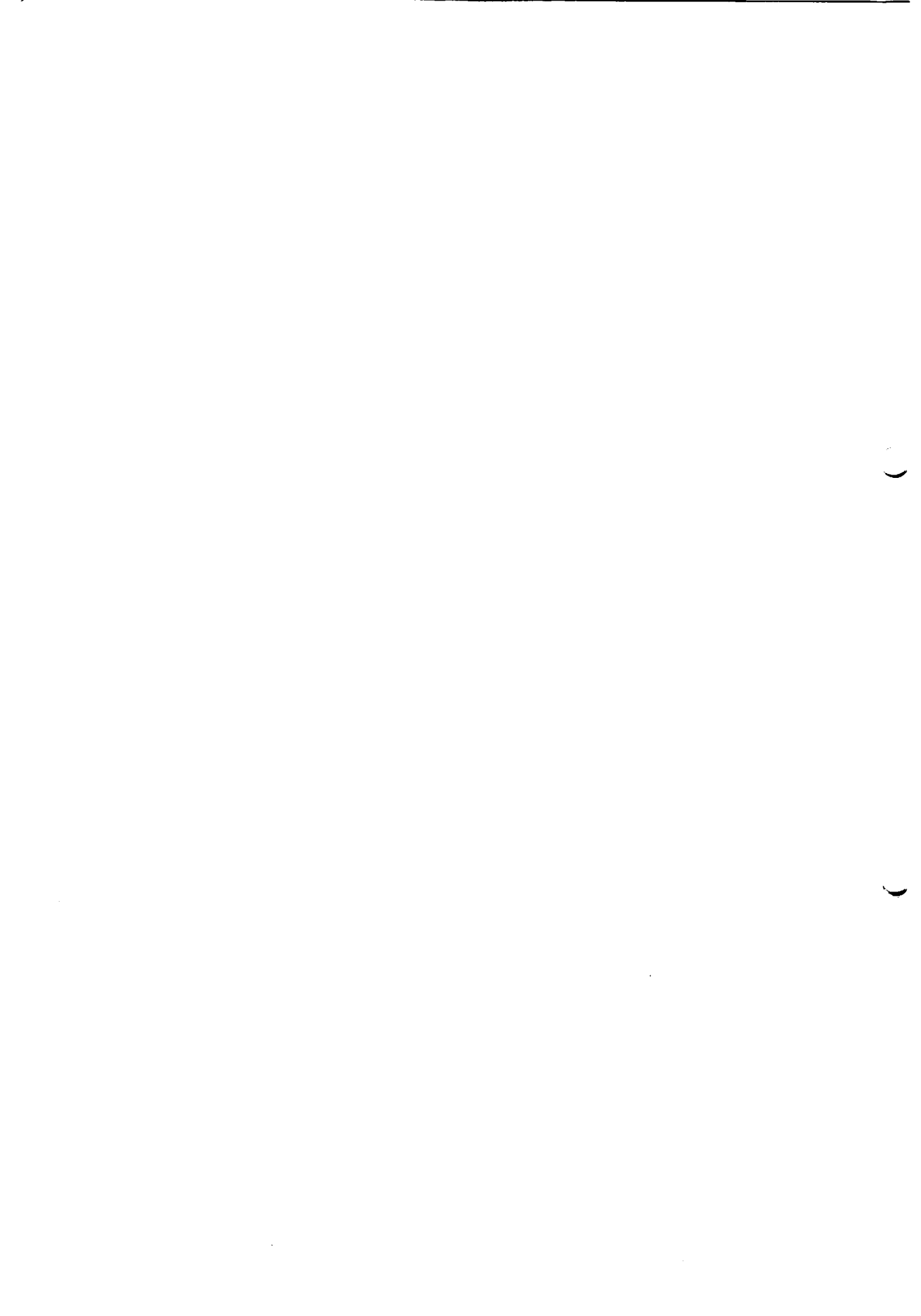


The cover features a background of fine, parallel diagonal lines. A large, stylized graphic of a circuit board or data bus is composed of multiple parallel lines that curve and zig-zag across the page. At the top, a thick black border frames the title text. The text 'MSX-C Library' is in a bold, sans-serif font, with 'Library' in white on a black rectangular background. Below it, 'USER'S MANUAL' is written in a similar bold font. In the bottom right corner, the publisher's name 'ASCII' is printed in a large, bold, sans-serif font.

MSX-C Library
USER'S MANUAL

ASCII



はじめに

この度は、MSX-C Library をお求め下さいましてありがとうございます。

MSX-C Library は MSX のグラフィック機能やマウス、プリンタなどの補助入出力装置を扱ったり、現在の MSX-C のバージョン (VER 1.1) ではサポートされていない倍精度実数や LONG 型演算をサポートするライブラリパッケージです。

MSX-C バージョン 1.0 および 1.1 では標準ライブラリしか用意されていないために、C 言語から直接、MSX の特徴であるグラフィック機能やマウス、プリンタ等を使用することはできませんでした。

MSX-C Library を使用することにより、C 言語でのプログラミングだけで MSX のほとんどの機能を使うことができ、本格的にプログラム開発をされる方にも、また、これから C 言語を学ばれる方にもご利用いただけます。

なお、本ソフトウェアには MSX-DOS、スクリーンエディタ、MSX・M-80、MSX・L-80、LIB-80、CREF-80、MSX-C は含まれておりません。MSX-C Library を使用してプログラム開発を行なう場合にはこれらのソフトウェアが必要となりますので、お持ちでない方は「MSX-DOS TOOLS」、 「MSX-C Ver.1.1」をお買い求め下さい。また、プログラム開発を行なう上でデバッグを効率的に行なうために、MSX 用シンボリックデバッガ「MSX-S BUG」が用意されていますので、お持ちでない方は是非お買い求め下さい。

本ソフトウェアをご使用になる前に、添付の「ソフトウェア使用承諾契約書」をよくお読みいただき、ご確認のうえ、添付のユーザー登録カードにご記入し、弊社までご返送下さい。ご返送をもって承諾契約書にご同意いただいたものといたします。

この「ユーザー登録カード」をご返送いただけない場合には、弊社といたしましては所定のアフターサービスをいたしかねますので、よろしくご了承のほどお願い申し上げます。

本パッケージには以下のものが含まれています。

- MSX-C Library システムディスク 1枚
(3.5-1DD フロッピーディスク)
- MSX-C Library ユーザーズマニュアル 1冊

- **MSX**、MSX-DOS、MSX・M-80、MSX・L-80、MSX-C はアスキーの商標です。
- MS-DOS は米国マイクロソフト社の商標です。
- Z80 は Zilog, Inc. の商標です。
- UNIX オペレーティングシステムは米国 AT&T のベル研究所が開発し、AT&T がライセンスしています。

ご注意

- (1)このソフトウェアならびにマニュアルを賃貸業に使用することを禁じます。また、このソフトウェアやマニュアルの一部または全部を無断でコピーすることはできません。
- (2)このソフトウェアは、製品登録カードを返送して当社に登録済みの方に限り、使用したりコピーしたりすることができます。なお、このソフトウェアを個人使用以外の目的でコピーすることはできません。
- (3)このマニュアルに記載されている事柄は、将来予告なく変更することがありますが、当社に登録されている方には案内をお送りします。
- (4)製品の内容については万全を期しておりますが、製品の内容についてのご不審や、誤り、マニュアルの記載もれなど、お気付きのことがございましたら、マニュアルの巻末の「お問い合わせ」についての要領で下記の問い合わせ先へお送り下さい。
- (5)このソフトウェアを運用した結果の影響については、(4)項にかかわらず、責任を負いかねますのでご了承下さい。

お問い合わせ先 〒107-24 東京都港区南青山6-11-1 スリーエフ南青山ビル
株式会社 アスキー ユーザーサポート係

TEL. 03-3498-0299 (祝祭日を除く月～金)
10:00～12:00, 13:00～17:00

- (6)MSX-C Ver.1.1およびMSX-C Libraryを使用して作成されたプログラムの販売については、下記にお問い合わせ下さい。

お問い合わせ先 〒107-24 東京都港区南青山5-11-5 住友南青山ビル
株式会社 アスキー システム事業部

TEL. 03-3486-8346 (祝祭日を除く月～金)
10:00～12:00, 13:00～17:00

目次

- 序章 MSX-C ライブラリパッケージの概要 3
 - 0.1 ライブラリ関数の概要 3
 - 0.2 本マニュアルの構成 4
- 第1章 本パッケージの意義とその位置づけ 5
 - 1.1 標準 C ライブラリ 5
 - 1.2 C ライブラリパッケージの位置づけ 6
- 第2章 MSX-C ライブラリパッケージの使用法 7
 - 2.1 パッケージの内容 7
 - 2.2 ライブラリパッケージの使用法 9
 - 2.3 ライブラリの保守 12
- 第3章 グラフィック関数パッケージ 17
 - 3.1 グラフィック関数パッケージの概要 17
 - 3.2 グラフィック関数パッケージの使用法 18
 - 3.3 グラフィック関数一覧 20
 - 3.4 グラフィック関数リファレンス 22
- 第4章 数値演算関数パッケージ 36
 - 4.1 数値演算関数パッケージの概要 36
 - 4.2 数値演算関数パッケージの使用法 36
 - 4.3 各数値型の詳細 37
 - 4.4 数値演算関数一覧 42
 - 4.5 数値演算関数リファレンス 45
 - 4.6 数値演算関数を使う上での注意点、留意点 56
- 第5章 その他の関数 59
 - 5.1 その他の関数の概要 59
 - 5.2 その他の関数の使用法 60
 - 5.3 その他の関数一覧 60
 - 5.4 その他の関数リファレンス 62

第6章 カーソル制御関数パッケージ 70

6.1 カーソル制御関数 (MSX-CURSES) の概要 70

6.2 MSX-CURSES 関数パッケージの使用法 74

6.3 MSX-CURSES 関数一覧 74

6.4 MSX-CURSES 関数リファレンス 77

付録 サンプルプログラム集 89

序章 MSX-Cライブラリパッケージの概要

0.1 ライブラリ関数の概要

本パッケージは MSX-C で MSX のグラフィック機能やマウス、プリンタなどの補助入出力装置を扱ったり、現在の MSX-C のバージョン (VER.1.1) ではサポートされていない倍精度実数や Long 型演算をサポートするライブラリパッケージです。

以下に本パッケージの関数を大きく 4 つに分けて、具体的な内容を示します。

グラフィックス関数パッケージ

線画(ライン)や塗りつぶし(ペイント)から、スプライト、VDP へのアクセスまで、MSX のすぐれたグラフィック機能を幅広くサポートする関数が用意されています。

数値演算関数パッケージ

数値演算を扱う関数群は大きく二つに分けることができます。まず、MSX-C ではサポートされていない LONG 型(4 バイト整数、本パッケージでは SLONG 型として扱う)での四則演算を行うための関数。そして、MSX-BASIC と同様の倍精度実数型の演算を行う関数群で構成されます。

その他の関数

その他の関数として、MSX の機能を拡張するマウスやプリンタ、ライトペンなどの補助入出力装置を扱う関数や、スロット間での関数呼び出しを行うための命令などが用意されています。

カーソル制御(MSX-CURSES)関数パッケージ

CURSES(カーサス)とは本来、UNIX というミニコンピュータ用の OS 上の、コンソール画面への文字表示やキャラクタウィンドウをサポートするためのライブラリ関数の総称です。MSX-CURSES はサブセット(一部分)ながら CURSES の仕様を MSX-DOS 上で実現するものです。

0.2 本マニュアルの構成

本マニュアルではライブラリ関数全体を上述べた4つの関数パッケージに分けて解説していきます。

以下の章では、まず本パッケージの意義と位置づけ、パッケージに含まれるファイルや、ライブラリ関数の全般的な使い方を述べた後、4つの関数群について詳細に具体的な関数の仕様とその使用法や、注意事項について解説していきます。

第1章 本パッケージの意義とその位置づけ

MSX-C コンパイラのパッケージに標準ライブラリとして付属している関数群を見渡して、「グラフィックは使えないの?」とか「音は出せないのか、BASIC でできるのに」といった疑問をもたれた方も多いかと思います。ここではそもそも標準ライブラリとは何か、本パッケージのライブラリとどんな関係にあるのかについて解説します。

1.1 標準 C ライブラリ

C 言語が UNIX というミニコンピュータ用の OS (オペレーティングシステム) 上のプログラミング言語として発展し、また UNIX システムそのものも、その大部分が C 言語によりプログラムされているということはよく知られています。UNIX システムが世界中の大学、その他の研究機関で長い年月をかけて進歩してきたのと同様に、C 言語についても UNIX の発展と共に育った言語であるために、FORTRAN や COBOL の様に、厳密な工業規格の形ではその言語仕様の定義がなされていません。そこで、UNIX の C の設計者自身がその著書「プログラミング言語 C」(リッチーとカーニハン)で示した仕様が一応の標準として多くの C の処理系がこれに準拠した形で設計されてきました (MSX-C もその一つである)。さらに、C 言語の普及にともなって、より厳密な標準規格化への試みが進められています (ANSI や JIS など)。これらの標準化案も基本的に、UNIX-C を土台としたものであることは言うまでもありません。

こうした C 言語の文法上の標準化と同時に、ライブラリについても標準ライブラリとしての標準化が進められています。標準ライブラリもまた UNIX-C のライブラリを基礎にしています。UNIX-C のライブラリは本来、その多くが UNIX システムそのものの持つ機能呼び出すいわゆるシステムコールと、C 言語とのインターフェースという性格を持ちます。もう少し分かりやすく言うと、例えば、UNIX の C コンパイラで、`read()` や `write()` などの関数を用いたプログラムをコンパイルすると、実際にディスクに対して読み書きをするコードが生成されるわけではなく、`READ`, `WRITE` というシステムコールを、それぞれに必要なパラメータをセットして呼び出すコードに展開されるにすぎないのです。

それでは、MSX-DOS 上で `read()` や `write()` のような関数を実現するにはどうしたらよいでしょう。MSX-DOS には `READ` や `WRITE` と同一仕様のシステムコールはありません。しかし、MSX-DOS にもディスクの読み書きを行うシステムコールはあるわけですから、内部的にはそれらを用いて、C 言語のプログラムからは UNIX と同様の関数呼び出しを行うようにすることは可能なわけです。すなわち、ライブラリという形で、システムとユーザプログラムとの間のインターフェースを統一することで UNIX と MSX-DOS というシステムの違いが吸収されたのです。

標準ライブラリという発想の意義はまさしくこの点にあり、ライブラリの外部的な仕様 (関数名や、引き数、戻り値など取り決め) の統一が、それらの関数を使ってプログラムしているかぎり他のシステムへの移植を非常に容易なものとしているのです (場合によっては単にコンパイルし直

すだけでよい)。

こんなわけで、C言語のライブラリである以上、標準的なCライブラリ(現在のところUNIX-Cライブラリ)の仕様はできるだけ満たすようにしなくてはなりません。また、仮にMSX-C独自の関数をライブラリとして用意するにしても、それらを標準ライブラリと混同することは、C言語本来の移植性の妨げにもなるでしょう。

MSX-C ver1.1のマニュアルにおいてもUNIX-Cとのライブラリ関数の仕様上の違いを特にあげているのは、MSX-Cを使いこなすためにはもちろん、ユーザにそのことをとくに意識してプログラムをしていただくためでもあったのです。

1.2 Cライブラリパッケージの位置づけ

しかし、MSXのユーザにしてみれば、UNIXやMS-DOSなどへの移植性もさることながら、MSXのもつすぐれたグラフィック機能や、ジョイスティックやマウスなどの特殊装置を十分に活かしたプログラムを作りたいと考えるのも当然の要請でしょう。本パッケージはこうした要請から、Cコンパイラのパッケージとは別個に(標準ということを問題にしない)MSX独自の数々の機能をC言語により用いるためのライブラリパッケージなのです。

本パッケージをこのように位置づけると、MSX-BASICを手本として関数の仕様を決めるといったことが考えられます。MSX-BASICはおよそMSXの機能をフルにサポートしているのでMSX専用のライブラリをつくるには大いに参考となるわけです。ただ、Cの関数をBASICの命令とまったく同じ仕様にしたのでは、C本来の柔軟性やエラーチェックはユーザ自身がプログラムで行い、関数の機能はできるだけプリミティブなものにするという思想からも外れてしまいます(極端な話間違った関数の使い方をするとsyntax errorなどと表示されたのではたまりません)。

そこで、本パッケージは、MSX-BASICを大いに意識して広くMSXの機能をサポートしながらも、個々の関数の機能を単純にして、エラーチェックなどはユーザの裁量に任せるようになっています。従って、ある意味ではBASICのように単純には行かないのですが、ユーザがより自由に「Cらしい」プログラムを書くことができるでしょう。

また、第6章で詳しく述べていますが、本パッケージにはMSX-CURSESという関数群が含まれています。CURSES(カーサス)とはUNIX上の「画面更新とカーソル動作の最適化のためのライブラリパッケージ」、すなわちテキスト画面を扱うアプリケーションのための関数群の総称です。この説明ではちょっとピンとこないかも知れませんが、UNIX上のスクリーンエディタviや、ロールプレイングゲームの元祖とも言うべきRogue(ログ)などがCURSESを用いたアプリケーションとして知られています。

MSX-CURSESはその名のとおりにUNIXのCURSESのサブセット(一部分)となっています(一部関数の仕様が異なるところもある)。この点ではMSX-CURSESの関数群については、グラフィックなどを扱う他の関数とは区別する必要があるかもしれません。

第2章 MSX-Cライブラリパッケージの使用法

2.1 パッケージの内容

本パッケージに付属のディスクに含まれるファイルの一覧を示します。

オブジェクトファイル

mllib.rel	MSX-Cライブラリのオブジェクトファイル
glib.rel	以下、MLIB.RELを4つに関数パッケージに分割したオブジェクト
math.rel	
msxbios.rel	
curses.rel	

.TCOファイル

mllib.tco	本ライブラリ使用のプログラムのパラメータチェック用の.tcoファイル (標準ライブラリのlib.tcoに匹敵するもの)
-----------	--

ヘッダーファイル

glib.h	グラフィック関数群の宣言やいくつかの型定義など
math.h	数値演算関数群の宣言や、データの型定義など
msxbios.h	MSXのBIOSを呼び出している関数の宣言など
curses.h	MSX-CURSESの関数群の宣言やデータ構造体の型定義など

ユーザプログラム作成用バッチファイル

msxc.bat	本ライブラリを用いてプログラムを作成するためのバッチ MSX-Cのパッケージに付属するc.batにあたるもの
----------	---

ライブラリソースファイル

glib.mac
glibc.c
mathmac.mac
mathc.c

prsc.c
msxbios.mac
msxbiosc.c
cursesc.c
curses2.c

ライブラリ保守用のファイル

genall.bat
genglib.bat
genmath.bat
genbios.bat
gencurs.bat
genmlib.bat
gentco.bat
gen.bat
glib.tco
math.tco
msxbios.tco
curses.tco

以下、curses.tco までの .tco ファイルはライブラリに変更を加えた際に mlib.tco を再作成するためのもので、実際のパラメータチェックには mlib.tco のみで足りる。

ユーティリティープログラム

echo.com ライブラリ再作成の際にもちいるプログラム
mx.com 同上

サンプルプログラムファイル

den.c
den.com
show.c
show.com
gcal.c
gcal.com

その他

mksys.bat 本パッケージを用いたプログラム開発環境ディスクをつくるためのバッチ

forremk.bat ライブラリ再作成用のシステムファイルをつくるバッチ
readme.doc 最新の更新事項の情報

2.2 ライブラリパッケージの使用法

本パッケージに付属のディスクには MSX-DOS のシステムや、MSX-C コンパイラそのもの、標準ライブラリなどプログラムの開発に必要なファイルが含まれていません。

そこで、まずはじめにすることは、いままでお使いの MSX-C コンパイラのシステムディスクに本パッケージ中から必要なファイルをコピーするなどして、プログラムの開発環境としてのシステムディスクを作ることです。

