機能	GRの内容を実効アドレスの数だけ左にシフトします。ただし、最上位ビット（符号ビット）はシフト前の値が保存され、空きビットには0が入ります。	
命令名	算術右シフト Shift Right Arithmetic	書　式 〔LABEL〕 SRA GR, adr 〔, XR〕
機能	GRの内容を実効アドレスの数だけ右にシフトします。ただし、最上位ビットはシフト前の値が保存され、その値で空きビットが埋められます。	
命令名	論理左シフト Shift Left Logical	書　式 〔LABEL〕 SLL GR, adr 〔XR〕〔, XR〕
機能	GRの内容を実効アドレスで示された数だけ左にシフトします。 シフトの結果による空きビットには0が入ります。	
命令名	論理右シフト Shift Right Logical	書　式 〔LABEL〕 SRL GR, adr 〔, XR〕
機能	GRの内容を実効アドレスで示された数だけ右にシフトします。 シフトの結果による空きビットには0が入ります。	

プログラム例

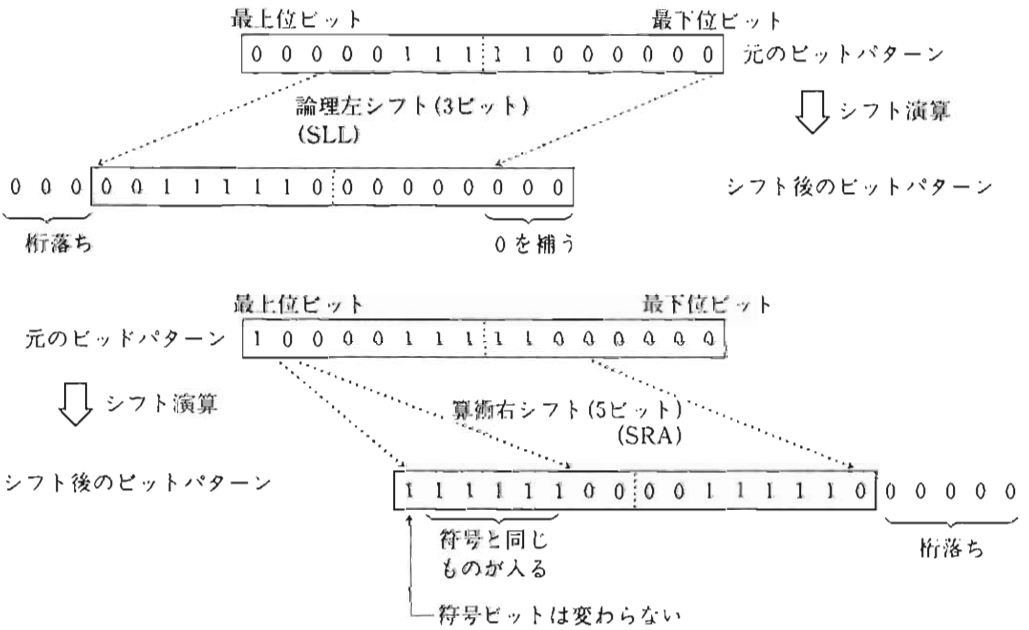
ラベル	命　令	オペランド	解　説
BGN	START		まず、GR1にA番地の内容である7FFF(16進)が格納されます。次にGR1の内容が左に5桁算術シフトされます。その結果GR1には7FE0(16進)が残ります。フラグは正ですので、$(00)_2$になります。 7FFF = 0111 1111 1111 1111 0111 1111 1110 0000 =7FE0
	LD	GR1, A	
	SLA	GR1, 5	
	EXIT		
A	DC	#7FFF	
	END		

まとめ

GR1＝1110 0001 0101 0111のとき、
SLA GR1, 5ならば、GR1＝1010 1010 1110 0000となります（左算術シフト）。
SRA GR1, 5ならば、GR1＝1111 1111 0000 1010となります（右算術シフト）。
SLL GR1, 5ならば、GR1＝0010 1010 1110 0000となります（左論理シフト）。
SRL GR1, 5ならば、GR1＝0000 0111 0000 1010となります（右論理シフト）。
算術シフトの場合は、符号ビットは変化しません。
論理シフトの場合は、符号ビットは無視されてシフトします。
シフト命令実行後の値についているアンダーラインは、実行前のGR1の値の中で保存されているものを表わしています。

コラム●シフト演算●

コンピュータの内部では、データは2進数で表現されています。その2進数を16桁なり32桁なりを集めて、データのひとまとまりとして扱っています。このひとまとまりを16ビットまたは32ビットといいます。ひとまとまりになったデータを右方向あるいは左方向に移動する演算が、シフト演算です。



命令名	正分岐 Jump on Plus or Zero	書　式 〔LABEL〕JPZ adr〔,XR〕
機能	FRのビット値が$(00)_2$「正」か$(01)_2$「零」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	負分岐 Jump on MInus	書　式 〔LABEL〕JMI adr〔,XR〕
機能	FRのビット値が$(10)_2$「負」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	非零分岐 Jump on Non Zero	書　式 〔LABEL〕JNZ adr〔,XR〕
機能	FRのビット値が$(00)_2$「正」か$(10)_2$「負」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	零分岐 Jump on ZEro	書　式 〔LABEL〕JZE adr〔,XR〕
機能	FRのビット値が$(01)_2$「零」のとき、実効アドレスに分岐します。それ以外のときは、次の命令に進みます。	
命令名	無条件分岐 unconditional JuMP	書　式 〔LABEL〕JMP adr〔,XR〕
機能	FRのビット値に関わらず、無条件に実効アドレスに分岐します。	