通常の本ライブラリを用いてのプログラム開発では、いままでにお使いの MSX-C コンパイラのシステムディスクに、本パッケージ付属のディスクから以下のファイルをコピーすれば足りるでしょう。

バッチファイル

msxc.bat

ライブラリファイル

mllib.rel

パラメータチェック用 TCO ファイル

mllib.tco

ヘッダーファイル

glib.h

msxbios.h

math.h


curses.h

また、本パッケージには、新しくおろしたてのディスクから MSX-DOS-TOOLS や MSX-C コンパイラ ver.1.1 のディスクから必要なファイルを次々とコピーし、システムディスクをつくるためのバッチ、mksys.bat も用意されています。このバッチを用いてシステムディスクを作成するには本パッケージのディスク、新しくフォーマットしたディスクの他に MSX-DOS-TOOLS や MSX-C コンパイラ ver.1.1 のディスクが必要です。


mksys.bat の起動法

まず、新しいディスクに mksys.bat をコピーして下さい。次にそのディスクをカレントドライ

ブに挿入し、MSX-DOS上で次のように入力してください。ディスクのライトプロテクトははずしておいて下さい。

A>mksys <ドライブ名>  (ドライブ名の指定にコロンはいりません)

ドライブ名は省略できません。ドライブが1台の場合は2ドライブシミュレータを用いて下さい。ドライブ名の指定にコロン(':')は必要ありません。実際の mksys.bat の起動は次のようになります。

A>mksys h 

これでバッチが起動され、TOOLS や MSX-C ver.1.1 のディスクの挿入を要求するメッセージが表示されますから、それにしたがってディスクの入れ替えをしてやれば、新しいディスクに次々に必要なファイルがコピーされシステムディスクができあがります。

2.2.1 ソースファイルの作成(ヘッダーファイルのインクルード)

Cでは、関数をプログラムで用いる場合、用いる関数を予め宣言しておくのが原則です。標準ライブラリの場合は stdio.h などのヘッダーファイルと呼ばれる特殊なファイルをインクルードすることによりユーザプログラムのソース内でいちいち個々の関数を宣言しなくて済みました。

本パッケージでも同じように、関数の宣言やデータの型宣言などをおこなっているファイルを用意しています。これらを必要に応じてインクルードすることによりユーザはそのプログラム内で自由にライブラリ関数や定義された定数などを使うことができます。

インクルード用のヘッダーファイルには下のようなものがあります。

glib.h
math.h
msxbios.h
curses.h

インクルードすべきヘッダーファイルは用いる関数によって異なります。具体的には次章以下の各関数群の使用法をご覧ください。

2.2.2 コンパイル

ソースファイルが書けたら通常どおりにコンパイルしますが、この際 MSX-C のパッケージに含まれている c.bat などの、FPC によりパラメータチェックを行っているバッチ処理では確実にエラーが起こります。なぜなら c.bat で FPC のチェックの対象としている lib.tco では本パッケージの関数はまったく定義されていないからです。

そこで、本パッケージの全ての関数のプロトタイプ宣言(関数の骨組みだけを定義している)を含む TCO ファイル, `mlib.tco` を併せて FPC にわたすことによりパラメータチェックを正常に行うことができます。

パラメータチェックの書式は以下の様になるでしょう。

```
fpc lib mlib prog
```

この例ではユーザプログラムファイル `prog.c` を CF にかけて生成した `prog.tco` のパラメータチェックを行っています。

もうひとつ CF について述べておかななくてはならないことがあります。ソースファイルでインクルードするヘッダーファイルのなかでは多くの関数や定数, 変数が定義されています。CF 実行時には、これらのグローバルに定義された関数名や変数名などはソースのなかでの参照の有無に関係なく、そのためのシンボルテーブルが作られます。このため、`stdio.h` のみをインクルードしていた場合に比して、CF 実行時にシンボルテーブルオーバーフローのエラーがおきやすくなります。この場合 CF のオプションによりシンボルテーブルの割合を少し増やしてやることによりほとんどエラーなく終了することができます。

後述のバッチ `msxc.bat` でも CF 実行時のオプションの指定が可能になっています。

2.2.3 リンク

コンパイルが済んだらライブラリの本体である REL(`mlib.rel`)ファイルも含めてリンクを行います。本パッケージのライブラリファイルは、ただ1つの REL ファイルにまとめられていますが、正式なライブラリ化により作成されていますのでリンカ(MSX-L80)の `/S` オプションにより必要なモジュールだけを抜き出して結合することができます。

ライブラリファイル名は `mlib.rel` です。本ライブラリ関数を用いたプログラムのリンクの際には必ずこれをリンクする必要があります。

リンクの際には、`mlib.rel` は、標準ライブラリファイル `clib.rel` やランタイムライブラリファイル `crun.rel`, `cend.rel` よりも前にリンクしてください。本パッケージのライブラリ関数のなかには標準ライブラリを内部で呼び出しているものや、標準ライブラリ関数と名前が重複するものがあるからです。

リンクの書式例を示します。

正しい例

```
l80 ck,prog,mlib/s,clib/s,crun/s,cend,prog/n/y/e:xmain
```

誤った例

```
l80 ck,prog,clib/s,mlib/s,crun/s,cend,prog/n/y/e:xmain
```

(この場合未定義エラーがでたり、同じ名前の標準ライブラリをリンクしてしまう場合がある)

以上、ユーザプログラムの作成手順をおおまかに追ってみたわけですが、ここまでの一連の処理をバッチファイルにしたものが `msxc.bat` として提供されています。

`msxc.bat` の内容は以下のとおりです。バッチ起動時のパラメータの 2 番目, 3 番目がそれぞれ, CF, CG のオプション指定に用いられるのがわかるでしょう。

```
cf %2 %1
fpc %1 mlib lib
cg -k %3 %1
m80 =%1/z
del %1.mac
180 ck,%1,mllib/s,clib/s,crun/s,cend,%1/n/y/e:xmain
```

また, `mllib.rel` は本パッケージの全ての関数を含んだファイルでしたが, これを 4 つの関数群にわけたファイルも用意されています。

<code>glib.rel</code>	グラフィック関数パッケージ
<code>math.rel</code>	数値演算関数パッケージ
<code>msxbios.rel</code>	マウスやプリンタなど基本的な出力関数パッケージ(本マニュアルではその他の関数として分類されている)
<code>curses.rel</code>	MSX-CURSES パッケージ

たとえば Long 型の演算関数のみを使う場合には, `mllib.rel` にかえて `math.rel` をリンクすることにより多少リンクに要する時間が短縮できることがあります。

ただし, 個々のライブラリ関数の中には, 内部的に, 他の関数群に属する関数を呼び出しているものがいくつかあります。この場合, リンカは呼び出されている関数の実体を与えられた REL ファイルから見つけることができず, 未定義エラーとなります。こうした際には `mllib.rel` でリンクしなおして下さい。

2.3 ライブラリの保守


ユーザが本ライブラリを使い込むうちに関数の細かい仕様を改良したり, 新たに自分で作成したサブルーチンをライブラリに加えたいと思うことがあるでしょう。本パッケージにはライブラリの本体である REL ファイルを再作成するために必要な全てのソースファイルを含んでいますので, 各ユーザがソースレベルで改編して自由に自分なりのライブラリパッケージを完成していくことができます。

2.3.1 ソースファイルの作成

ライブラリのソースを直接編集して新しいライブラリファイルの作成までを行うわけですから、作業の前にもとのライブラリのバックアップを作っておくことをおすすめします。

バックアップの取り方

MSX-Cライブラリパッケージのディスクをドライブ A に入れ、フォーマット済みのディスクをドライブ B に入れ(2ドライブシミュレータの場合は入れずに)、次のようなコマンドを実行します。

```
A>copy a:*.* b: 
```

(実際の表示省略)

2ドライブシミュレータの場合は MSX-DOS からのメッセージに従ってディスクを交換して下さい。MSX-DOS のプロンプトの A>が出たらバックアップは完了しました。ソースファイルの編集などはこのバックアップされたディスクでおこなって下さい。

次にエディタを起動してソースファイルを編集します。

ここで注意すべきなのはソースファイル中の関数のおかれる順番です。最終的な目標である glib.rel.mlib.rel などは、リンク時に必要な関数だけを抜き出すことができる本格的なライブラリですから、この場合、ライブラリ関数内で別のライブラリ関数の呼び出しを行っているときには、呼ばれる側の関数はソース中では必ず呼ぶ側よりも前で定義されて(書かれて)いなくてはなりません(この理由については MSX-C 付属のマニュアルのライブラリの作成の項をご覧ください)。

こうして必要な変更を加え、できあがったファイルをディスクにセーブすればソースの作成は完了です。

2.3.2 ヘッダファイルの変更

次は変更を加えたソースに対応するヘッダファイルに新しく加えた関数の宣言をします。こうすることで新しく加えた関数を、ヘッダファイルをインクルードするだけで使うことができます。

2.3.3 コンパイル、アセンブルとバッチファイル

ヘッダファイルの変更までできたら、ソースをコンパイル、アセンブルします。これには一気にライブラリ化するためのバッチファイルが用意されています。これを走らせることによりすべての REL ファイルと TCO ファイルを生成します。

ここで使用するバッチファイルは genall.bat です。

A) バッチファイルを起動する前に

ライブラリファイルが無事に作成されるまでには多くのファイルがディスク上に作られたり消されたり、あるいはリネームされたりします。ディスクの容量がいっぱいになりバッチが途中で止まってしまったり、大事なファイルが消されてしまったりなどのトラブルを防ぐためにライブラリの再作成には専用に1枚ディスクを用意されることをおすすめします。ただし、ディスクドライブが1台のみの方の場合、2ドライブシミュレータにより2枚のディスクを用いることができますが、処理時間と手間がたいへん大きいので1枚のディスクで行なう方が効率的かも知れません。

■ 2枚のディスクを用いる場合(2ドライブシミュレータの場合も含む)

1枚目のディスクには下のようなファイルが必要です。これをライブラリ作成用のシステムディスクと呼びます。

システムディスクはふだんお使いのMSX-C用のシステムディスクに足りないファイルを本パッケージのディスクからコピーすれば良いでしょう。必要なファイルをコピーしたらライトプロテクトをオン(書き込み禁止状態)にしておいて下さい。

システムディスクに必要なファイル

msxdos.sys

command.com

cf.com コンパイラ本体、パーサ

cg.com コンパイラ本体、コードジェネレータ

m80.com アセンブラ

lib80.com ライブラリマネージャ

fpc.com パラメータチェッカ

mx.com ライブラリ保守支援ツール、本パッケージに付属のもの

echo.com 本パッケージに付属のもの

genall.bat 起動に使うバッチ

genmath.bat

gencurs.bat

genglib.bat

genbios.bat

genmlib.bat

gentco.bat

gen.bat

最後に起動されるバッチ(.lib から .rel へのリネームを行なっている)

arel.bat

TEMP.BATの生成に必要なファイル(アセンブラ処理のプロトタイプが定義されている)

crel.bat

同上(Cプログラムの処理のプロトタイプ)

stdio.h	標準入出力ヘッダーファイル
bdosfunc.h	MSX-C コンパイラ 1.1 に付属のもの
glib.h	ヘッダーファイル
math.h	ヘッダーファイル
msxbios.h	ヘッダーファイル
curses.h	ヘッダーファイル
msxbiosc.c	
msxbios.mac	
glibc.c	
glib.mac	関数を加えたり変更したソースファイル
mathc.c	
mathmac.mac	
prsc.c	
cursesc.c	
curses2.c	

以上 34 ファイル

もう一枚のディスク(これをここではワーク用のディスクと呼ぶ)は新しくおろしたディスクをフォーマットしてディスクのライトプロテクトはそのままオフ(書き込みができる状態)にしておいてください。ワークディスクはフォーマットされたディスクであれば問題ありません。

■ 1 枚のディスクを用いる場合

この場合は、システムディスクとワークディスクが兼用となります。この場合はふだんお使いの MSX-C のシステムディスクをそのままお使いになるのではなく、新しいディスクに上記の必要なファイルだけをコピーし、これをライブラリ作成用のシステム兼ワークディスクとしてお使い下さい。そうしないとバッチの途中でファイルを作ることができなくなる場合があります、ライブラリが再作成できません。

B) GENALL.BAT の起動


必要なファイルが揃っていることを確認したら genall.bat を起動してみましょう。

■ 2 枚のディスクを用いる場合

まずカレントドライブ(プロンプトが "A:") なら A ドライブ)に上のシステムディスクをいれ、もう一方のドライブには新しくおろしたワークディスクを入れて下さい。このときカレントドライブのディスクだけがライトプロテクトされた状態です。


ここではライブラリをすべて再作成しますから genall.bat に渡す引き数はファイルを作成するドライブ名です。カレントドライブが A ドライブの場合、次のようにコマンド入力します。こ

これは2ドライブシミュレータにより2枚のディスクを使う場合はまずカレントドライブのディスクをいれて同じコマンド入力となります。(ただし実際のオペレーションでは途中でディスク入れ替えのメッセージが表示され、それにしたがってディスクを入れ換えなくてはならない)

A>genall b  (ドライブ名の指定にコロンはいりません)
(実際の表示省略)

■ 1枚のディスクを用いる場合

必要なファイルがすべて入ったディスク(システム兼ワークディスク)をカレントドライブに入れます。ライトプロテクトはオフです。この場合、ライブラリファイルを作成するディスクはカレントドライブのディスクということですから、GENALLの1番目の引き数はカレントドライブ名となります。

A>genall a 
(実際の表示省略)

2.3.4 その他, 注意事項

エラーがなく最後のバッチ gen.bat まで実行が終わったら、新しい関数を含んだライブラリの出来上りです。ふだんお使いの MSX-C のディスクにコピーして、これまでの MSX-C ライブラリと同じようにコンパイル、リンクができます。

これまで書かれたようにすることで、ユーザにあった MSX-C ライブラリにすることができますが、genall バッチを1回実行するのに3時間から4時間ほどかかってしまいます。ですから追加する関数は十分デバッグしてからライブラリに加えることをおすすめします。また、バッチファイルを解析することでより短時間にライブラリを更新することができるのが解ると思います。

第3章 グラフィック関数パッケージ

3.1 グラフィック関数パッケージの概要

本パッケージ中には、グラフィック画面を扱うために数多くの関数が用意されています。これをさらに下のような7つの関数群に分類しそれぞれについて解説を行います。

- ・初期化及び VRAM のアクセスに関する関数
- ・VDP のレジスタを操作する関数
- ・色を操作する関数
- ・描画
- ・スプライト
- ・ビットブロック転送
- ・その他の関数

3.1.1 初期化及び VRAM のアクセスに関する関数

グラフィック関数を使用する前に必ず呼ばなければならない関数と、VRAM をアクセスするのに必要な関数を集めてあります。

3.1.2 VDP のレジスタを操作する関数

VDP(9918 や 9938)の内部レジスタを読み書きする関数群です。

3.1.3 色を操作する関数

フォアグラウンド、バックグラウンド、ボーダーの色を設定したり、パレットを操作する関数群です。

3.1.4 描画ルーチン

グラフィック画面に描くための関数群。BASIC の LINE や CIRCLE などの図形を描くための命令に相当するルーチンや、PAINT(塗りつぶし)や PSET(点描)などを描画ルーチンとしてこの部類にいれることができるでしょう。基本的には、これらの関数は BASIC 的な感覚で使えますが、ロジカルオペレーション指定が可能かどうかや、カラーの有効範囲などがスクリーンモードによって異なるため、注意が必要です。

3.1.5 スプライトに関するルーチン

スプライトも画面表示には違いないのですが、単純に点や線を描くのとは違いスプライトの大きさやキャラクタのパターンを設定したりなどの前準備が必要です。本パッケージには、スプライトパターンやカラーの設定から実際の表示までに必要な全てのルーチンが揃っています。

3.1.6 ビットブロック転送ルーチン

ビットブロック転送というちょっと聞きなれない言葉ですが、BASICのCOPY命令に相当する一連の関数と考えれば良いでしょう。

これらのルーチンを用いることによって、デジタイズやグラフィックソフトなどで作成した画像データをCのプログラムで自由に転送し、画面の任意の位置に表示することができます。

C言語が人工知能などのプログラミングにも適していると言われていまして、画像データさえうまく作れば、ちょっとしたアドベンチャーゲームなどをつくることも可能でしょう。

下位レベルの関数(VRAMアクセスやVDPレジスタの操作など)を除けばグラフィックを扱うほとんどの関数が、機能の面から考えるとBASICで馴染みのものばかりですから、例えば、「スプライトとは何か」とか「スクリーンモードによる具体的な違いは何か」などについてはMSX-BASICのマニュアルや、「MSX2 テクニカルハンドブック」などの解説をご覧ください。

但し、Cでは引き数の省略ができませんし、また細かいエラーチェックも行っておりませんので、BASICと比較して、引き数や戻り値に十分注意した、よりきめの細かいプログラムを心がけねばなりません。

3.2 グラフィック関数パッケージの使用法

3.2.1 ソースファイルの作成

ライブラリ関数群は試してみれば外部定義された(同一ソース以外でプログラムされた)関数の集まりですから、これらの関数を呼び出すさいには# include プリプロセッサ文により予め関数の宣言が行われているヘッダーファイルを読み込まなくてはなりません。

stdio.hのインクルードの後でglib.hをインクルードすることにより本章で解説するグラフィック関数をユーザは特に宣言することなく用いることができます。ただしglib.hのなかですべてのグラフィック関数が宣言されているわけではなく、glib.hのなかでさらにmsxbios.hというヘッダーファイルの呼び出しを行っていて、この2つのヘッダーファイルをあわせてはじめてグラフィック関数の宣言が網羅されることになります。

本章で解説するグラフィック関数を用いるプログラムのソースの先頭部分は例えば次のように

なるでしょう。

```
#pragma nonrec

#include <stdio.h>           ◯ヘッダーのインクルード
#include <glib.h>
.
.
.
func1(fore, back, bord)
TINY fore, back, bord;
{
    screen((TINY)8);        ◯実際にグラフィック関数を用いている部分
    color(fore, back , bord)
    .
    .
}
```

glib.h のなかでは関数の宣言だけでなくいくつかの型や定数の定義などもおこなわれています。glib.h で定義された型などについてはこれらを用いる必要のある個々の関数の解説中でそのつど解説します。

3.2.2 コンパイル, リンク

前節で述べたように glib.h のなかで msxbios.h がインクルードされていますので、cf(コンパイラのフロントエンド)をかける際には、カレントドライブのディスクには glib.h のほかに msxbios.h が必要です。

第2章で述べたように、本パッケージでは、パラメータチェック用の TCO ファイルやライブラリファイル(REL ファイル)は関数群毎に分けず、それぞれ1ファイルにまとめているので、パラメータチェックについては

```
m lib.tco
```

リンクの際には

```
m lib.rel
```

を fpc, 180 のパラメータとして与えます。詳しくは 2.2 をご覧下さい。

3.3 グラフィック関数一覧

初期化及び VRAM のアクセスに関する関数

関数名	用途	参照ページ
ginit	グラフィック関数を呼ぶ前に必ず呼ぶ関数	23
interlace	インターレースモードの設定	23
setrd	VDP を読み込みモードに設定する	24
invdp *	VRAM から 1 バイトデータを読み込む	24
setwrt	VDP を書き込みモードに設定する	24
outvdp *	VRAM に 1 バイトのデータを書き込む	24
vpeek	VRAM の指定したアドレスの内容を返す	24
vpoke	VRAM に 1 バイトのデータを書き込む	24
filvrm	VRAM の指定領域を指定の 1 バイトのデータでクリアする	24
ldirmv	VRAM からメインメモリへのデータ転送	25
ldirvm	メインメモリから VRAM へのデータ転送	25

VDP のレジスタを操作する関数

関数名	用途	参照ページ
wrtvdp	VDP のコントロールレジスタにデータを書き込む	25
rdvdp *	VDP のコントロールレジスタの内容を読み込む	25
rdvsts	VDP のステータスレジスタの値を返す	26

色を操作する関数

関数名	用途	参照ページ
color	スクリーンのフォアグラウンド、バックグラウンド、ボーダーの色設定	26
iniplt	パレット及び VRAM に保存されたパレットデータの初期化	26
rstplt	VRAM からパレットデータをリストアする	26
getplt	パレットの内容を返す	27
setplt	パレットのセット	27

注意 * の付いた関数は割り込みフラグを変化させません。それ以外の関数は「割り込み許可」で返ってきます。

描画関数

関数名	用途	参照ページ
pset	グラフィック画面に指定色で点を描く	27
line	2点の指定座標を結ぶ直線を描く	28
boxline	2点の指定座標を対角とする四角形を描く	28
boxfill	2点の指定座標を対角とする四角形を塗りつぶす	29
circle	指定座標を中心とする円を描く	29
paint	グラフィック画面の指定された境界色で囲まれた部分を指定の色で塗りつぶす	29
point	グラフィック画面の指定の座標の色コードを返す	30

スプライト関係の関数

関数名	用途	参照ページ
inispr	全てのスプライトの初期化	30
calpat *	指定されたスプライトパターンに対応するスプライトパターンテーブルの先頭アドレスを返す	31
calatr *	指定したスプライト面に対応するスプライトアトリビュートテーブルの先頭アドレスを返す	31
sprite	スプライトパターンの設定	31
colspr	スプライトのライン毎の色を指定する	31
putspr	スプライトの表示	32

ビットブロック転送

関数名	用途	参照ページ
cpyv2v	VRAMの指定ページを他のページへ転送する	32
cpyv2m	VRAMの指定ページをメインRAMに転送する	33
cpym2v	メインRAMの画像データをVRAMへ転送する	33

その他のグラフィック関係の関数

関数名	用途	参照ページ
totext	スクリーンモードをテキストモードに戻す	33
grpprt	現在のグラフィックカーソル位置に一文字表示する	34
knjprt	現在のグラフィックカーソル位置に JIS 漢字コードに対応する文字を表示する	34
glocate *	カーソル移動	34
setpg	VRAMの表示ページと書き込みページの設定	35
vramsize *	VRAMのサイズを返す関数	35

3.4 グラフィック関数リファレンス

本節では個々の関数の仕様と、その用法の解説をおこないます。関数の解説において、なじみのうすいいくつかの変数型が用いられていますが、これはヘッダーファイルのなかでつぎのように定義されています。

TINY 符号なし 1 バイト整数型。すなわち、char に定義 (typedef) されている。
NAT 符号なし 2 バイト整数型。unsigned に定義されている。

3.4.1 グラフィックに関するシステム変数

グラフィック関数群のために以下にあげるようなシステム変数が用意されています。これらは実際には glib.h の中で定義されている #define マクロであり、MSX のワークエリアを直接参照します。

TINY c_dpage ;
TINY c_apage ;

それぞれ現在の表示ページ、アクティブページを保持している変数です。参照は自由にできますが、直接代入できるのは c_apage だけです。c_dpage を変更する際は必ず setpg() 関数を使用してください。c_dpage に直接代入した場合の動作は保証されません。

TINY c_fore ;
TINY c_back ;
TINY c_bord ;

それぞれ現在のフォアグラウンドカラー、バックグラウンドカラー、ボーダーカラーを保持している変数です。参照は自由にできますが、直接代入できるのはグラフィックモードにおける c_fore もしくは c_back だけです。テキストモードにおいてもしくは c_bord を変更する際は必ず color() 関数を使用して代入してください。直接代入した場合の動作は保証されません。

NAT c_lastx ;
NAT c_lasty ;

それぞれ一番最後に指定された X 座標、Y 座標を保持している変数です。参照、代入とも自由にできます。またこれらは glocate() 関数を使用して設定することもできます。

主に連続した直線を引くのに、line() 関数と共に使用されます。

```
TINY    c_screen ;
TINY    c_sprite ;
```

それぞれ現在のスクリーンモード、スプライトサイズを保持している変数です。参照は自由にできますが、代入するには必ず `screen()`、`inispr()` 関数を使用してください。

```
NAT     c_xmax ;
NAT     c_ymax ;
```

それぞれ現在のスクリーンモードで表示できる X 座標、Y 座標の最大値を保持しています。参照は自由にできますが、代入はできません。実際にはそれぞれ、`gtxmax()`、`gtymax()` に `define` されています。

3.4.2 グラフィックに関する定数

グラフィック関数群のために以下にあげるような定数が用意されています。これらは `glib.h` の中で定義されています。

ロジカルオペレーション

```
PSET      0x00
AND       0x01
OR        0x02
XOR       0x03
PRESET    0x04
TPSET     0x08
TAND      0x09
TOR       0x0a
TXOR      0x0b
TPRESET   0x0c
```

3.4.3 初期化及び VRAM のアクセスに関する関数

```
VOID     ginit()
```

グラフィック関数を呼ぶ前に必ずこの関数を呼び出してください。

VOID interlace(mode)
TINY mode ;

インターレースモードを設定します。インターレースモードについては MSX-BASIC のマニュアルを参照してください。

VOID setrd(adrs)
NAT adrs ;

VDP を adrs 番地からの読み込みモードに設定します。次の invdp() と共に使用されます。

TINY invdp()

VRAM から 1 バイトデータを読み込みます。

VOID setwrt(adrs)
NAT adrs ;

VDP を adrs 番地からの書き込みモードに設定します。次の outvdp() と共に使用されます。

VOID outvdp(data)
TINY data ;

VRAM に 1 バイトデータを書き込みます。

TINY vpeek(vadrs)
NAT vadrs ;

VRAM の vadrs 番地の内容を返します。

VOID vpoke(vadrs,data)
NAT vadrs ;
TINY data ;

VRAM の vadrs 番地に data の 1 バイトを書き込みます。

VOID filvrm(vadrs,len,data)
NAT vadrs ;

```
NAT    len ;
TINY   data ;
```

VRAM の `vadr`s で指定したアドレスからの `len` バイトを `data` の値でうめます。

```
VOID    ldirmv(dst, src, len)
TINY    *dst ;
NAT     src ;
NAT     len ;
```

VRAM からメインメモリへのデータ転送を行います。

`dst` に転送先のメイン RAM のアドレス、`src` は転送元の VRAM のアドレス、`len` には転送するデータ長をバイト単位で与えます。

```
VOID    ldirvm(dst, src, len)
NAT     dst ;
TINY    *src ;
NAT     len ;
```

メインメモリから VRAM へデータを転送します。

`dst` に転送先の VRAM のアドレス、`src` に転送元のメイン RAM のアドレス、`len` には転送するデータ長をバイト単位で与えます。

3.4.4 VDP のレジスタを操作する関数

```
VOID    wrtvdv(vreg, data)
TINY    vreg ;
TINY    data ;
```

VDP のコントロールレジスタにデータを書き込みます。

`vreg` でレジスタ番号を指定し、`data` で書き込む値を渡します。`vreg` は VDP のレジスタ番号であり、MSX-BASIC の VDP コマンドとは 9 以上の場合 1 つずれていることに注意してください。つまり MSX-BASIC でのレジスタ 9 はこの関数ではレジスタ 8 になります。

```
TINY    rdvdv(vreg)
TINY    vreg ;
```

`vreg` で指定される VDP のコントロールレジスタの内容を読み込みます。`vreg` は VDP のレジ

スタ番号であり、MSX-BASICのVDPコマンドとは9以上の場合1つずれていることに注意してください。つまりMSX-BASICでのレジスタ10はこの関数ではレジスタ9になります。

この関数はwrtvdp()を使用してVDPのコントロールレジスタに書き込んだ場合のみ有効です。I/O命令で直接VDPのコントロールレジスタに書き込んだ時、この関数の動作は保証されません。

またvregの範囲は0から23までです。24以上の値を指定したときの動作は保証されません。

TINY rdrvsts(sreg)

TINY sreg ;

sregで指定したVDPのステータスレジスタの値を返します。MSX-BASICのVDP関数とはレジスタの指定方法が異なることに注意してください。MSX-BASICのVDP(8)はrdvsts(0)に、VDP(-1)はrdvsts(1)に相当します。

3.4.5 色を操作する関数

VOID color(fores,back,bords)

TINY fores,back,bords ;

スクリーンのフォアグラウンド、バックグラウンド、ボーダーの色を設定します。

戻り値はありません。無効な色(スクリーン2上で256を指定したなど)に対しての動作は保証されません。

使用例

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

```
#include <glib.h>
```

```
/* フォアカラーをバックに、バックをボーダーに、ボーダーをフォアに変更する */
```

```
VOID func1()
```

```
{
    color(c_bord, c_fore, c_back);
}
```

VOID iniplt()

パレット及びVRAMに保存されたパレットデータを初期化します(デフォルト値に戻す)。

VOID rstplt()

VRAM からパレットデータをリストアします。

NAT getplt(pal)

TINY pal;

pal で指定されたパレットの内容を返します。戻り値として NAT 型の値 (setplt 参照) を返します。

使用例

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

/* 指定されたパレットの緑のレベルを最高にする */

```
VOID func1(pal)
```

```
TINY pal;
```

```
{
```

```
    NAT     i;
```

```
    i = getplt(pal);
```

```
    setplt(pal, i | 0x700);
```

```
}
```

VOID setplt(pal, grbdat)

TINY pal;

NAT grbdat;

パレットをセットします。

pal で設定したいパレット番号を指定します。grbdat は 2 バイトの値で下位の 12 ビットを 4 ビット毎に区切ってグリーン、レッド、ブルーの輝度を指定します。ビットの割当は以下のとおりです。

```
Msb+-----+-----+-----+-----+Lsb
      無効   緑の輝度  赤の輝度  青の輝度
```

指定されたパレットデータは VRAM に保存されます。

3.4.6 描画関数

```
VOID    pset(x,y,color,logop)
NAT     x,y ;
TINY    color ;
TINY    logop ;
```

グラフィック画面に指定色で点を描きます。

(x,y)で指定した座標に color 色で点を描きます。MSX2 マシンの場合、ロジカルオペレーションの指定が有効です。

x, y は自動的に c_lastx, c_lasty に代入されます。

```
VOID    line(x1,y1,x2,y2,color,logop)
NAT     x1,y1,x2,y2 ;
TINY    color ;
TINY    logop ;
```

座標(x1,y1),座標(x2,y2)を結ぶ直線を color で指定した色で描きます。

MSX2 マシンの場合、logop で、BASIC と同様のロジカルオペレーションの指定が可能です。MSX1 の場合や、ロジカルオペレーションの指定の必要がない場合は、logop は省略せず必ず PSET を指定してください。

x2, y2 は自動的に c_lastx, c_lasty に代入されます。

使用例

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

/* 三角形の表示 */
VOID func1()
{
    line(0, 0, 100, 50, (TINY)15, PSET);
    line(c_lastx, c_lasty, 50, 100, (TINY)15, PSET);
    line(c_lastx, c_lasty, 0, 0, (TINY)15, PSET);
}

```

```
VOID    boxline(x1,y1,x2,y2,color,logop)
NAT     x1,y1,x2,y2 ;
TINY    color ;
```



```
TINY    logop ;
```

座標(x1,y1),座標(x2,y2)を対角とする四角形を描きます。それ以外は line と同じです。

```
VOID    boxfill(x1,y1,x2,y2,color,logop)
NAT     x1,y1,x2,y2 ;
TINY    color ;
TINY    logop ;
```

座標(x1,y1),座標(x2,y2)を対角とする四角形を塗りつぶします。それ以外は line と同じです。

```
VOID    circle(x,y,r,color,s_angl,e_angl,aspect)
NAT     x,y,r ;
TINY    color ;
int     s_angl,e_angl ;
NAT     aspect ;
```

座標(x,y)を中心とする半径 r の円を color 色で描きます。s_angl, e_angl の絶対値はそれぞれ円の開始角度, 終了角度を指定し, 増加することに反時計回りに角度が変化します。0 は真右, 2000H は真上, 4000H は真左, 6000H は真下を指定します。s_angl, e_angl が負の場合は円の中心へ向けての直線が引かれます。描画は開始角度から反時計回りに終了角度まで実行されます。

aspect はアスペクトレシオで, 1H から 7FFFH までの値が指定されると横長の, 8001H から 0FFFFH までの値が指定されると縦長の楕円が描画されます。8000H が指定されると真円となります。0 を指定した場合の動作は保証されません。r は常に長径を指定します。aspect と MSX-BASIC のアスペクトレシオの関係は以下ようになります。

```
if (aspect <= 32768)
    アスペクトレシオ = aspect / 32768;
else
    アスペクトレシオ = 0x8000 / abs( 65536 - aspect );
```

MSX-BASIC と違い color,s_angl,e_angl,aspect は省略できません。BASIC の

```
CIRCLE (128, 100),80
```

と同じ事をさせるには例えば以下のようにしてください。

```
circle(128, 100, 80, c_fore, 0, 0x7fff, 0x8000);
```

x, y は自動的に c_lastx, c_lasty に代入されます。

```
VOID    paint(x,y,color,b_color)
NAT     x,y ;
TINY    color,b_color ;
```

グラフィック画面の指定された境界色で囲まれた部分を指定の色で塗りつぶします。

本ルーチンはスクリーンモードによって多少機能が異なります。スクリーンモード 3 もしくは 5 から 8 では、b_color 色で囲まれた (x,y) を含む領域を color 色で塗りつぶします。これに対しスクリーンモード 2 ないし 4 の場合は境界色 (b_color) の指定は無効で、color で囲まれた部分を color 色で塗りつぶします。

スクリーンモードが 2 ないし 4 のときは、b_color には必ず TINY 型のダミーの値をいれてください。

x, y は自動的に c_lastx, c_lasty に代入されます。

```
TINY    point(x,y)
NAT     x,y ;
```

グラフィック画面の指定の座標の色コードを返します。引き数として色を調べたい座標を与えます。グラフィック関係の他の関数と違い、c_lastx, c_lasty を変更しません。

3.4.7 スプライト関係の関数

```
VOID    inispr(size)
TINY    size ;
```

全てのスプライトを初期化します。

スプライトパターンはすべて 0 でうめられ、カラーは、フォアグラウンドカラーに初期化されます。そして、スプライトは表示されない位置におかれます (水平位置がモード 0 から 3 では 209, モード 4 から 8 では 217)。

同時にスプライトサイズが size で指定した値に設定されます。size の取りうる値は以下の通りです。

- 0 : 8×8 モード, 拡大なし。
- 1 : 8×8 モード, 縦横拡大あり。
- 2 : 16×16 モード, 拡大なし。
- 3 : 16×16 モード, 縦横拡大あり。

4 : スプライトの表示を禁止.

4の「スプライトの表示を禁止」はMSX2でのみ有効であり、MSXでsizeに4を指定した場合の動作は保証されません。

```
NAT    calpat(pat)
TINY   pat ;
```

指定されたスプライトパターン(0から255)に対応するスプライトパターンテーブルの先頭アドレスを返します。

```
NAT    calatr(plane)
TINY   plane ;
```

planeで指定したスプライト面(0から31)に対応するスプライトアトリビュートテーブルの先頭アドレスを返します。

```
VOID    sprite(pat,data)
TINY    pat,*data ;
```

スプライトパターンを設定します。

patに設定したいスプライトパターン番号、dataにはスプライトデータへのポインタを与えます。スプライトサイズが8×8の時はdataでポイントされるデータの最初の8バイトが、16×16の場合は最初の32バイトが使用されます。長さが必要とされるデータ長に満たないときはパターンの終わりの方は不定です。

```
VOID    colspr(plane,color)
TINY    plane,*color ;
```

スプライトのライン毎の色を指定します。これにより、スプライトパターンのビットが1の部分の色をライン毎にかえることが可能です。このルーチンはスクリーン4から8でのみ有効です。その他のモードではスプライト色は次のputsprで一色のみの指定が可能です。スクリーン0から3に対して実行された場合の動作は保証されません。

planeでスプライト面番号を指定し、colorにライン毎の色、その他のデータへのポインタを与えます。

colorのデータはスプライトのサイズにより最初の8バイトまたは16バイトが有効となります。データがこれに満たない場合は終わりの方のラインの色は不定です。

colorデータの各バイトは具体的に次のような意味を持ちます。

NAT dx,dy ;
 TINY dp ;
 TINY logop ;

VRAM のページ sp の (sx1,sy1) から (sx2,sy2) の領域を、ページ dp の (dx,dy) へ転送します。dx, dy の値は自動的に、c_lastx, c_lasty に代入されます。

VOID cpyv2m(sx1,sy1,sx2,sy2,sp,dest)
 NAT sx1,sy1,sx2,sy2 ;
 TINY sp ;
 TINY *dest ;

VRAM のページ sp の (sx1,sy1) から (sx2,sy2) の領域を、dest でポイントされるメイン RAM に転送します。dest の先頭 2 ワードには x 方向, y 方向のドット数(それぞれ sx2-sx1+1, sy2-sy1+1)がこの順に設定され、その後に実際のデータが続くように設定されます。

sx2, sy2 の値は自動的に、c_lastx, c_lasty に代入されます。

VOID cpym2v(src,dir,dx,dy,dp,logop)
 TINY *src ;
 TINY dir ;
 NAT dx,dy ;
 TINY dp ;
 TINY logop ;

src でポイントされるメイン RAM の画像データを、VRAM のページ dp の (dx,dy) へ転送します。

転送するデータは、先頭部分に画像の x 方向, y 方向のドット数の情報を持っていないわけではありません。この形式を持つデータとしては、上の cpyv2m() によって VRAM から転送したデータや BASIC の COPY 命令により VRAM からファイルにコピーしたデータなどがあります。

dir では転送の方向を指定しますが、具体的な指定方法は MSX-BASIC の COPY 命令と同じですので、そちらをご参照下さい。

3.4.9 その他のグラフィック関係の関数

VOID totext()

スクリーンモードを強制的に以前のテキストモードに戻します。テキストモードでこのルーチンを実行しても何もおこりません。

引き数, 戻り値はありません。

```
VOID    grpprt(ch,logop)
char    ch ;
TINY    logop ;
```

グラフィックスクリーンに対して, `c_lastx`, `c_lasty` の位置に文字 `ch` を表示します。

`ch` は 1 バイトのキャラクターです。 `c_lastx`, `c_lasty` は次の文字の位置を示すように更新されます。 `logop` でロジカルオペレーションを指定します。 スクリーン 2 から 4 では `logop` は無視されますが、省略せずに必ず TINY 型の数値を指定してください。

スクリーン 2 から 4 では文字のドットがオンになっている部分のみがフォアグラウンド色で表示されます。 スクリーン 5 から 8 では文字のドットがオンになっている部分はフォアグラウンド色に、オフになっている部分はバックグラウンド色になり、これに対して指定されたロジカルオペレーションが実行され表示されます。

本ルーチンで漢字を表示することはできません。 漢字の出力には下の `knjprt()` を使用して下さい。

```
VOID    knjprt(JIS_knj,logop,mode)
NAT     JIS_knj ;
TINY    logop ;
TINY    mode ;
```

グラフィックスクリーンに対して, `c_lastx`, `c_lasty` の位置に, `JIS_knj` であたえた JIS 漢字コードに対応する文字を `mode` で示す表示モードで表示します。

本命令はスクリーン 5 から 8 に対してのみ使用できます。 `c_lastx`, `c_lasty` は次の文字の位置を示すように更新されます。 `logop` でロジカルオペレーションを指定します。

文字のドットがオンになっている部分はフォアグラウンド色に、オフになっている部分はバックグラウンド色になり、これに対して指定されたロジカルオペレーションが実行され表示されます。

`mode` に関しては、以下のような 3 種類の表示モードが有効です。

mode	モードの内容
0	16×16
1	偶数番目のドットを表示
2	奇数番目のドットを表示