コラム●フラグ●

分岐命令は、フラグの値を参照してそのジャンプ先を決定します。
そのフラグの値を定める要因となる命令は、以下のとおりです。

- LEA、ADD、SUB、AND、OR、EOR、SLA、SLL、SRL命令による演算結果の符号(正$(00)_2$、零$(01)_2$、負$(10)_2$)
- CPA、CPL命令による比較の結果(GRの内容の方が実行アドバイスの内容よりも大きい$(00)_2$、同じ$(01)_2$、小さい$(10)_2$)

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		このプログラムは、複数のデータを読み込み、そのデータの中から正の最小値を探し、そのデータを保存するというものです。ただし、「3」というデータがある場合は、そのデータだけを除外して処理を行ないます。 最初に、CASLで扱える符号付きの最大値7FFFHをMIN番地に格納しておきます。DATA番地からのデータをGR0に読み込み、それが、 ・A番地の内容1より小さい（0や負のデータは切り捨てる）。 ・B番地の内容3に等しい（3は除外する）。 ・MIN番地の内容より大きいか等しい。 とき、L2にジャンプします。そうでないときは、GR0の内容をMINに格納します。L2以下では、GR1を1だけ増加して次のデータを読み込めるようにします。GR1がC番地の内容(5＝データ数）より小さいときは、L1にジャンプして再び比較を行ないます。 最後に3以外の正の最小値4がMIN番地に格納されて、プログラムが終了します。
	LD	GR0, MAX	
	ST	GR0, MIN	
	LEA	GR1, 0	
L1	LD	GR0,DATA,GR1	
	CPA	GR0, A	
	JMI	L2	
	CPA	GR0, B	
	JZE	L2	
	CPA	GR0,MIN	
	JPZ	L2	
	ST	GR0,MIN	
L2	LEA	GR1,1,GR1	
	CPA	GR1,C	
	JMI	L1	
	EXIT		
A	DC	1	
B	DC	3	
C	DC	5	
MAX	DC	#7FFF	
MIN	DS	1	
DATA	DC	5	
	DC	−1	
	DC	0	
	DC	3	
	DC	4	
	END		

PUSH、POP　機械語命令:スタック操作命令
CALL、RET　機械語命令:コールリターン命令

命令名	プッシュ PUSH effective address	書　　式 〔LABEL〕 PUSH adr 〔,XR〕
機能	スタックポインタ(SP)から1をアドレス減算した後、実効アドレスによって示された番地をSPが示す番地に格納します。	
命令名	ポップ POP up	書　　式 〔LABEL〕 POP GR
機能	スタックポインタ(SP)の示す番地の内容をGRに設定し、SPに1をアドレス加算します。	
命令名	コール CALL subroutine	書　　式 〔LABEL〕 CALL adr 〔,XR〕
機能	実効アドレスに示される番地に分岐し、処理の流れがサブルーチンに移されます。このとき、戻り番地がスタックに保存されます。	
命令名	リターン RETurn from subroutine	書　　式 〔LABEL〕 RET
機能	スタックに保存されていた戻り番地がプログラムカウンタにセットされ、サブルーチンからメインプログラムに処理の流れが戻されます。	

プログラム例

ラベル	命　令	オペランド	解　説
MAIN	START		まず、メインプログラムはサブルーチンを呼び出すだけの構成になっています。サブルーチンではPUSH、POP命令を行ない、スタックを通してGR1に100を入れます。そして、RET命令によってメインプログラムに戻ります。
	CALL	SUBR	
	EXIT		
	END		
SUBR	START		
	PUSH	100	
	POP	GR1	
	RET		
	END		

IN、OUT、EXIT　マクロ命令

命令名	IN 命令	書　　式
		〔LABEL〕IN 入力領域、入力文字長
機能	キーボードから1レコードの文字データを入力し、80語の入力領域に書いたラベル名の番地から格納します。入力文字長には、文字数が格納されます。	
命令名	OUT 命令	書　　式
		〔LABEL〕OUT 出力領域、出力文字長
機能	出力領域に格納されている文字データを、表示画面に1レコードとして出力します。ただし、出力文字長の長さまでしか出力しません。	
命令名	EXIT 命令	書　　式
		〔LABEL〕EXIT
機能	メインプログラムの実行を終了します。	

プログラム例

ラベル	命　令	オペランド	解　説
BGN	START		まず、A番地に80語の領域を取り、IN命令によりキーボードから文字列がA番地から一文字ずつ順番に格納され、B番地には文字数が格納されます。次に、OUT命令によりA番地から格納されている文字列をB番地に示す文字数だけ読み出し、画面に表示します。 最後にEXIT命令で、プログラムが終了します。
	IN	A, B	
	OUT	A, B	
	EXIT		
A	DS	80	
B	DS	1	
	END		

索引

カシオスーパーカレッジZ-1
Software Library C言語・BASIC・CASL編

発行　カシオ計算機株式会社
〒163-02 東京都新宿区西新宿2-6-1
電　話(03)3347-4811(代)

1992年12月発行

制作/編集　㈱モダン

CASIO

1992年12月現在　MD-2220204D