```
VOID    glocate(x,y)
NAT     x,y ;
```

c_lastx, c_lasty にそれぞれ x, y を代入します。

```
VOID    setpg(dsppag, actpag)
TINY    dsppag, actpag ;
```

VRAM の表示ページと書き込みページの設定を行います。dsppag に表示ページ番号, actpag にアクティブページを指定します。戻り値はありません。

無効なページ番号を指定したとか, MSX 上で実行した場合の動作は保証されません。実際に切り替え可能なページは以下のとおりです。

画面モード	VRAM64K マシン	VRAM128K マシン
5	0 から 1 ページ	0 から 3 ページ
6	0 から 1 ページ	0 から 3 ページ
7	使用不能	0 から 1 ページ
8	使用不能	0 から 1 ページ

使用例

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

/* 表示ページのみを 1 に変更し, アクティブページは変えない */
VOID func1()
{
    setpg((TINY)1, c_ apage);
}
```

```
NAT    vramsize()
```

VRAM のサイズを返す関数です。今の所 MSX なら 16 を, VRAM64K の MSX2 なら 64 を, VRAM128K の MSX2 なら 128 を返します。

使用例

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

/* VRAM サイズのチェック */
VOID chk_size()
{
    if(vramsize() < 128) {
        printf("I need at least 128K bytes of VRAM to run\n");
        exit(1);
    }
}
```

第 4 章 数値演算関数パッケージ

4.1 数値演算関数パッケージの概要

現在の MSX-C によってサポートされる数値型は整数型と文字型のみとなっています。しかし、本格的な MSX-DOS 上でのユーティリティの作成や一般的なプログラムの高速化のために MSX-C を利用する場合には、やはり 5 桁以上の有効桁数や浮動小数による演算が必要となってきます。それを C 言語によって実現しようとする、手間やソースコードが急激に増大してしまい、実現をあきらめたり、他の処理系で行うことになってしまいました。そのようなことをプログラム作成者が意識せずにあたかも新しい数値型があるようにサポートするためのものが数値演算関数パッケージです。

数値演算関数パッケージでは 4 バイト整数型 (Long 型) と MSX-BASIC とおなじ倍精度実数型 (double 型) の 2 種類をサポートしています。

4 バイト整数型は整数型の倍の有効桁数 (9 桁) ですので大抵のユーティリティはこれで十分でしょう。サポートされる演算は四則、剰余、比較、論理演算、シフト・ローテイトです。

もう一つの型、倍精度実数型は唯一の実数型で、演算は有効桁数 14 桁の BCD によって行われます。サポートされる演算・関数は四則、べき乗、三角関数、指数・対数関数などです。

以上の他に従来の整数型との変換関数や、4 バイト整数型と倍精度実数型の変換関数が用意されています。また、新たに供給された関数の他に書式付き入出力関数 (scanf(), printf() など) が 4 バイト整数型と倍精度実数型の入出力のために拡張されています。

4.2 数値演算関数パッケージの使用法

数値演算関数パッケージを利用するには専用のヘッダファイル (math.h) をプログラムの先頭でインクルードしなければなりません。普通は標準ヘッダファイル (stdio.h) の次でインクルードします。だいたい次のようになります。

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

/*      1度のラジアンを計算      */
main()
{
    XDOUBLE pi, k180;
    XDOUBLE deg;
```



```

    atoxd(&pi, "3.1415926536");
    atoxd(&k180, "180");
    xddiv(&deg, &pi, &k180); /* deg = pi/180; */

    printf("1 deg = %f rad\n", &deg);
}

```

ヘッダを読み込めばあとは、従来どおりのコーディングができます。コンパイル・リンク手順については、本パッケージの他のモジュールと同様に行えます。

4.3 各数値型の詳細

ここでは4バイト整数型(以下 Long 型)と倍精度実数型のそれぞれについて詳細を示します。

4.3.1 型の定義

A) Long 型の定義

Long 型は符号付き Long 型の1種類で符号無し Long 型は存在しません。しかし、入出力においてサポートされていますので少し注意することで符号無し Long 型のように扱えます。(これについては「4.3.3 定数代入関数」, 「4.3.7 書式付き入出力関数の拡張」を参照して下さい。)符号付き Long 型の定義は次のようになります。

使用する領域は4バイトで、1バイト目を値の最下位、4バイト目を値の最上位とします。符号は4バイト目の第7ビットとし、2の補数表現とします。MSX-C上では、次のように型を定義しています。(この型の定義は数値演算関数パッケージのヘッダファイル math.h に含まれています。)

```

typedef struct {
    char    byte[4];
} SLONG;

```

B) 倍精度実数型の定義

倍精度実数型は MSX-BASIC で使用される BCD 表現の倍精度実数と同じものです。詳しくは、MSX-BASIC のリファレンスマニュアルをご覧ください。倍精度実数型の定義は次のようになります。

使用する領域は8バイトで、1バイト目を符号(ビット7)と指数部(ビット0から6)、2バイト目から8バイト目を仮数部1桁目から14桁目までとして使用しています。MSX-C上では、次の

ように型を定義しています。(この型の定義は数値演算関数パッケージのヘッダファイルに含まれています。)

```
typedef struct {
    char    byte[8];
} XDOUBLE;
```

4.3.2 変数の宣言

Long 型や倍精度実数型の変数を使用する場合には、他の型と同じ様に使用する前に宣言する必要があります。宣言は次のようにします。

```
SLONG    l;
XDOUBLE  d;
```

変数の初期化は、外部変数または静的変数の場合可能です。次のようにキャラクタ型の配列として初期化をします。(これについてのもう少し詳しい説明が 4.6.2 B)にありますのでそちらもご覧下さい。)

```
SLONG    l = {0x78, 0x56, 0x34, 0x12};          /* 外部変数のとき */
static SLONG    m = {10, 20, 30, 40};          /* 静的変数のとき */
XDOUBLE  d = {0x46, 0x12, 0x34, 0x56, 0x78, 0x90, 0x12, 0x34};
static XDOUBLE  e = {0x43, 0x10, 0, 0, 0, 0, 0, 0}; /* 静的変数のとき */
```

Long 型変数 l は 16 進数で 12345678 を表し、m は 10 進数で 673059850 を表します。また、倍精度実数型変数 d は 123456.78901234 を表し、e は 100 を表します。

4.3.3 定数代入関数

10 進定数を Long 型変数や倍精度実数型変数に代入するには、定数代入関数を使って文字列を数値に変換します。

```
SLONG    l;
XDOUBLE  d;
atosl(&l, "4980205");
atoxd(&d, "498.0205");
```

このようにすることで Long 型変数 l には 10 進数で 4980205 という数値が代入され、倍精度実数型変数 d には 498.0205 という数値が代入されます。このとき `atosl()` 関数はオーバーフローを無視するので符号無し Long 型として代入することもできます。また、10 進数以外の定数で代入し

たい場合には、`sscanf()`関数で実現できます。

4.3.4 変数から変数への代入

変数から変数への代入は、`Long`型どうしや倍精度実数型どうしで行えません。型が違う場合は変換関数を使用する必要があります。(変換関数については「4.3.8 変換関数」を参照して下さい。)

```
SLONG l, m;
XDOUBLE d, e;
slcpy(&l, &m); /* l = m; Long 型の場合 */
xdcpy(&d, &e); /* d = e; 倍精度実数型の場合 */
```

4.3.5 演算関数

演算関数とは、何かパラメータを与えると演算結果を返すものを指します。この場合、四則演算も演算関数に含まれます。演算結果はユーザによって指定された領域に格納され、関数値としては返されません。関数値としては `Long` 型の場合は演算結果へのポインタが、倍精度実数型の場合は演算結果の状態が `STATUS` 型で返されます。

オーバーフローなどの演算結果について `Long` 型はエラー情報を返しませんので注意して下さい。しかし倍精度実数型についてはエラーが発生したかどうかを (`OK` または `ERROR` で)返しますから、必要であればチェックを行って下さい。

```
SLONG l, m, n;
XDOUBLE d, e, f;

... /* 何か別の処理 */
sladd(&l, &m, &n); /* l = m + n; Long 型で和を求める。 */

/* d = e * f; 倍精度実数型で積を求める。 */
if(xdmul(&d, &e, &f) == ERROR) {
    fprintf(stderr, "Math pack error\n");
    exit(1);
}
```

パラメータの説明をしましょう。第1パラメータは、演算結果を返す変数へのポインタです。第2パラメータ、それからあれば第3パラメータは演算の対象となる変数へのポインタです。これらのパラメータの順は、式で表現した場合と同じようになっています。また、第3パラメータは整数を指定する関数もありますので注意して下さい。

パラメータの変数どうしが重なっていても正常に動作しますので、不要な変数を持つ必要がありません。つまり、`sladd(&l, &l, &l);` とすることで `Long` 型変数 `l` は2倍になり、`xbsub(&`

d.&d,&d)； とすることで、倍精度実数型変数 d は 0 に初期化することができます。

2つの型のうち、Long 型の演算関数はほとんどが演算結果へのポインタを返すので次のようなことができます。

```
SLONG  a, b, c;
        smul(&c, atol(&a, "9938"), atol(&b, "1985"));
```

この記述は `c = (a=9938) * (b=1985);` と同じことを Long 型で行います。

4.3.6 比較関数

比較関数は、1つの変数値とゼロまたは2つの変数値を比較してその結果を整数値の-1,0,1のどれかで返すものです。パラメータは Long 型か倍精度実数型へのポインタで、関数値は整数型です。関数値が整数ですので条件式の必要な場所に直接記述することができます。

```
SLONG  l, m;
XDOUBLE d;

...
if(sicmp(&l, &m) > 0) { /* 何か別の処理 */
} else { /* l > m のときの処理 */
}
/* l <= m のときの処理 */

while(xdsign(d) > 0) { /* d > 0 のときの処理 */
}
}
```

4.3.7 書式付き入出力関数の拡張

A) Long 型の書式付き入出力について

Long 型を書式付きで入力するには `scanf()` 関数中で、`int` 型を入力するための `%d` の代わりに `%ld` を指定します。他の `%x`, `%o`, `%u` も同様に `%lx`, `%lo`, `%lu` と指定することができます。格納先を示す引き数は `int` 型などと同じで Long 型へのポインタを指定します。

Long 型を書式付きで出力するには `printf()` 関数中で、`int` 型を出力するための `%d` の代わりに `%ld` を指定します。他の `%x`, `%o`, `%u` も同様にできます。また、対応する値には変数そのものではなく、Long 型変数へのポインタを渡します。

```

int    i;
SLONG  l;

scanf("%d %ld", &i, &l);
printf("integer:%d long:%ld\n", i, &l);

```

B) 倍精度実数型の書式付き入出力について

倍精度実数型を書式付きで入力するには `scanf()` 関数中で、`int` 型を入力するための `%d` の代わりに `%f` または `%e` を指定します。`%f`、`%e` どちらを指定した場合にも固定小数点、浮動小数点の両方の方式で入力することができます。ただし、空白で区切るという制限がありますので注意して下さい。格納先を示す引き数は `int` 型などのときと同じで倍精度実数型へのポインタを指定します。

倍精度実数型を書式付きで出力するには `printf()` 関数中で、`int` 型を出力するための `%d` の代わりに `%f` または `%e` を指定します。`%f` が指定された場合には固定小数点方式で、`%e` が指定された場合には浮動小数点方式で表示されます。また、対応する値には変数そのものではなく、倍精度実数型変数へのポインタを渡します。

```

int    i;
XDOUBLE f;

scanf("%d %f", &i, &f);
printf("integer:%d double:%f\n", i, &f);

```

A)、B)と従来の型をまとめると使用可能なものは次のような表になります。

	<code>scanf()</code>	<code>printf()</code>
<code>int</code>	<code>%d, %x, %o</code>	<code>%d, %x, %o</code>
<code>unsigned</code>	<code>%u, %x, %o</code>	<code>%u, %x, %o</code>
<code>char</code>	<code>%c</code>	<code>%c</code>
<code>string(char *)</code>	<code>%s</code>	<code>%s</code>
<code>long(SLONG)</code>	<code>%ld, %lx, %lo</code>	<code>%ld, %lx, %lo</code>
<code>unsigned long</code> (<code>SLONG</code> 入出力のみ)	<code>%lu, %lx, %lo</code>	<code>%lu, %lx, %lo</code>
<code>double(XDOUBLE)</code>	<code>%f, %e</code>	<code>%f, %e</code>

4.3.8 変換関数

変換関数は頻度の高い型変換を直接行うための関数です。ですから、ある型からある型への変換関数が必ず存在するわけではありません。また、定数代入関数も変換関数の一部と考えることができます。

下の表では `scanf()`、`printf()` を含めて 1 関数で変換可能なもの、変換不要なものを示します。

表 変換可能な型

(関数または = のとき直接変換可能、No のとき直接変換不可能)

dst type \ src type	int	unsigned	char	string	long	unsigned long	double
int	=	=	=	<code>sprintf()</code>	<code>itosl()</code>	No	<code>itoxd()</code>
unsigned	=	=	=	<code>sprintf()</code>	<code>uitosl()</code>	<code>uitosl()</code>	No
char	=	=	=	<code>sprintf()</code>	No	No	No
string(char *)	<code>atoi()</code>	<code>atoi()</code>	=	<code>strcpy()</code>	<code>atosl()</code>	<code>atosl()</code>	<code>atoxd()</code>
long(SLONG)	No	No	No	<code>sprintf()</code>	<code>slcpy()</code>	<code>slcpy()</code>	<code>sltoxd()</code>
unsigned long	No	No	No	<code>sprintf()</code>	<code>slcpy()</code>	<code>slcpy()</code>	<code>ultoxd()</code>
double (XDOUBLE)	<code>xdtol()</code>	No	No	<code>sprintf()</code>	<code>xdtosl()</code>	No	<code>xdcpy()</code>

「=」がある変換は直接代入またはキャストによって変換できます。関数名がある変換はその関数によって変換ができます。「No」と表示されているものに関しては 1 度 `sprintf()` でストリングに変換してから、`sscanf()` や定数代入関数を使って再度変換することで実現できます。また、長さが違いますが内部表現が同じである long から int や unsigned への変換はキャストを使うことでできます。(例：`i = *(int *)&l;`)

4.4 数値演算関数一覧

定数代入関数

関数名	用途	参照ページ
<code>atosl</code>	数値を表す文字列を Long 型に変換	45
<code>atoxd</code>	数値を表す文字列を倍精度型変数に変換	45

変数から変数への代入

関数名	用途	参照ページ
slcpy	Long 型の値を Long 型の変数に代入	45
xdcpy	倍精度実数型の値を倍精度実数型の変数に代入	46

演算関数 A) Long 型の演算関数

関数名	用途	参照ページ
sladd	Long 型どうしの加算	46
slsub	Long 型どうしの減算	46
slmul	Long 型どうしの乗算	46
sldiv	Long 型どうしの除算(符号つき)	46
slmod	Long 型どうしの剰余算(符号つき)	46
uldiv	Long 型どうしの除算(符号なし)	47
ulmod	Long 型どうしの剰余算(符号なし)	47
slneg	Long 型の数値に-1 をかけたものを返す	47
slabs	Long 型の絶対値を返す	47
slnot	Long 型をビットごとに反転させる	47
sland	Long 型でビットごとの論理積を求める	47
slor	Long 型でビットごとの論理和を求める	48
slxor	Long 型でビットごとの排他的論理和を求める	48
slls	Long 型を算術的左シフト行う	48
slll	Long 型を論理的左シフト行う	48
slsra	Long 型を算術的右シフト行う	48
slsrl	Long 型を論理的右シフト行う	49
slrlc	Long 型を左ローテイト行う	49
slrl	〃	49
slrrc	Long 型を右ローテイト行う	49
slrr	〃	50

B) 倍精度実数型の演算関数

関数名	用途	参照ページ
xdadd	倍精度実数型どうしの加算	50
xdsb	倍精度実数型どうしの減算	50
xdmul	倍精度実数型どうしの乗算	50
xddiv	倍精度実数型どうしの除算	50
xdpow	倍精度実数型どうしのべき乗を計算する	51

関数名	用途	参照ページ
xdneg	数値に-1 をかけたものを返す	51
xdfabs	数値の絶対値を求める	51
xdfix	数値の小数部分を切り捨てる	51
xdfloor	数値を越えない最大の整数を求める	51
xdceil	数値より小さくない最小の整数を求める	51
xdsqrt	倍精度実数型で平方根を計算する	52
xdsin	数値をラジアンとして正弦(サイン)を計算する	52
xdcos	数値をラジアンとして余弦(コサイン)を計算する	52
xdtan	数値をラジアンとして正接(タンジェント)を計算する	52
xdatn	数値の逆正接(アークタンジェント)を計算する	52
xdlog	数値の対数を計算する	52
xdexp	e に対する指数を計算する	52
xdrnd	乱数を発生させる	53

比較関数

関数名	用途	参照ページ
slsgn	Long 型の符号値を返す	53
slcmp	Long 型どうして比較し結果を返す	53
xdsgn	倍精度実数の符号値を返す	53
xdcmp	倍精度実数型どうして比較し結果を返す	54

変換関数

関数名	用途	参照ページ
sltoa	Long 型の数値を文字列に変換する	54
itosl	整数から Long 型に変換する	54
uitosl	符号なし整数から Long 型に変換する	54
xdtoa	倍精度実数型を文字列に変換する	54
itoxd	整数型から倍精度実数型に変換する	55
xdtoi	倍精度実数型から整数型に変換する	55
sltoxd	Long 型から倍精度実数型に変換する	55
ultoxd	Long 型の値を符号なしとして倍精度実数型に変換する	55
xdtosl	倍精度実数型を Long 型に変換する	55

4.5 数値演算関数リファレンス

4.5.1 定数代入関数

```
SLONG  *atosl(ans,s)
SLONG  *ans;
char    *s;
```

数値を表す文字列を変換して Long 型変数に代入します。関数値としては、第 1 パラメータと同じものが返されます。

```
STATUS  atoxd(ans,s)
XDOUBLE *ans;
char    *s;
```

数値を表す文字列を変換して倍精度実数型変数に代入します。関数値としては、変換が成功したかのステータスが返されます。

4.5.2 変数から変数への代入

```
SLONG  *slcpy(ans,p1)
SLONG  *ans,*p1;
```

p1 で示される Long 型変数の値が ans で示される Long 型の変数に代入されます。つまり、`*ans = *p1;` なることを Long 型で行います。

```
XDOUBLE *xdcpy(ans,p1)
XDOUBLE *ans,*p1;
```

p1 で示される倍精度実数型の値が ans で示される倍精度実数型の変数に代入されます。

4.5.3 演算関数

A) Long 型の演算関数

```
SLONG *sladd(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうして加算を行います。つまり、 $*ans = *p1 + *p2$; なる計算を Long 型で行います。

```
SLONG *slsub(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうして減算を行います。つまり、 $*ans = *p1 - *p2$; なる計算を Long 型で行います。

```
SLONG *slmul(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうして乗算を行います。つまり、 $*ans = *p1 * *p2$; なる計算を Long 型で行います。

```
SLONG *sldiv(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうして除算を行います。つまり、 $*ans = *p1 / *p2$; なる計算を Long 型で行います。この関数は符号つき Long 型を想定しています。

```
SLONG *slmod(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうしの剰余を計算します。つまり、 $*ans = *p1 \% *p2$; なる計算を Long 型で行います。この関数は符号つき Long 型を想定しています。

```
SLONG *uldiv(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうして除算を行います。つまり、 $*ans = *p1 / *p2$; なる計算を Long 型で行います。この関数は符号無し Long 型を想定しています。

```
SLONG *ulmod(ans,p1,p2)
SLONG *ans,*p1,*p2;
```

Long 型どうしの剰余を計算します。つまり、 $*ans = *p1 \% *p2$ ； なる計算を Long 型で行います。この関数は符号無し Long 型を想定しています。

```
SLONG  *slneg(ans,p1)
SLONG  *ans, *p1 ;
```

Long 型の数値に -1 をかけたものを ans に返します。 $*ans = - *p1$ ； と同じ事を Long 型で行います。数値がマイナスの最大値 -2147483648 であった場合には変化がありません。 ans と $p1$ が同じであった場合は、指定された変数の値に -1 がかけられます。

```
SLONG  *slabs(ans,p1)
SLONG  *ans, *p1 ;
```

Long 型の絶対値を ans に返します。 $*ans = abs(*p1)$ ； と同じ事を Long 型で行います。数値がマイナスの最大値 -2147483648 であった場合には変化がありません。

```
SLONG  *slnot(ans,p1)
SLONG  *ans, *p1 ;
```

Long 型をビットごとに反転させます。つまり、 $*ans = \sim *p1$ ； なる計算を Long 型で行います。

```
SLONG  *sland(ans,p1,p2)
SLONG  *ans, *p1, *p2 ;
```

Long 型でビットごとの論理積を求めます。つまり、 $*ans = *p1 \& *p2$ ； なる計算を Long 型で行います。

```
SLONG  *slor(ans,p1,p2)
SLONG  *ans, *p1, *p2 ;
```

Long 型でビットごとの論理和を求めます。つまり、 $*ans = *p1 \mid *p2$ ； なる計算を Long 型で行います。

```
SLONG  *slxor(ans,p1,p2)
SLONG  *ans, *p1, *p2 ;
```

Long 型でビットごとの排他的論理和を求めます。つまり、 $*ans = *p1 \wedge *p2$ ； なる計算を

Long 型で行います。

```
SLONG  *slla(ans,p1,k)
SLONG  *ans,*p1;
TINY   k;
```

Long 型を k ビット算術的左シフト行います。つまり、 $*ans = *p1 \ll k$; なる計算を Long 型で行います。最後に追い出されたビットは BOOL `slcy` に格納されます。この関数はオーバーフローを考えません。

`slcy` ←

31 ← *p1 ← 0

 ← 0

```
SLONG  *sll(ans,p1,k)
SLONG  *ans,*p1;
TINY   k;
```

Long 型を k ビット論理的左シフト行います。つまり、 $*ans = *p1 \ll k$; なる計算を Long 型で行います。最後に追い出されたビットは BOOL `slcy` に格納されます。この関数は `slla()` と全く同じ動作をします。

`slcy` ←

31 ← *p1 ← 0

 ← 0

```
SLONG  *sra(ans,p1,k)
SLONG  *ans,*p1;
TINY   k;
```

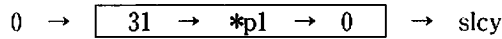
Long 型を k ビット算術的右シフト行います。つまり、符号ビットを保存しながら $*ans = *p1 \gg k$; なる計算を Long 型で行います。最後に追い出されたビットは BOOL `slcy` に格納されます。この関数はオーバーフローを考えません。

31 → *p1 → 0

 → `slcy`

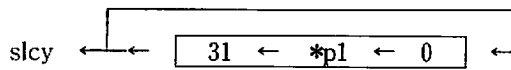
```
SLONG  *srl(ans,p1,k)
SLONG  *ans,*p1;
TINY   k;
```

Long 型を k ビット論理的右シフト行います。つまり、 $*ans = *p1 \gg k$; なる計算を Long 型で行います。最後に追い出されたビットは BOOL `slcy` に格納されます。



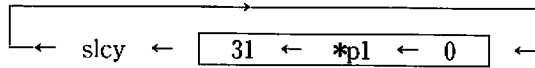
```
SLONG *srlc(ans,p1,k)
SLONG *ans,*p1;
TINY k;
```

Long 型を k ビット左ローテイトを行います。ビット 31 から追い出されたビットはビット 0 に移ります。最後に追い出されたビット 31 は BOOL slcy に格納されます。



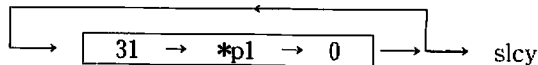
```
SLONG *srl(ans,p1,k)
SLONG *ans,*p1;
TINY k;
```

Long 型を k ビット左ローテイトを行います。ビット 31 から追い出されたビットは BOOL slcy に格納され、slcy はビット 0 に移ります。最後に追い出されたビット 31 が BOOL slcy に格納されます。



```
SLONG *srrc(ans,p1,k)
SLONG *ans,*p1;
TINY k;
```

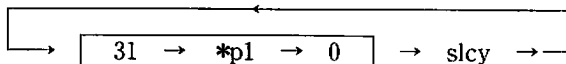
Long 型を k ビット右ローテイトを行います。ビット 0 から追い出されたビットはビット 31 に移ります。最後に追い出されたビット 0 は BOOL slcy に格納されます。



```
SLONG *srr(ans,p1,k)
SLONG *ans,*p1;
TINY k;
```

Long 型を k ビット右ローテイトを行います。ビット 0 から追い出されたビットは BOOL slcy

に格納され、slcy はビット 31 に移ります。最後に追い出されたビット 0 が BOOL slcy に格納されます。



B) 倍精度実数型の演算関数

```
STATUS xdadd(ans,p1,p2)
XDOUBLE *ans,*p1,*p2;
```

倍精度実数型どうしの加算を行います。つまり、 $*ans = *p1 + *p2$ ； なる計算を倍精度実数型で行います。

```
STATUS xdsb(ans,p1,p2)
XDOUBLE *ans,*p1,*p2;
```

倍精度実数型どうしの減算を行います。つまり、 $*ans = *p1 - *p2$ ； なる計算を倍精度実数型で行います。

```
STATUS xdmul(ans,p1,p2)
XDOUBLE *ans,*p1,*p2;
```

倍精度実数型どうしの乗算を行います。つまり、 $*ans = *p1 * *p2$ ； なる計算を倍精度実数型で行います。

```
STATUS xddiv(ans,p1,p2)
XDOUBLE *ans,*p1,*p2;
```

倍精度実数型どうしの除算を行います。つまり、 $*ans = *p1 / *p2$ ； なる計算を倍精度実数型で行います。

```
STATUS xdpow(ans,p1,p2)
XDOUBLE *ans,*p1,*p2;
```

倍精度実数型のべき乗を計算します。つまり、 $*ans = pow(*p1, *p2)$ ； なる計算を倍精度実数型で行います。

```
STATUS xdneg(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値に -1 を掛けたものを返します。つまり、 $*ans = - *p1$ ； なる計算を倍精度実数型で行います。

```
STATUS xdfabs(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値の絶対値を求めます。つまり、 $*ans = fabs(*p1)$ ； なる計算を倍精度実数型で行います。

```
STATUS xdfix(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値の小数部分を切り捨てます。MSX-BASICのFIX関数と同じ事を倍精度実数型で行います。つまり、数値のゼロよりの整数を求めます。

```
STATUS xdfloor(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値を超えない最大の整数を求めます。つまり、 $*ans = floor(*p1)$ ； なる計算を倍精度実数型で行います。

```
STATUS xdceil(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値より小さくない最小の整数を求めます。つまり、 $*ans = ceil(*p1)$ ； なる計算を倍精度実数型で行います。

```
STATUS xdsqrt(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

倍精度実数型で平方根を計算します。つまり、 $*ans = sqrt(*p1)$ ； なる計算を倍精度実数型で行います。

```
STATUS xdsin(ans,p1)
```

```
XDOUBLE *ans, *p1 ;
```

数値をラジアンとして正弦(サイン)を計算します。つまり、 $*ans = sin(*p1)$ ； なる計算を倍

精度実数型で行います。

```
STATUS xdcos(ans,p1)
XDOUBLE *ans, *p1;
```

数値をラジアンとして余弦(コサイン)を計算します。つまり、 $*ans = \cos(*p1)$ ；なる計算を倍精度実数型で行います。

```
STATUS xdtan(ans,p1)
XDOUBLE *ans, *p1;
```

数値をラジアンとして正接(タンジェント)を計算します。つまり、 $*ans = \tan(*p1)$ ；なる計算を倍精度実数型で行います。

```
STATUS xdatn(ans,p1)
XDOUBLE *ans, *p1;
```

数値の逆正接(アークタンジェント)を計算します。つまり、 $*ans = \text{atan}(*p1)$ ；なる計算を倍精度実数型で行います。

```
STATUS xdlog(ans,p1)
XDOUBLE *ans, *p1;
```

数値の対数を計算します。つまり、 $*ans = \log(*p1)$ ；なる計算を倍精度実数型で行います。

```
STATUS xdexp(ans,p1)
XDOUBLE *ans, *p1;
```

e に対する指数を計算します。つまり、 $*ans = \exp(*p1)$ ；なる計算を倍精度実数型で行います。

```
STATUS xdrnd(ans,p1)
XDOUBLE *ans, *p1;
```

0 から 1 までの乱数を得ます。 $*p1 < 0$ の場合は乱数系列の初期化を行い、 $*p1 = 0$ の場合は前回の値を返し、 $*p1 > 0$ の場合は次の乱数を発生させそれを返します。

4.5.4 比較関数

```
int      slsgn(p1)
SLONG   *p1 ;
```

指定された Long 型の符号値を返します。結果は下に示します。

			slsgn()
*p1	>	0	なら 1
*p1	=	0	なら 0
*p1	<	0	なら -1

```
int      slcmp(p1,p2)
SLONG   *p1,*p2 ;
```

Long 型どうしで比較を行い、その結果を下に示すように返します。

			slcmp()
*p1	>	*p2	なら 1
*p1	=	*p2	なら 0
*p1	<	*p2	なら -1

```
int      xdsgn(p1)
XDOUBLE *p1 ;
```

指定された倍精度実数の符号値を返します。結果は下に示します。

			xdsgn()
*p1	>	0	なら 1
*p1	=	0	なら 0
*p1	<	0	なら -1

```
int      xdcmp(p1,p2)
XDOUBLE *p1, *p2;
```

倍精度実数型どうして比較を行い、その結果を下に示すように返します。

			xdcmp()	
*p1	>	*p2	なら	1
*p1	=	*p2	なら	0
*p1	<	*p2	なら	-1

4.5.5 変換関数

```
char      *sltoa(s,l,radix)
char      *s;
SLONG     *l;
TINY      radix;
```

Long 型の数値を radix 進数として文字列に変換したものを s で示される領域に格納し、s を返します。radix のビット 7 が立っている場合には符号なしの数値として扱います。

```
SLONG     *itosl(ans,i)
SLONG     *ans;
int       i;
```

整数から Long 型に変換します。このとき符号の拡張が行われます。

```
SLONG     *uitosl(ans,ui)
SLONG     *ans;
unsigned  ui;
```

符号無し整数から Long 型に変換します。もともと符号はありませんから、拡張は行われません。

```
char      *xdtoa(s,d,digits)
char      *s;
XDOUBLE   *d;
TINY      digits;
```

倍精度実数型の数値を文字列として変換したものを `s` で示される領域に格納し、`s` を返します。`digits` で小数点以下の桁数を指定します。`digits` のビット 7 が立っていない場合は固定小数点形式で、立っている場合は浮動小数点形式に変換します。

```
STATUS itoxd(d,i)
XDOUBLE *d;
int      i;
```

整数型から倍精度実数型に変換します。

```
STATUS xdtol(i,d)
int      *i;
XDOUBLE *d;
```

倍精度実数型から整数型に変換します。小数点以下は切り捨てられ、変換された数値が整数の範囲外である場合には関数値として `ERROR` が返されます。

```
STATUS sltox(d,l)
XDOUBLE *d;
SLONG   *l;
```

Long 型から倍精度実数型に変換します。

```
STATUS ultox(d,ul)
XDOUBLE *d;
SLONG   *ul;
```

Long 型の値を符号無しとして倍精度実数型に変換します。

```
STATUS xdtosl(l,d)
SLONG   *l;
XDOUBLE *d;
```

倍精度実数型を Long 型に変換します。小数点以下は切り捨てられます。変換された数値が Long 型の範囲外であってもエラーが返されることはありません。

4.6 数値演算関数を使う上での注意点, 留意点

4.6.1 使用上の注意点

A) 倍精度実数型を使用するためにはスタック領域がページ 2 または 3 にある必要があります (スタック領域に Math-Pack を呼び出すコードを置くため)。通常スタック領域は指定のページにあります。自動変数として非常に大きな配列を宣言するとページ 0 または 1 になる場合があります。

B) Long 型では演算や変換を行ってもその結果の範囲についてのチェックをしません。これは整数型などと同じですが、倍精度実数型から変換する場合には事前に範囲のチェックなどをして不良動作をしないようにすることをお勧めします。

C) 倍精度実数型をアスキー文字列として入出力する場合、その一時バッファとして BUF (0F55EH から 258 バイト) と FBUFFER (0F7C5H から 43 バイト) を使用しますので、この領域を使用する場合には十分ご注意下さい。

4.6.2 テクニック集

A) 数値演算関数パッケージを使用する場合にはヘッダ `math.h` をインクルードしますが、その中には `scanf()`, `printf()` を拡張するための `#define` 文があります。これによって `scanf()`, `printf()` が置き換えられています。拡張されていない `scanf()`, `printf()` を使用したい場合には、`#include <math.h>` 以降の行で `#undef printf` などとしてください。なおこの場合も拡張された `scanf()`, `printf()` は `mscanf()`, `mprintf()` として使用することが出来ます。

```
#define printf  mprintf
#define fprintf mfprintf
#define sprintf msprintf
#define scanf  mscanf
#define fscanf mfscanf
#define sscanf msscanf
```

(math.h より抜粋)

また、本パッケージライブラリファイルには Long 型, 倍精度実数型の両方を一度に利用することができるよう `scanf()`, `printf()` が拡張されています。しかし、ソースファイル `prsc.c` の `#include <math.h>` の前でコンパイルスイッチ `NOUSESL` または `NOUSEXD` を定義し、リコンパイルすることで Long 型, 倍精度実数型のそれぞれを使わないライブラリが作成できます。新た

に作成されたライブラリを使用する場合には、アプリケーションプログラムでは、`#include <math.h>`の前でコンパイルスイッチ `NOUSESL` または `NOUSEXD` を定義し、リンク時、新たなライブラリをリンクする必要があります。

例 倍精度実数型を使用しない例

```
#define NOUSEXD      /* 倍精度実数型を使用しない宣言 */
/* この行をいれてコンパイルした prsc.c のリロケートブルオブジェクトモジュール
   も必要です。
   必ず math.h の前にいれてコンパイルして下さい。
   Long 型を使用しない場合は #define NOUSESL とする。
*/

#include <math.h>

main()
{
  ...
}
```

B) プログラムの作成には初期化というものは重要なものです。初期化には定数代入関数を使用してもいいのですが、それだとしてもオブジェクトサイズが大きくなってしまいます。ここでは `Long` 型を中心に別の初期化方法について紹介します。

`Long` 型を 0 に初期化したい場合にはキャストを利用して次のようにすると効率的です。

```
*((int *)&longvar+1) = *(int *)&longvar = 0;
```

-1 に初期化したい場合には 0 の代わりに -1 を指定すればできます。

`Long` 型を 0 に初期化した後は `char` 型から `Long` への変換が簡単にできます。

```
*((int *)&longvar+1) = *(int *)&longvar = 0;
longvar.byte[0] = charvar;
```

静的な(変数宣言時の)初期化についても述べましょう。変数の宣言時には単なる配列のように初期化ができます。

```
SLONG longvar = {0x10, 0x27, 0, 0}; /* longvar=10000: */
XDOUBLE dblvar = {0x43, 0x36, 0x52, 0x42, 0x20, 0, 0, 0};
/* dblvar=365.2422: */
```

`Long` 型の初期化のためのデータは `MSX-BASIC` で次のようなプログラムを実行させれば簡単

に作成できます。

```
100 A#=10000           '求めたい値を代入しておく
110 FOR I=1 TO 4       'Long型は4バイト
120   PRINT A#-INT(A#/256)*256; '1バイトを10進数で表示
130   A#=INT(A#/256)   '次のバイトのための準備
140   NEXT
```

倍精度実数型の初期化のためのデータも次のようにできます。

```
100 A#=365.2422       '求めたい値を代入しておく
110 I=VARPTR(A#)      '格納位置を求める
120 FOR J=1 TO I+7    '倍精度実数型は全部で8バイト
130   PRINT PEEK(J);  '1バイトを10進数で表示
140   NEXT
```

内部の表現がわかっている場合には変数宣言時以外にも、各型のメンバに代入することで動的に初期化ができます。

```
/* longvar=10000 ; */
longvar.byte[0] = (char)0x10;
longvar.byte[1] = (char)0x27;
*((int *)&longvar + 1) = 0;
```

C) slbuf, slcy について

数値演算関数パッケージには slbuf, slcy という外部変数が宣言されています。

slbuf は一時的な演算結果の格納に使います。使用するには演算関数などの結果格納の指定(第1パラメータ)に NULL ポインタを指定します。すると関数値として slbuf へのポインタが返されます。演算結果は使うがいつまでもとっておく必要がない場合に使うと便利です。例えば

```
printf("%ld%fn", sladd(NULL, &a, &b));
```

とすれば不要な変数を宣言せずに Long 型変数 a と b の和を表示することができます。NULL の代わりに ERROR を指定しても同様なことができます。この場合 NULL とは使用される領域が異なります。

slcy はシフト・ローテイト関数使用時にキャリーフラグとして動作します。つまりシフトやローテイトをした後変数から追い出されたビットの状態がこの変数に入ります。ユーザは自由にこの変数を参照したり、変更することができます。ただし、slcy は BOOL 型変数なので YES または NO のいずれかの値しか代入することができません。また、論理演算関数を行っても NO になることはありません。

slbuf, slcy はどちらも Long 型専用のもので倍精度実数型の関数では使用することはできません。

第5章 その他の関数

5.1 その他の関数の概要

ここでは関数を以下の3つに分類してその概要を解説します。

- ・スロットを管理する関数
- ・BIOS を呼ぶ関数
- ・その他

5.1.1 スロットを管理する関数

スロット間での関数呼び出しをおこなったり、ROM-BIOS 内のルーチンをコールするための関数などが用意されています。

これらの関数を用いる予備知識としての MSX のスロットという概念や、ROM-BIOS や ROM-BASIC などについては「MSX2 テクニカルハンドブック」等をご参照下さい。

5.1.2 BIOS を呼ぶ関数

MSX 本体に内蔵された ROM には数多くの MSX の基本的な入出力を扱うルーチンが含まれています。これらの入出力ルーチンを総称して BIOS (Basic I/O System) と呼びます。

アプリケーションプログラムでは MSX の各機種間での差異を吸収するために入出力はこの BIOS を介して行うことが推奨されています。しかし、これまでは C のプログラムだけで BIOS ルーチン呼び出すためには標準ライブラリ関数の `bios()` や `biosh()` などの汎用的な関数を用いるしかなく C プログラマにとっては BIOS-ROM 内のルーチンを積極的に用いることができる環境ではありませんでした。そこで BIOS ルーチンの中でも必要性の高いものを個別に C の関数呼び出しの形で用いることができるようにしたのが BIOS を呼ぶ関数群です。

BIOS ルーチンでここではサポートされていないものを用いたい場合は上のスロットを管理する関数の `calbio()` などを用いることができます。

5.1.3 その他の関数

その他に乱数を発生する関数、MSX のバージョンを獲得する関数、割り込みの禁止、許可を行う関数などがあります。

5.2 その他の関数の使用法

本章で解説する関数の宣言は `msxbios.h` のなかで宣言されています。ソースファイルの中でインクルードが必要です。また、グラフィック関数を併せて用いる場合で、`glib.h` のインクルードを行っている場合にはこのなかで `msxbios.h` のインクルードがおこなわれているため、格別にこれをインクルードする必要はありません。

コンパイルは通常どおりにおこない、リンク時には `mllib.rel` をリンクしてください。パラメータチェックには `mllib.tco` をパラメータとしてあたえます。コンパイル、パラメータチェック、リンクの手順は `msxc.bat` というバッチにまとめられています。

5.3 その他の関数一覧

スロットを管理する関数

関数名	用途	参照ページ
<code>calbio</code>	ROM-BIOS のアドレスをコールする	62
<code>calbas</code>	//	62
<code>calsub</code>	SUB-ROM のアドレスをコールする	62
<code>calslt</code>	指定のスロットのアドレスをコールする	63
<code>rdslt</code>	スロットのアドレスの内容を返す	63
<code>wrslt</code>	スロットのアドレスにデータを書き込む	63
<code>callx</code>	現在表にでているスロットのアドレスをコールする	63

注意 * の付いた関数は割り込みフラグを変化させません。それ以外の関数は「割り込み許可」で返ってきます。

BIOS を呼ぶ関数

関数名	用途	参照ページ
inifnk	ファンクションキーの内容をデフォルトに戻す	64
disscr	スクリーンの表示の禁止	64
enascr	スクリーンの表示の許可	64
screen	スクリーンモードの変更	64
gicini	PSG の初期化	64
sound	PSG のレジスタにデータを書き込む	64
rdpsg	PSG の指定されたレジスタの内容を返す	65
chsns	キーボードからの入力があるかどうかの判定	65
chget	キーボードからの一文字入力	65
chput	テキストスクリーンへの一文字出力	65
lptout	プリンタへの一文字出力	65
lptstt	プリンタの準備ができているかどうかの判定	65
pinlin	スクリーンエディタを起動する	65
inlin	//	66
breakx	Control-STOP が押されているかどうかの判定	66
beep	ビーブ音の発生	66
cls	スクリーンの表示を消す	66
locate	カーソルの位置を指定する	66
erafnk	ファンクションキーの表示を消す	66
dspfknk	ファンクションキーの表示	66
gtstck	ジョイスティックの状態を調べる	66
gttrig	トリガボタンの状態を調べる	67
gtpad	入出力装置の状態を調べる	67
gtpdl	パドルの内容を読む	68
chgsnd *	サウンドポートのオン/オフによる音の発生	68
snsmat	キーマトリクスの値を見る	69
kilbuf	キーボードバッファの内容を消去する	69

その他

関数名	用途	参照ページ
rnd *	ランダムな整数を返す	69
di	割り込みの禁止	69
ei	割り込みの許可	69
msx2 *	MSX のバージョンを調べる	69
fnkstr *	ファンクションキーの内容を保持しているエリアへのポインタを返す	69

5.4 その他の関数リファレンス

本節の解説で用いられる変数型はヘッダーファイルのなかで次のように定義されています。

TINY char 型に定義 (typedef) されています。
NAT unsigned 型に定義されています。

5.4.1 スロットを管理する関数

VOID calbio(adrs, reg)
NAT adrs ;
REGS *reg ;

ROM-BIOS の指定されたアドレスをコールします。ルーチンをコールする前に値がポインタで指定されたところから Z80 のレジスタにロードされ、ルーチンからリターンする際に Z80 のレジスタの内容が同じ所にコピーされます。ROM-BIOS だけが表に出ます。adrs は 0H から 3FFFH の間でなければなりません。

REGS は msxbios.h の中で以下のように定義されています。

```
typedef struct regs {  
    TINY f;  
    TINY a;  
    NAT bc;  
    NAT de;  
    NAT hl;  
} REGS;
```

VOID calbas(adrs, reg)
NAT adrs ;
REGS *reg ;

calbio() とほぼ同じですが、ROM-BIOS と BASIC-ROM の両方が表に出ます。adrs は 4000H から 7FFFH の間でなければなりません。

VOID calsub(adrs, reg)
NAT adrs ;

REGS *reg ;

SUB-ROM の指定されたアドレスをコールします。ルーチンをコールする前に値がポインタで指定されたところからレジスタにロードされ、ルーチンからリターンする際にレジスタの内容が同じ所にコピーされます。SUB-ROM だけが表に出ます。adrs は 0H から 3FFFH の間でなければなりません。

VOID calslt (slot, adrs, reg)

TINY slot ;

NAT adrs ;

REGS *reg ;

指定されたスロットの指定されたアドレスをコールします。ルーチンをコールする前に値がポインタで指定されたところからレジスタにロードされ、ルーチンからリターンする際にレジスタの内容が同じ所にコピーされます。

指定されたアドレスを含むページだけが表に出ます。

TINY rdslt (slot, adr)

TINY slot ;

NAT adr ;

指定されたスロットの指定されたアドレスの内容を返します。

VOID wrslt (slot, adr, value)

TINY slot ;

NAT adr ;

TINY value ;

指定されたスロットの指定されたアドレスに、データを書き込みます。

VOID callx (adrs, reg)

NAT adrs ;

REGS *reg ;

現在選択されているスロットの指定されたアドレスをコールします。ルーチンをコールする前に値がポインタで指定されたところから Z80 のレジスタにロードされルーチンからリターンする際に Z80 のレジスタの内容が同じところにコピーされます。この関数は通常のアドレスをコールするためのものですが、MSX-DOS の環境では次のような使い方もできます。

```

#define BDOS    0x5
#define GETDATE (NAT)0x2a

getdate(y, m, d)
int    *y, *m, *d;
{
    REGS    r;

    r.bc = GETDATE;
    callx(BDOS, &r);
    *y = r.hl;
    *m = r.de / 256;
    *d = r.de % 256;
}

```

5.4.2 BIOS を呼ぶ関数

VOID inifnk()

ファンクションキーの内容をデフォルトに戻します。

VOID disscr()

スクリーンの表示を禁止します。

VOID enascr()

スクリーンの表示を許可します。

VOID screen(mode)

TINY mode ;

スクリーンモードを変更します。無効なモード (MSX 上で 4 から 8 を指定したなど) に対しての動作は保証されません。

なお、スクリーンモードについては、MSX-BASIC のマニュアルを御参照下さい。

VOID gicini()

PSG を初期化します。

```
VOID    sound(reg, val)
TINY    reg ;
TINY    val ;
```

PSG のレジスタにデータを書き込みます。

reg で PSG レジスタ, val で出力データを指定します。PSG のデータについての詳細は MSX-BASIC の SOUND 文についての解説や、「MSX2 テクニカルハンドブック」等をご覧ください。

```
TINY    rdpsg(reg)
TINY    reg ;
```

PSG の指定されたレジスタの内容を返します。

```
BOOL    chsns()
```

キーボードからの入力があれば 1 を、なければ 0 を返します。

```
char    chget()
```

キーボードから 1 文字入力します

```
VOID    chput(c)
char    c ;
```

テキストスクリーンに 1 文字出力します。

```
BOOL    lptout(c)
char    c ;
```

プリンタに 1 文字出力します。出力に成功すれば 1 を、Control-STOP で中断されれば 0 を返します。

```
BOOL    lptstt()
```

プリンタが READY であるなら 1 を、そうでないなら 0 を返します。

```
char    *pinlin()
```

スクリーンエディタを起動します。キャリッジリターンが押されるとその時点でカーソルがあった行の内容へのポインタを返します。Control-STOP か Control-C が押されると NULL へのポインタが返されます。pinlin を呼んだときのカーソルの X 座標にかかわらず、常に行の先頭からの内容を返します。

char *inlin()

pinlin とほぼ同じですが inlin を呼んだ時のカーソルの X 座標以降の内容しか返しません。ただしカーソルの Y 座標が変わったときは行の先頭からの内容を返します。プロンプトメッセージを伴った入力の際に便利です。

BOOL breakx()

Control-STOP が押されていれば 1 を、そうでなければ 0 を返します。

VOID beep()

ビープ音を発生します。

VOID cls()

スクリーンの表示を消します。テキストスクリーンでもグラフィックスクリーンでも有効です。

VOID locate(csrx, csry)

TINY csrx, csry ;

テキストカーソルの位置を指定します。

VOID erafnk()

ファンクションキーの表示を消します。

VOID dspfnk()

ファンクションキーを表示します。

TINY gtstck(port)

TINY port ;

ジョイスティックの状態を調べます。

port で調べたいジョイスティックポートを指定します。ジョイスティックポートと戻り値は次のとおり。

```
port: 0   カーソルキー
      1   ジョイスティック#1
      2   ジョイスティック#2
```

戻り値： 方向に応じて 0 から 8 が返される。

```
BOOL   gttrig(trig)
```

```
TINY   trig;
```

トリガボタンの状態を調べます。

trig で調べたいトリガボタンの番号を指定します。戻り値はトリガーが押されていれば 1、さもなければ 0 が返される。

トリガボタンの番号と実際の装置との対応は以下のとおり。

通常時	
0	スペースキー
1	ジョイスティック#1のトリガボタン1
2	ジョイスティック#2のトリガボタン1
3	ジョイスティック#1のトリガボタン2
4	ジョイスティック#2のトリガボタン2
マウス使用時	
1	マウス1の左ボタン
2	マウス2の左ボタン
3	マウス1の右ボタン
4	マウス2の右ボタン

```
TINY   gtpad(pad)
```

```
TINY   pad;
```

各種の入出力装置の状態を調べます。本関数で扱うことのできる装置は、タッチパネル、ライトペン、マウス、トラックボールなどです。

pad で調べたい装置の ID を与え、戻り値で状態を知ることができます。装置 ID と戻り値の関係は以下の通りです。

装置 ID	指定される装置	返ってくる情報
0 1 2 3	タッチパネル 1	パネルに触っていれば 0FFH, そうでなければ 00H X 座標 (0 から 255) Y 座標 (0 から 255) ボタンが押されていれば 0FFH, いなければ 00H
4 5 6 7	タッチパネル 2	同上
8 9 10 11	ライトペン	0FFH であればデータ有効, 00H であれば無効 X 座標 (0 から 255) Y 座標 (0 から 255) スイッチが押されていれば 0FFH, いなければ 00H
12 13 14 15	マウス 1 または トラックボール	常に 0FFH を返す (入力要求に用いる) X 方向の変位 (-128 から 127) Y 方向の変位 (-128 から 127) 常に 00H を返す (意味なし)
16 17 18 19	マウス 2 または トラックボール	同上

注意 1 ライトペンの座標 (pad=9,10) とスイッチ (pad=11) の情報は pad=8 として BIOS をコールしたとき同時に読み込まれてバッファリングされますがのときの戻り値が 0FFH の時だけ同時に読み込まれた他の値が有効となります。

注意 2 マウスとトラックボールは自動判別します。

注意 3 マウスやトラックボールの座標を求める場合まず入力要求を行い、その後実際に座標を求めるコールをするという手順を踏みます。このとき 2 つのコールの時間間隔をできるだけ小さくして下さい。入力要求から座標の入力要求までの間が必要以上に開くと得られたデータは保証されません。

TINY gtpdl (pdl)

TINY pdl ;

指定されたパドルの内容を読みます。

VOID chgsnd (onoff)

BOOL onoff ;

サウンドポートのビットのオン、オフを行うことにより音を発生します。data が 0 のときオフ、0 以外の時オンとなります。

TINY `snsmat(row)`

TINY `row` ;

キーマトリクスの `row` で指定した行の値を読みます。

戻り値の押されているキーに対応するビットが0になります。MSXのキーマトリクスについては「MSX2 テクニカルハンドブック」をご覧ください。

VOID `kilbuf()`

キーボードバッファの内容をすべてクリアします。

5.4.3 その他

NAT `rnd(range)`

NAT `range` ;

0 から `range-1` までの範囲でランダムな整数を返します。

VOID `di()`

割り込みを禁止します。

VOID `ei()`

割り込みを許可します。

BOOL `msx2()`

MSXのバージョンを調べます。使用中のマシンがMSX2ならば真、MSXならば偽が返されます。本関数は、`msxbios.h`の中で定義されている`#define`マクロです。

char `*fnkstr(fkey)`

TINY `fkey` ;

指定されたファンクションキーの内容を保持しているエリアへのポインタを返します。`fkey`は1から10まででなければなりません。これは実際には`msxbios.h`で定義されているマクロです。

第6章 カーソル制御関数パッケージ

6.1 カーソル制御関数(MSX-CURSES)の概要

CURSES が UNIX のテキスト画面を扱うためのライブラリ関数群であることは既に述べました。MSX-CURSES は、UNIX の CURSES の中心的な関数についてはほぼサポートしており、根本的な原理も同じであると言って良いでしょう。ただし、個々の関数で若干仕様の異なるものがあり、その点注意が必要です。

しかし、UNIX の CURSES を使った経験のある方でしたら、6.4 の関数リファレンスをご覧になるだけで、UNIX の CURSES との違いを把握し、すぐにでもプログラムが可能でしょう。ただ、そうした方はごくまれだと思われますし、MSX-CURSES をより有効にお使いいただくためにはある程度の CURSES の原理に関する理解は必要でしょう。そこで、本節ではまずはじめに、MSX-CURSES の基本的な概念とその原理について解説した後、MSX-CURSES を構成する関数を総括的に見渡してみましよう。

6.1.1 MSX-CURSES の画面更新最適化の原理

A)用語の解説

MSX-CURSES の原理について解説するに先立ち、解説中で統一的に用いられる概念を表す用語の解説を行います。

物理画面 実際のテレビやモニタの画面のこと。実画面や、コンソール画面と呼ぶ場合もあります。

ウィンドウ 物理画面のある領域に現在どのような表示がなされているか、あるいは、どのような表示がなされるべきかというイメージを常に保存する一種のバッファ。MSX-CURSES ではこのバッファ領域と画面上の位置などの情報を含め、WINDOW 型の構造体としてウィンドウを扱います。

論理カーソル 各ウィンドウはそれぞれの構造体にウィンドウバッファ上の仮想的なカーソル位置の情報も保存しています。ウィンドウへの書き込みはこの論理カーソル位置に行われ、書き込み終了と同時に最後に書き込んだ位置の1つ後ろに移動します。

スクリーン 物理画面いっぱい(フルスクリーン)のウィンドウを特にスクリーンと呼びます。

論理画面 スクリーン、ウィンドウを総称して論理画面と呼びます。

B)最適化の原理

MSX-CURSES の目的は、画面に文字を表示する際に、できるだけ効率よく画面の書換えを行なうということに尽きます。このことを画面更新の最適化と呼んでいます。

MSX-CURSES では UNIX の CURSES と同様にスタンダードスクリーン(stdscr)とカレントスクリーン(curscr)という二つのスクリーンを用いることにより画面更新の最適化を行います。二つのスクリーンはメモリ上にバッファ領域として、それぞれ(画面幅×行数)バイト(すなわち、物理画面の大きさと同じ)の大きさを持ちます。二つのスクリーンの用途はだいたい以下のとおりです。

- | | |
|--------------------|--|
| スタンダードスクリーン | MSX-CURSES のスクリーンへの書き込みルーチン(addch, addstr など)は物理画面にはなくこのスタンダードスクリーンに対し書き込みを行う。書き込みを行う座標は論理カーソル位置により決定する。 |
| カレントスクリーン | 実際の画面のイメージを常に保存する。スタンダードスクリーンやその他のウィンドウ、スクリーンに対して行われた更新を実画面に反映させる(これを画面のリフレッシュという)際にカレントスクリーンの内容(すなわち現在の画面表示の状態)と比較し変化のあった部分だけを実画面に表示することになる。同時にカレントスクリーンの更新もおこなわれ次回の書き込みに備える。 |

もう少し具体的に述べてみましょう。

まず、MSX-CURSES の関数群を使用する前提として CURSES の初期化ルーチン呼び出しなくてはなりません。初期化ルーチン(initscr())では、上記2つのスクリーンバッファのためのメモリが確保され、両スクリーンは20H(スペースコード)で埋められます。論理カーソル位置はホームポジションに初期化されます。

ここでスタンダードスクリーンへの書き込みルーチンにより、例えば

```
addstr("Hello!!!");
```

というようにすると、スタンダードスクリーンの最上行の先頭から"Hello!!!"という文字列が書き込まれ論理カーソル位置が最後の"! "の直後に移動します。この時点では実際の実画面には何も表示されません。スタンダードスクリーンに行われた書き込みを実画面に反映するにはリフレッシュを行わなければなりません。上の関数呼び出しに続いて、

```
refresh();
```

とすることにより、はじめて実際のスクリーン画面に Hello!!! と表示されます。

refresh()では、スタンダードスクリーンとカレントスクリーンを頭から1文字ずつ比較してい

き、一致しなかった文字だけをカレントスクリーンにコピーし、同時に実画面にも表示するので、この例では、カレントスクリーンはすべてスペースで埋められていますから(画面がクリアされた直後の状態)Hello!!!という文字列だけが表示されます。さらに引続き、

```
move(0, 0);
addstr("Yellow!!");
refresh();
```

とするとどうでしょう。まず move によって論理カーソルの位置をホームに戻してからスタンダードスクリーンへの書き込みを行なっています。これによりスタンダードスクリーンには”Yellow!!”の文字列が”Hello!!!”に上書きされました。

ここで refresh() の行う動作を追ってみます。

まず、”Y”と”H”が比較され、一致しないため実画面の左上はしに”Y”が表示されカレントスクリーンにも”H”にかわって”Y”のキャラクターコードが書かれます。スタンダードスクリーンもカレントスクリーンも次の文字は”e”ですからなにもしません。”o”までは一致し続けますから同様です。”w”でようやく2文字目の表示が行われます。それ以降は画面の最後まで(最後の”!”以降はどちらもスペースで埋まっているから一致する)実際の画面更新は起こらずリフレッシュを終了します。以上で画面更新最適化のアルゴリズムの基本的な部分が理解いただけたいと思います。

MSX-CURSES ではこのほかにも画面の書換えをより効率よく行うために多くの内部的な工夫がなされています。例えば、refresh() は呼ばれる毎に全ての文字比較をおこなうのではなく変更のあった行の変更された範囲に限って比較をおこなうといった具合です。

おおよそ CURSES などといった UNIX 流のライブラリに、はじめて接する方にとっては以上の説明は甚だ不足しているかと思われます。MSX-CURSES の実用性についても疑いをもたれる方も多いでしょう。しかし、MSX-CURSES の関数群を、その特性を理解し活用することによって UNIX の vi のようなエディタを最小限の労力で作成することも十分に可能であると考えます。さらに MSX-CURSES の画面更新の実効性についても、付属のサンプルプログラムを実際に動かすことで実感していただけたと思います。

6.1.2 ウィンドウ

CURSES では、画面上の任意の位置に任意の大きさのウィンドウを開き各ウィンドウについてその内容の更新を行うことができます。addch()には waddch(), move()には wmove()のようにスタンダードスクリーンへの書き込み関数の名前に”w”をつけたものが同一機能のウィンドウを扱う関数です。

スタンダードスクリーンやカレントスクリーンもフルスクリーン(画面いっぱい)のウィンドウと理解することができ、データ構造もユーザが newwin()関数により開いたウィンドウとまったく同じです。したがってスタンダードスクリーンを扱う関数の大半はウィンドウを扱う関数を、stdscr(スタンダードスクリーン)というウィンドウを引き数として呼び出しているに過ぎません。

たとえば `addch(ch)` は `waddch(stdscr, ch)` という呼び出しをおこないます。

さらに、リフレッシュについても `wrefresh()` という関数があり、ウィンドウ毎に行うことができるのです。

6.1.3 MSX-CURSES 関数パッケージの概要

ここでは、MSX-CURSES の関数群を機能の面から下のようなグループにわけ、簡単に紹介していきます。

- ・初期化、オプションモードの設定
- ・論理画面への書き込み
- ・コンソール画面の更新
- ・論理画面からの入力
- ・コンソールからの入力

A) 初期化、オプションモードの設定

MSX-CURSES の関数を用いるためには前述したようなバッファ等の初期化を行わなければなりません。さらに画面更新時に画面をはみ出したときの処理をどうするか、などのモード指定が可能です。

B) 論理画面への書き込み関数

MSX-CURSES においては、前述のように、ウィンドウやスタンダードスクリーンへの書き込みとリフレッシュの繰り返しにより画面への文字の出力が行われます。ウィンドウやスタンダードスクリーンへの書き込みルーチンには、単に文字や文字列を書き込むだけでなく、行や文字の削除および挿入、行のスクロール、標準関数の `printf()` に匹敵するような書式付きの書き込みを行うルーチンなどもあります。

C) コンソール画面への出力を行う関数

コンソール画面への文字出力を行う関数は、前述の `refresh()` と `wrefresh()` です。この関数はスタンダードスクリーンやウィンドウについて設定されたオプションモードに従って、つねに最も効率のよい画面更新を行います。リフレッシュでは画面を更新すると同時に、カレントスクリーンの更新も行います。

D) 論理画面からの入力関数

MSX-CURSES にはウィンドウやスクリーンから情報を得るための関数も用意されています。

すなわち、現在の論理カーソルの位置や、カーソル位置にある文字を知ることができます。これらの関数を用いれば、例えば画面にメニューの一覧が表示され、実行したい機能をメニュー表示にカーソルキーを合わせてリターンキーをおすことにより任意に選べるなどの、ピジュアルなコマンドを作成することもできるでしょう。

E) コンソールからの入力ルーチン関数

コンソールからの入力では、キーボードから入力された文字を画面にエコーバックするかどうかを設定することができます。

コンソール入力関数としては、コンソールのキーボードからコンソールモードに従って文字や、文字列の入力を行う関数があります。

6.2 MSX-CURSES 関数パッケージの使用法

MSX-CURSES の各関数を用いる際には、ソースプログラム内で `curses.h` をインクルードする必要があります。

ライブラリファイルは `mllib.rel` で、これをリンク時にユーザプログラムより前におきます。

FPC によるパラメータチェックの際には `mllib.tco` をチェックの対照としなくてはなりません。以上、コンパイルからリンクまでの手順は `msxc.bat` というパッチファイルで提供されています。

6.3 MSX-CURSES 関数一覧

初期化処理、オプションモードの設定など

関数名	用途	参照ページ
<code>initscr</code>	MSX-CURSES を用いるための初期化を行う関数	78
<code>newwin</code>	新たにウィンドウを開く	78
<code>subwin</code>	ウィンドウ内に新しいウィンドウを開く	78
<code>endwin</code>	CURSES を用いたプログラムを終了するときと呼ぶ関数	79
<code>mvwin</code>	ウィンドウのホームポジションの座標変更	79
<code>delwin</code>	ウィンドウを閉じる	79
<code>scrollok</code>	カーソルがスクリーンからはみ出した場合の処理決定	79

コンソール出力関数

関数名	用途	参照ページ
refresh	実画面の更新を行う	80
wrefresh	//	80

論理画面への書き込み関数

関数名	用途	参照ページ
move	論理カーソル位置の移動	80
wmove	//	80
addch	現在の論理カーソル位置に1文字書き込む	80
waddch	//	80
mvaddch	論理カーソルを任意の位置に移動後1文字書き込む	81
mvwaddch	//	81
addstr	現在の論理カーソルの位置に文字列を書き込む	81
waddstr	//	81
mvaddstr	論理カーソルを移動後文字列を書き込む	81
mvwaddstr	//	82
box	ウィンドウを指定したキャラクタで囲む	82
overwrite	ウィンドウから別のウィンドウにバッファの内容をコピーする	82
touchwin	画面更新の最適化のための情報を無効にし次のリフレッシュ時にウィンドウの全面に比較を行う	82
erase	論理画面をスペースでうめる	83
werase	//	83
clear	ウィンドウをクリアする	83
wclear	//	83
clearok	クリアフラグのオン、オフを行う	83
clrtoebot	論理カーソル位置から画面の右端までをクリアする	84
wclrtoebot	//	84
clrtoeol	論理カーソル位置からその行末までをクリアする	84
wclrtoeol	//	84
delch	論理カーソル位置の1文字を削除する	84
wdelch	//	84
mvdelch	論理カーソルを移動後1文字を削除する	84
mvwdelch	//	84
deleteln	論理カーソル位置の1行を削除する	84
wdeleteln	//	85
insertln	論理カーソル位置に空行1行を挿入する	85
winsertln	//	85

関数名	用途	参照ページ
scroll	ウィンドウ内で1行スクロールを行う	85
insch	カーソル位置に1文字挿入	85
winsch	〃	85
mvwinsch	論理カーソルを移動後その位置に1文字挿入	85
mvwinsch	〃	85
printw	ウィンドウへの書式付き書き込み	86
wprintw	〃	86
mvprintw	論理カーソルを移動後の書式付き書き込み	86
mvprintw	〃	86

論理画面からの入力関数

関数名	用途	参照ページ
inch	論理カーソル位置の文字を返す	86
winch	〃	86
mvinch	論理カーソルを移動後カーソル位置の1文字を返す	86
mvwinch	〃	86
getyx	現在の論理カーソル位置を返す	87

コンソールからの入力関数

関数名	用途	参照ページ
echo	コンソールキーボードから入力を行う際に入力された文字を画面に表示するかどうかの決定をする	87
noecho	〃	87
getch	コンソールキーボードからウィンドウの論理カーソル位置に1文字読み込む	87
wgetch	〃	87
mvgetch	論理カーソル移動後 getch(), wgetch() によりコンソールから1文字を読み込む	87
mvwgetch	〃	87
getstr	コンソールキーボードからウィンドウのカーソル位置に1行読み込む	87
wgetstr	〃	88
mvgetstr	論理カーソルの移動後コンソールからウィンドウに1行読み込む	88
mvwgetstr	〃	88

注意 関数名で、先頭に 'w' がつかないものと、'w' のつくものがある場合、前者がスタンダードスクリーンを扱う関数で、後者が同一機能で、スタンダードスクリーン以外のスクリーンやウィンドウを扱うものです。

6.4 MSX-CURSES 関数リファレンス

6.4.1 MSX-CURSES のシステム変数, データ構造

MSX-CURSES では以下のような変数名が予約されています。ユーザはこれらの名前をユーザプログラム内で再定義することはできません。またこれらの変数の内容を変更する場合でも MSX-CURSES の関数を介してのみ行うことができます。

A) システム変数

TINY	LINES;	コンソール画面の行数, <code>initscr()</code> により初期化される。
TINY	COLS;	コンソール画面のカラム数, <code>initscr()</code> により初期化される。
TINY	<code>_tty;</code>	コンソール入力時の入力モードのフラグ。
WINDOW	<code>*stdscr;</code>	スタンダードスクリーンへのポインタ
WINDOW	<code>*curscr;</code>	カレントスクリーンへのポインタ

B) ウィンドウの構造

ウィンドウのデータののための構造体”WINDOW”は `curses.h` のなかで次のように型定義されています。通常はこれらの構造体の内容を直接変更, 参照する必要はありません。

```
typedef struct {
    TINY          _cury, _curx;           ☐論理カーソル位置
    TINY          _maxx, _maxy;          ☐ウィンドウの大きさ
    int           _begx, _begy;          ☐ホームポジションの座標
    int           _flags;                ☐最適化に用いるフラグ
    BOOL         _clear;                 ☐クリアフラグ
    BOOL         _leave;                 ☐未使用
    BOOL         _scroll;                ☐スクロールフラグ
    INDEXLIN     *_index;                ☐各行毎の情報
} WINDOW;
```

```
typedef struct {
    char          *_y;                    ☐行バッファ
    int           _firstch;               ☐変更のあった最初のカラム
    int           _lastch;                ☐最後のカラム
    int           _ins_del;               ☐ハードウェアスクロールの
                                         ためのフラグ
} INDEXLIN;
```

6.4.2 初期化処理, オプションモードの設定など

WINDOW *initscr()

MSX-CURSES の各ルーチンを用いる場合には必ず呼ばなければならない初期化のための関数です。

stdscr(スタンダードスクリーン)と curscr(カレントスクリーン)の2つのウィンドウ領域を確保し、これを初期化します。戻り値は stdscr へのポインタが返されますが、領域の確保に失敗したときは NULL を返します。

WINDOW *newwin(lines,cols,begin_y,begin_x)

int lines,cols,begin_y,begin_x ;

新たにウィンドウを開設します。lines,cols でウィンドウの行数,カラム数を,begin_y,begin_x でウィンドウのホームポジションの位置を指定します。

戻り値は WINDOW 型の構造体へのポインタが返され、以後、これを用いてウィンドウへアクセスします。また、ウィンドウのためのバッファ領域が確保できなかった場合 NULL が返されず。

もし、lines,cols に 0 を指定した場合

```
lines = LINES(物理画面の行数) - begin_y;
cols = COLS(物理画面のカラム数) - begin_x;
```

となります。したがってフルスクリーンのウィンドウ(すなわちスクリーン)を開きたい場合は

WINDOW *fullwin;

```
fullwin = newwin(0, 0, 0, 0);
```

として下さい。

WINDOW *subwin(win,lines,cols,begin_y,begin_x)

WINDOW *win ;

int lines,cols,begin_y,begin_x ;

ウィンドウの中に、新しいウィンドウを開きます。newwin と異なる点は、どちらかのウィンドウに対して行われた書き込みが他方にも反映されることです。

win で指定したウィンドウとバッファエリアを共有するウィンドウが開かれるわけです。

サブウィンドウの位置と大きさは、win で指定する外側のウィンドウの中に入るように指定しなくてはなりません。サブウィンドウのためのメモリが足りなかったり、サブウィンドウの位置や大きさが不適当な場合 NULL が返されます。

```
VOID    endwin()
```

CURSES を用いたプログラムを終了するときと呼ぶルーチンです。スクリーンデータのためのバッファ領域を解放します。

```
STATUS mvwin(win,y,x)
```

```
WINDOW *win ;
```

```
int     y,x ;
```

ウィンドウのホームポジションの座標を(x,y)に移動します。もし(x,y)がコンソール画面の端をはみ出してしまう場合、本関数は ERROR を返し、移動は行われません。

```
VOID    delwin(win)
```

```
WINDOW *win ;
```

win で指定したウィンドウを閉じます。ウィンドウのためのバッファは解放されます。delwin() でバッファを解放しても、画面表示に影響を与えません。

このルーチンでサブウィンドウ(subwin で開設したウィンドウ)を閉じても、ウィンドウのバッファ領域の解放は行われません。通常、外側のウィンドウを解放した後、サブウィンドウを閉じることにより、バッファ領域の解放を行います。

```
VOID    scrollok(win,boolf)
```

```
WINDOW *win ;
```

```
BOOL    boolf ;
```

論理カーソルが、ウィンドウやスクリーンの最下行からの改行や、最終行の最終桁での文字入力によりスクリーンからはみ出した場合の処理を決定します。本ルーチンで boolf に TRUE をすると、上記の場合、強制的にウィンドウ内で 1 行スクロールが起こります。

逆に、boolf を FALSE で実行するとウィンドウのスクロールが禁止され、ウィンドウの右下隅に文字は出力されなくなります。

6.4.3 コンソール出カルーチン

STATUS refresh()

STATUS wrefresh(win)

WINDOW *win ;

実画面への表示を行います。スクリーンやウィンドウへの書き込みをおこなった後に、リフレッシュすることによりはじめて実画面への表示が行われます。同時に、カレントスクリーンの更新も行います。

refresh()はスタンダードスクリーン、wrefresh()はその他のスクリーンやウィンドウの更新に用います。

6.4.4 論理画面への書き込みルーチン

STATUS move(y,x)

int y,x ;

STATUS wmove(win,y,x)

WINDOW *win ;

int y,x ;

論理カーソル位置を移動します。

move()では、スタンダードスクリーン”stdscr”上の論理カーソルを(x,y)で指定した位置に移動します。y、xが画面の範囲外の時はERRORが返されます。

その他のスクリーンやウィンドウの論理カーソルはwmove()で移動します。

スクリーンやウィンドウへの書き込みは常に論理カーソル位置に対して行われます。

STATUS addch(ch)

char ch ;

STATUS waddch(win,ch)

WINDOW *win ;

char ch ;

スクリーンやウィンドウの現在の論理カーソル位置に1文字を書き込みます。書き込むキャラ

クタを `ch` で与えます。

`addch()` はスタンダードスクリーンに対しての書き込み、`waddch()` はそれ以外のスクリーンやウィンドウに対しての書き込みを行います。 `waddch()` で書き込みを行うウィンドウは `win` で指定します。

文字を書き込むことにより、ウィンドウをはみ出す場合、スクロールが禁止されていると `ERROR` となります。それ以外の場合は自動的にスクロールが起こります。(但し、強制的にリフレッシュが起こることはありません。)

STATUS `mvaddch(y,x,ch)`

int `y,x;`

char `ch;`

STATUS `mvwaddch(win,y,x,ch)`

WINDOW `*win;`

int `y,x;`

char `ch;`

論理カーソルを任意の位置に移動後、1文字を書き込みます。

論理カーソルの移動先を (x,y) 、書き込むキャラクタを `ch` で与えます。論理カーソルの移動先が不適當であったり、スクロールが禁止されているウィンドウで、スクロールが起こるような書き込みが行われた場合 `ERROR` となります。

STATUS `addstr(str)`

char `*str;`

STATUS `waddstr(win,str)`

WINDOW `*win;`

char `*str;`

現在の論理カーソルのある位置に文字列を書き込みます。

書き込む文字列へのポインタを引き数として渡します。`addstr()` や `waddstr()` は `waddch()` を文字数分だけ呼んでいるので `ERROR` となる条件は `waddch()` と同じです。

STATUS `mvaddstr(y,x,str)`

int `y,x;`

char `*str;`

STATUS `mvwaddstr(win,y,x,str)`

WINDOW `*win;`

```
int      y,x ;
char     *str ;
```

論理カーソルを移動後、文字列の書き込みをおこないます

(x,y)で論理カーソルの移動先, str は書き込む文字列へのポインタを渡します. ERROR の返される条件は wmove, waddstr の場合と同じです.

```
STATUS  box(win,vert,hor)
WINDOW  *win ;
char     vert,hor ;
```

ウインドウ内を指定したキャラクタで箱状に囲みます.

vert で垂直枠のキャラクタ, hor で水平枠のキャラクタを与えます.

本命令はウインドウに枠をつけたい場合に用いられます. 通常, box()により枠取りをしたウインドウの内部に枠が壊されないような大きさのサブウインドウを開き, 書き込みはサブウインドウに対して行うなどにより用います.

呼び出し例

```
win = newwin(0, 0, 0, 0);
sub = subwin(win, (int)LINES - 2, (int)COLS - 2, 1, 1);
box(win, '|', '-');
wrtwin(sub);          /* 書き込みを行う関数 */
```

```
VOID     overwrite(win1,win2)
WINDOW  *win1, *win2 ;
```

ウインドウから別のウインドウに対して, バッファの内容のコピーを行います.

win1 から win2 にウインドウの重なった部分のみがコピーされます. 2つのウインドウに重なった部分のまったくない場合にはコピーは行われません.

```
STATUS  touchwin(win)
WINDOW  *win ;
```

画面更新の最適化のための情報を無効にし, 次回のリフレッシュ時にウインドウ全面の比較を行うようにします.

CURSES では一部または全部が重なったウインドウを交互に更新した場合, 画面更新の最適化処理が逆に災いして正常に更新されません. touchwin()により最適化処理の一部を無効にすることにより重なりあったウインドウを交互に更新する際にも, 正常な更新を行うことができます.

具体的には、更新(リフレッシュ)したいウィンドウが直前に更新したウィンドウと異なり、かつ2つのウィンドウに重なった領域がある場合に今回更新すべきウィンドウに対して用います。

呼び出し例

```
STATUS func(win) /* 重なりあったウィンドウを更新する関数 */
WINDOW *win;
{
    static WINDOW *lastwin = NULL;

    if(lastwin != win) {
        lastwin = win;
        touchwin(win);
    }
    return(wrefresh(win));
}
```

```
VOID erase()
```

```
VOID werase(win)
```

```
WINDOW *win;
```

論理画面をスペースで埋めます。erase()でスクリーンをクリアしても、クリアフラグはセットされません。

```
STATUS clear()
```

```
STATUS wclear(win)
```

```
WINDOW *win;
```

erase()を呼び出した後、次のclearok()を呼び出しクリアフラグをオンにします。

wclear()でカレントスクリーンをクリアすると、スタンダードスクリーンがスペースクリアされ、強制的にリフレッシュがおこり画面がクリアされます。本関数はERRORを返しません。(常にOKが返される)

```
VOID clearok(scr, boolf)
```

```
WINDOW *scr;
```

```
BOOL boolf;
```

クリアフラグのオン、オフを行います。

クリアフラグがオンのスクリーンにたいしてリフレッシュを行うと、物理画面をクリアした後、

リフレッシュをおこないます。この際、クリアフラグは自動的にリセット(オフ)されます。
クリアフラグのオン、オフはスクリーンに対してのみ有効です。

VOID clrrobot()

VOID wclrrobot(win)

WINDOW *win;

論理カーソルの位置から論理画面の右端までをスペースでクリアします。

VOID clrtoeol()

VOID wclrtoeol(win)

WINDOW *win;

論理カーソルの位置からその行末までをスペースでクリアします。

STATUS delch()

STATUS wdelch(win)

WINDOW *win;

論理カーソル位置の1文字を削除します。

カーソル位置より右にある文字はすべて1文字ずつ左に詰められます。本関数はERRORを返しません。(常にOKが返される)

STATUS mvdelch(y,x)

int y,x;

STATUS mvwdelch(win,y,x)

WINDOW *win;

int y,x;

論理カーソルを移動後1文字を削除します。ERRORの返される条件は、wmove()と同じです。

VOID deleteln()

VOID wdeleteln(win)

WINDOW *win ;

論理カーソル位置の 1 行を削除します。

以降の行が 1 行ずつ上に詰められ最下行は空行となります。

VOID insertln()

VOID winsertln(win)

WINDOW *win ;

論理カーソル位置に空行 1 行を挿入します。

STATUS scroll(win)

WINDOW *win ;

ウインドウ内で 1 行スクロールをおこないます。ウインドウの最下行は空行になります。

STATUS insch(ch)

char ch ;

STATUS winsch(win,ch)

WINDOW *win ;

char ch ;

論理カーソル位置に 1 文字を挿入します。

この際画面からはみ出した文字のデータは失われてしまいます。本関数は ERROR を返しません。(常に OK が返される)

STATUS mvinsch(y,x,ch)

int y,x ;

char ch ;

STATUS mvwinsch(win,y,x,ch)

WINDOW *win ;

int y,x ;

char ch ;

論理カーソルを移動した後その位置に 1 文字を挿入します。ERROR が返される条件は、wmove()と同じです。

STATUS printf(format, arg1, arg2, ...)
char *format ;

STATUS wprintf(win, format, arg1, arg2, ...)
WINDOW *win ;
char *format ;

STATUS mvprintf(y, x, format, arg1, arg2, ...)
int y, x ;
char *format ;

STATUS mvwprintf(win, y, x, format, arg1, arg2, ...)
WINDOW *win ;
int y, x ;
char *format ;

6.4.5 論理画面からの入力

char inch()

char winch(win)
WINDOW *win ;

論理カーソル位置の文字を返します。

関数の戻り値として論理カーソル位置の 1 文字が返されます。

char mvinch(y, x)
int y, x ;

char mvwinch(win, y, x)
WINDOW *win ;
int y, x ;

論理カーソル移動後の論理カーソル位置の 1 文字を返します。

カーソルの移動位置が不適当な場合 ERROR が返されます。

```

VOID    getyx(win,y,x)
WINDOW *win;
int     y,x;

```

指定されたウィンドウの現在の論理カーソル位置を y,x に代入します。
このルーチンは `curses.h` の中でマクロとして定義されています。

6.4.6 コンソールからの入力をおこなう関数

```

VOID    echo()

VOID    noecho()

```

`getch()` や `getstr()` などによりコンソールのキーボードからの入力を行う際に入力された文字を画面にエコーバックするかどうかを決定します。`echo()` でエコーオン、`noecho()` でエコーオフです。初期状態はエコーオンです。

```

char    getch()

char    wgetch(win)
WINDOW *win;

```

コンソールキーボードからウィンドウの論理カーソル位置に 1 文字読み込みます。`getch()` ではスタンダードスクリーンに、`wgetch()` では指定したウィンドウに、それぞれコンソールからコンソールモードに従って、1 文字を読み込みます。

' $\backslash r$ '(キャリッジリターン)、は' $\backslash n$ '(ニューライン)にマッピングされます。

```

char    mvgetch(y,x)
int     y,x;

char    mvwgetch(win,y,x)
WINDOW *win;
int     y,x;

```

論理カーソルを移動後、`getch()`、`wgetch()` によりコンソールから 1 文字を読み込みます。カーソルの移動先が不適切な場合(ウィンドウをはみ出す場合) `ERROR` が返されます。

STATUS getstr(str)

char *str ;

STATUS wgetstr(win, str)

WINDOW *win ;

char *str ;

コンソールキーボードからウィンドウの論理カーソル位置に、1行を読み込みます。'¥n'が入力されるまで読み込まれ、str でポイントされる文字配列に格納されますが、この際'¥n'は除かれ代わりに NUL('¥0')がつけられます。本関数は ERROR を返しません。(常に OK が返される)

STATUS mvgetstr(y, x, str) ;

int y, x ;

char *str ;

STATUS mvwgetstr(win, y, x, str)

WINDOW *win ;

int y, x ;

char *str ;

論理カーソルの移動後、コンソールからウィンドウに対して、1行を読み込みます。
カーソルの移動先が不適切な場合(ウィンドウをはみ出す場合)ERROR が返されます。


付録 サンプルプログラム集

A) 数値演算パッケージを用いたプログラム”den”について

1 概要

den は MSX-C ライブラリパッケージに含まれる数値演算モジュールを使用した整数型 16 進電卓プログラムです(10 進計算も可)。通常の電卓のようにメモリへの代入、参照やカッコ計算を含む優先順位を考慮した計算ができます。

2 den の使用例

ここでは簡単な使用例を示します。電卓としての詳細は次頁の「3 仕様」を参照して下さい。den の起動は MSX-DOS プロンプトが出ている時、den  と入力するだけです。すると、数値モードの文字(D,U,X,O のどれか)が左端に表示され、その右に初期値 0 が表示されます。

```
D          0
```

数値を入力すると 0 が入力にあわせて変化していきます。1350 と入力すると表示は、

```
D          1350
```

となります。以下入力とそれに対する表示を示します。

入力	表示
+	D 1350
23	D 23
=	D 1373
*5=	D 6865
M0	D 6865
/23=	D 298
+m0	D 6865
=	D 7163
%7=	D 2
2*(D 0
3+5	D 5
)	D 8
=	D 16

このようにほとんど普通の電卓のように動作します。ただし、キーの割り付けがC言語風なので注意して下さい。

3 仕様

ここでは、den の仕様と、キーの割り付けを説明します。キーの割り付けに使われる文字は大文字と小文字の区別がされます。

3.1 数値モード(基数)

den の数値モードは、符号つき 10 進と符号無し 10 進、16 進、8 進の 4 種類があります。符号つきと符号なしの違いは商、剰余を求める場合と、算術右シフトを行う場合にのみ現れます。その他の場合に違いは出てきません。基数の違いは入力と出力に指定された基数を使って行われるだけにすぎません。数値モードの変更は次のようになります。

```
符号つき 10 進   : D
符号無し 10 進   : U
符号無し 16 進   : X
符号無し 8 進    : O
```

数値モードは常に数値表示欄の左端に D,U,X,O の文字で表示されています。

3.2 置数

置数(数字を入力すること)は数値の上位の桁から 1 桁ずつ入力します。符号つきで負数を入力する場合は、0 以外の数値を入力してから符号を反転させる文字_(アンダースコア)で負数にします。入力を間違えた場合は BS(BACKSPACE)キーを使用することによって 1 桁ずつ取り消すことができます。また、入力された数値が使用できる数値の範囲を超えた場合には、上位の桁がなくなります。置数時有効な文字は基数によって変わってきます。使用できる文字を数値モード別に示します。

```
符号つき 10 進   : 0 1 2 3 4 5 6 7 8 9 _
符号無し 10 進   : 0 1 2 3 4 5 6 7 8 9
符号無し 16 進   : 0 1 2 3 4 5 6 7 8 9 a b c d e f
符号無し 8 進    : 0 1 2 3 4 5 6 7
```

3.3 使用できる数値の範囲

den で扱われる数値はすべて SLONG 型(32bit)となっています。それぞれの数値モードの表現で使用できる範囲を示します。

符号つき 10 進	:	-2147483648 から 2147483647
符号無し 10 進	:	0 から 4294967295
符号無し 16 進	:	0 から FFFFFFFF
符号無し 8 進	:	0 から 3777777777

3.4 オールクリア, クリア, =

オールクリアは、計算を始める前に行うことで、それまでの計算に影響を受けることなく実行することができます(表示レジスタ、数値スタック、演算子スタックをクリアします)。またクリアは、置数に大幅な間違いが見つかったときに使用することで、キー入力数を少なくすることができます(表示レジスタのみをクリアします)。=はそれまでの入力をすべて計算して、結果を表示するのに使います。

A	オールクリア
C	クリア
=	=
リターンキー	=と同じ

3.5 二項演算子およびカッコ

二項演算子には加減乗除・剰余(+ - * / %), 論理積・論理和・排他的論理和(& | ^)があります。おのおのの演算子には優先順位がついており、優先順位の高い演算から実行されます。この優先順位を変更するためには、優先順位を上げたいところでカッコを使用します。優先順位の順位表を示します。順位は小さい方が優先度が高く、優先順位が同じものについては式を入れた順に評価されます。

1 ()	(カッコ)
2 * / %	(乗算, 除算, 剰余)
3 + -	(加算, 減算)
4 &	(論理積)
5	(論理和)
6 ^	(排他的論理和)

3.6 単項演算子

演算子には二項演算子の他に単項演算子があります。これは演算子のキーを押すと、表示されている数値に演算をし、演算結果を表示するものです。ですから優先順位はありません(言いかえると最高位の優先度)。単項演算子には否定(not), シフト・ローテイトがあります。

~	(否定 ビットごとの反転)
<	(左シフト)
>	(右シフト 符号ありの場合は算術右シフト)

[(左ローテイト)
]	(右ローテイト)
{	(キャリーを含んでの左ローテイト)
}	(キャリーを含んでの右ローテイト)

シフトとローテイトに関しては1回のキー操作で1ビットずつ移動します。また、置数時に使われる符号反転()もこの単項演算子のひとつと考えられます。

3.7 メモリ機能

メモリ機能は4種類あり、代入・参照・加算・減算となっております。それぞれ電卓的表現で、Min・MR・M+・M-となります。メモリは番号0から9までの10個があります。メモリを使用する場合にはメモリ機能の操作(下記参照)に1桁のメモリ番号(0-9)を指定します。メモリ機能は次のようになります。(＃はメモリ番号)


M#	メモリへの代入(Min)
m#	メモリの参照(MR)
M+#	メモリへの加算(M+)
M-#	メモリからの減算(M-)

M+・M-に、イコール機能はありません。また、数値モードが変更された場合には現在のモードで表示されます(符号ありと符号なしが混在している場合には注意が必要です)。

4 コンパイル方法

den.com のソースファイルは den.c です。

コンパイルはCライブラリに含まれる msxc.bat にコンパイルスイッチをつけてコンパイルします。

A>msxc den -r5:6:2 -r2000 

5 注意点

5.1 0による除算・剰余

サンプルとして含まれている den は 0 による除算・剰余のエラーチェックをまったく行っていません。ですから、カッコの多い複雑な式を計算する場合には除数にご注意下さい。

5.2 カッコの機能について

den はカッコつきの計算ができますが、そのもとの機能は少し特殊です。左カッコ '(' が入力されるとそれ以降の演算の優先度を上げ、右カッコ ')' が入力されるとそれに対応する左カッコから右カッコまでの計算を行います。ですから、演算子を忘れて左カッコを入力しても den はそれをカッコとして受け付けます。実際の例を示すと、 $2 \times 3(+2) =$ と入力すると den では $2 \times (3+2) =$ と入力されたものと認識します。

B) グラフィック関数を用いたプログラム " gcal" について

1 概要

サンプルプログラム " gcal" は、グラフィック関数を使って2か月分のカレンダーとバイオリズムをグラフィック画面に描くというプログラムです。マシンが MSX の場合は自動的に MSX 用(スクリーンモード 2)が、MSX2 の場合は(スクリーンモード 5)が動作します。

2 使用方法

MSX-DOS のコマンドラインより次のような入力を行って下さい。


```
gcal [/h] <yyyy/mm/dd> <yyyy/mm/dd>
```

最初の日付は生年月日を、2 番目の日付は調べたい日を入力して下さい。2 番目の日付は、調べたい日が" 今日" であるときに限り省略が可能です。このときは、MSX 内の CLOCK-IC から呼び出されてきますので、CLOCK-IC の日付が、正しいかどうか確認されることをおすすめします。なお、日付の変更方法は、MSX-DOS TOOLS の MSX-DOS manual 第2部をご覧ください。日付の区切りは、'/' のほかに '-'、' ' (空白)が使用できます。カレンダーは、2 番目の日付の月と、その次の月、バイオリズムは、調べたい日を中心に 41 日分を表示します。" gcal" のあとに、"/h" オプションをつけることによりコマンドリファレンスを表示、参照することができます。

3 コンパイル方法

gcal.com のソースファイルは gcal.c です。

コンパイルは C ライブラリに含まれる msxc.bat にコンパイルスイッチをつけてコンパイルします。

```
A>msxc gcal -r13:11:4 -r1500 
```

4 注意点

動作が終了すると停止状態になりますが、'q'、'Q'又は ESC キーを押すことによって MSX-DOS に戻ります。但し、コントロール-STOP をしてしまった場合は、グラフィック画面のまま終了してしまいますのでリセットをかけて下さい。

C) MSX-CURSES を用いたプログラム” show” について

書式 show file1 [file2]

機能 1つまたは2つのファイルの内容を表示します。

解説

MSX-DOS のコマンドラインで1つまたは2つのファイル名を与えて起動することのより1つめのファイルの1ページ目が表示されます。その後のコマンドとして以下のものが有効です。

- ESC ファイルを2つ指定した場合もう一方のファイルに表示を切り替えます。ファイルを1つしか指定しない場合は無効です。
- RET 1行スクロールします。
- DOWN 5行スクロールします。
- SPACE 次のページを表示します。
- q/Q コマンドを終了します。
- h/? ヘルプを表示します。

SHOW はファイル表示のための2つのウィンドウと先頭行のステータスライン及びヘルプ表示のためのウィンドウの計4つのウィンドウをもちいています。MSX-CURSES のプログラミング例としてソースファイルをご参照ください。

お問い合わせについて

弊社では厳重に梱包した上、細心の注意を払って製品を発送しております。万一、輸送上のトラブルが起こった場合にはご一報いただければ新しいものと交換いたします。

マニュアル作成にあたり、なるべく詳細な説明をするように心がけたつもりですが、理解できないところは、実際にコンピュータと向きあって納得のいくまで確かめて下さい。また、他のページを参照するのもひとつの方法です。それでも疑問点が解決できないときは、購入された販売店に問い合わせるか、(株)アスキー ユーザーサポート(直通電話03-3498-0299)までお電話いただければ、係がお答えします。しかしながら、回線が混み合いご迷惑をかけることもありますので、なるべくお手紙にてお願いいたします。その際には、下記の要領で記入して下さい。記入されていない項目が一つでもありますと、回答できかねる場合があります。十分注意して下さい。

また、本製品以外に対してのご意見、ご希望がございましたら、弊社までおよせください。

記

1.送付先 〒107-24 東京都港区南青山6-11-1 スリーエフ南青山ビル
株式会社アスキー ユーザーサポート係

TEL. 03-3498-0299

(祝祭日を除く月～金曜日、10:00～12:00、13:00～17:00)

2.必要項目

(1)お客様の氏名、住所(郵便番号)、電話番号(市外局番も含む)

(2)製品名、製品シリアル番号

(3)機器構成

本体装置名、メモリバイト数

CRT装置名、フロッピーディスク装置名

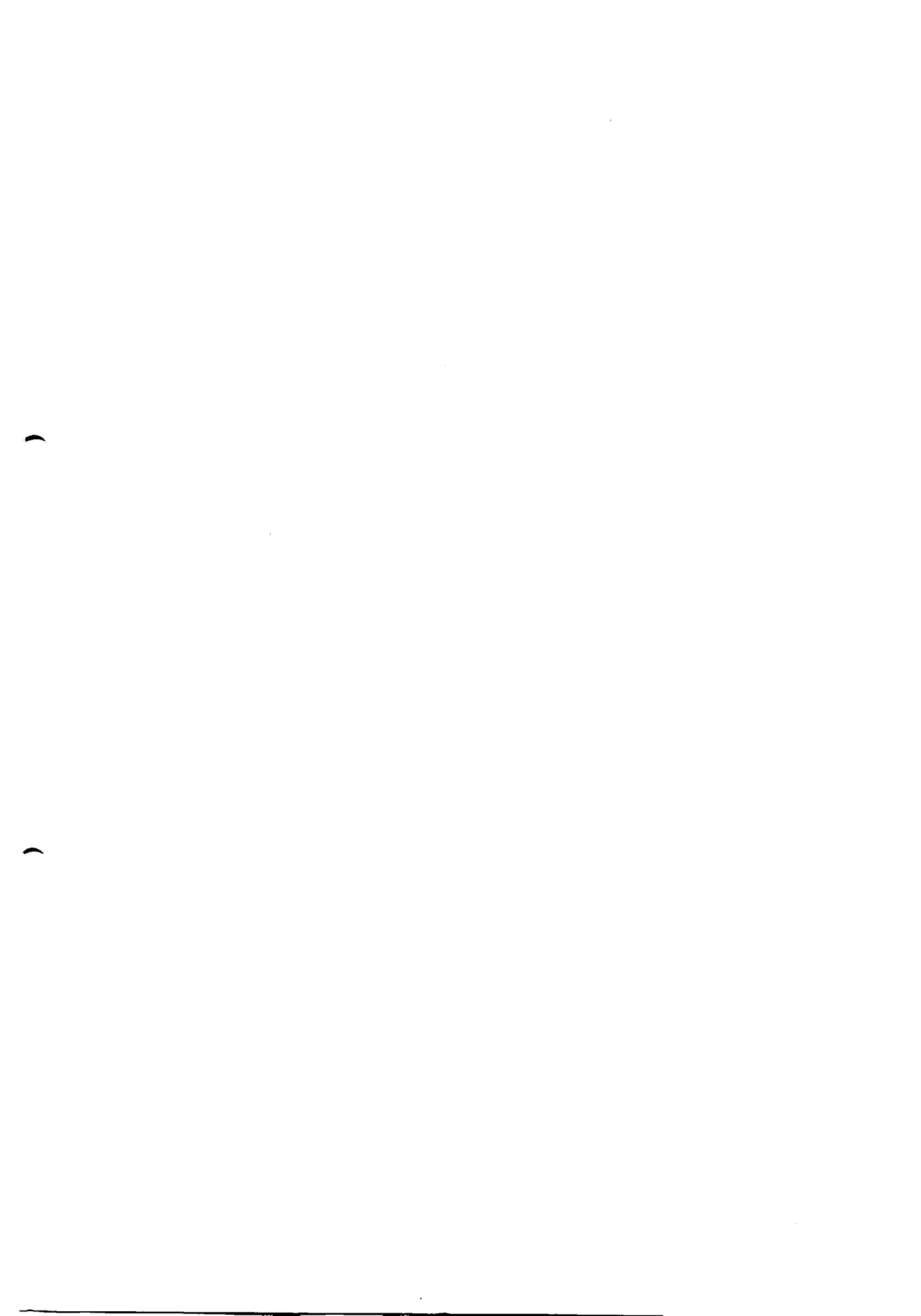
プリンタ装置名

その他I/O、I/F装置名

(4)お問い合わせ内容

お問い合わせの内容は、できるだけ製品のマニュアルに記述されている用語を用いて、具体的かつ明確に記述してください。なお、障害と思われる現象については、その現象を再現可能な情報が必要です。当社で再現できないものは、調査ができません。その現象が発生するまでの操作手順、データを必ず添付して下さい。データディスクがある場合は、そのコピーも同封していただくと調査がスピーディーになります。

また、お客様固有と思われるアプリケーションの設計、作成、運用、保守については、当社のサポート範囲外ですので、お問い合わせいただいても回答できません。ご了承下さいませよう願います。



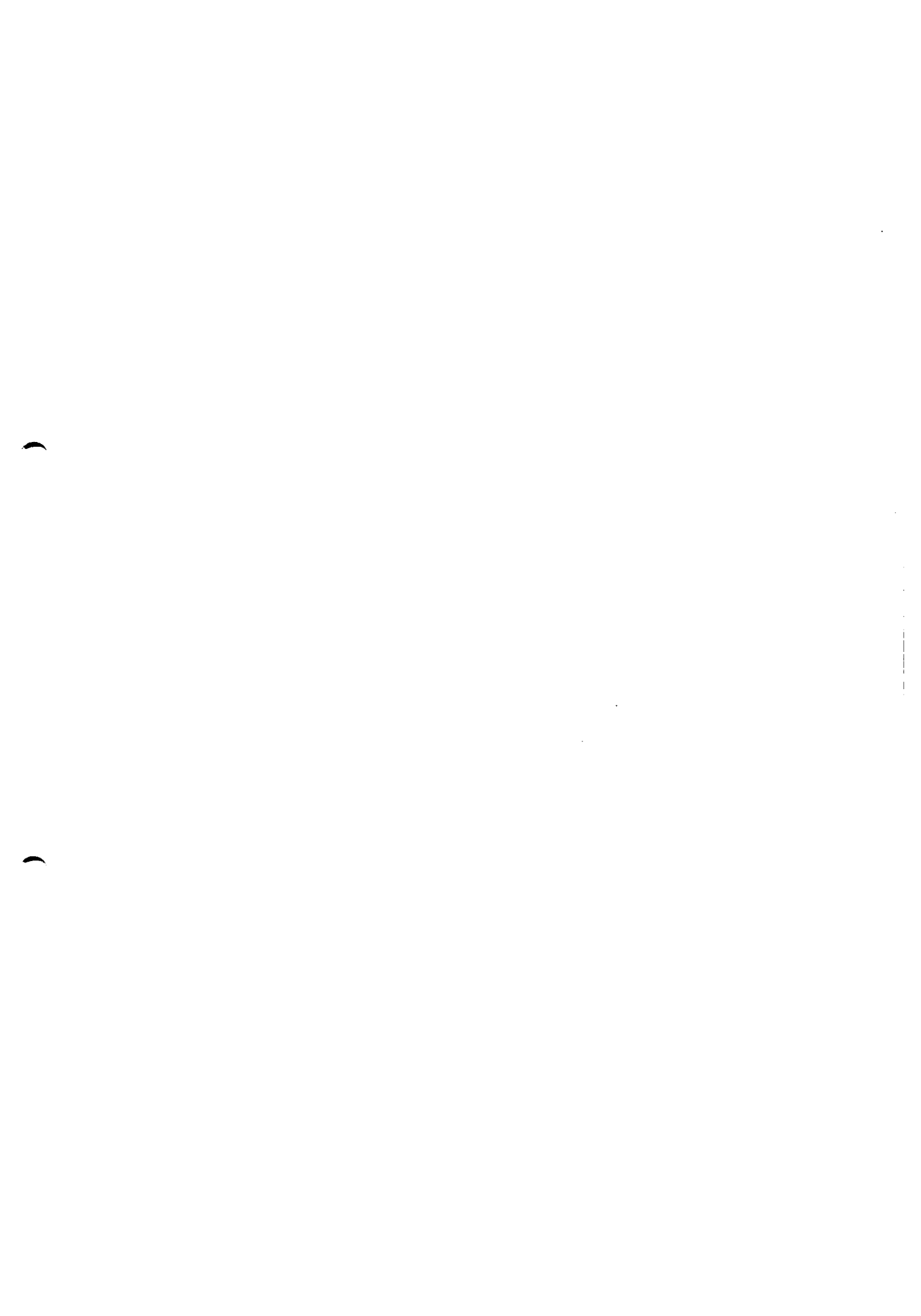
MSX-C Library USER'S MANUAL

1991年7月1日 第3版第1刷発行

監修/編集 株式会社アスキー システム事業部
担当 北浦 訓行

発行所 株式会社アスキー
〒107-24 東京都港区南青山6-11-1 スリーエフ南青山ビル

制作 株式会社ジャパックスインターナショナル



ASCII