

パソコンをTSS端末にするためのCプログラム(1)

武政, 尹士
佐賀大学工学部物理学教室

正木, 延幸
佐賀大学工学部物理学教室

<https://doi.org/10.15017/1470178>

出版情報 : 九州大学大型計算機センター広報. 24 (1), pp.19-46, 1991-01-25. 九州大学大型計算機センター

バージョン :

権利関係 :

パソコンをTSS端末にするためのCプログラム(1)

*武政 尹士, *正木 延幸

1. はじめに

数年前より、大学の研究室から自分の所属する大学の計算機センターは言うにおよばず、全国7大学の共同利用大型計算機センターのマシンを自由に利用出来るようになりました。更には、これらの計算機センターを経由して、国内ばかりではなく海外の研究者と電子メールを交換したり、また直接海外のコンピュータをアクセス出来るようになってきました。このような時、多くの人々は日頃使い慣れたパソコンを大型計算機のTSSターミナル(端末)として利用しています。この時、パソコンを端末として利用するためのソフトウェアが必要です。これがターミナルエミュレータと呼ばれているものです。このようなターミナルエミュレータは、今日まで数多く開発されて来ました。そして、それらの中には通信速度の点を除けば、専用端末より使い易く、かつ専用端末より便利な機能を多くもっているものが発表されています。著者の一人(T.T.)も、ここ数年九州大学大型計算機センターのコンピュータFACOM 780/20をターゲットとして、NECのPC-9801シリーズのパソコン上で動作する無手順(TTY)通信方式のインテリジェントTSS端末エミュレータを開発してきました[1]。そこでは、開発言語としてCが用いられています。

その間に、主に東芝のダイナブックやJ-3100シリーズのパソコンを使用されておられる方々から、私たちの開発してきたエミュレータをそれらのパソコンに移植してほしい由の要請が少なからずありました。しかし、残念ながら私たちはこのような仕事の本職ではありませんので、そのような要請に対して「ソースプログラムを公表していますので、それを解読して他機種でも動作するようにして下さい」とお答えしてきました。エミュレータ開発の初期の段階では、プログラミング構造も簡単であったためにこれでよかったのですが、機能が多くなるにつれてこれが難しくなってきました。

と言うのは、私たちのエミュレータ開発は最初から厳密な全体的設計を行なってプログラミングして行ったわけではなく、ユーザーの皆様からの色々な要望や、私たち自身の個人的考えを元に、後から後へと種々の機能を継足して来ました。そのために、今やソースコードは20,000行を超えプログラミングは迷路のように入り組んでいます。そして、現在では開発者以外の人々にとってはソースコードを入手しようとも解読が不可能に近い状態になっています。

そこで上のような要望に答える1つとして、今回より数回にわたって、TTY型ターミナルエミュレータの基本的構造の説明や、それに付加することによって便利になる色々な機能を実現するための基本的考え方とC言語を用いてのプログラミング法を、実際にソースプログラムを掲載しながら、初心者向けに出来るだけやさしく、かつ丁寧に解説して行きたいと思っています。基本的エミュレータに付加する機能としては、日本語対応、ファイル転送、擬似フルスクリーン、

平成2年11月30日受理

* 佐賀大学 理工学部 物理学教室

受信データのスクロール、PFD、グラフィック描画、グラフィックハードコピー、アドレス付きの文字を端末に送ってくるグラフィックソフトへの対応、UNIXマシンへの対応等々があります。そして、このシリーズをこれからターミナルエミュレータプログラムを作成されようとしておられる方や、既に所有しているターミナルプログラムを自分の環境に合うように手を加えたいと思っておられる読者の方々の一助にしたいと思っています。

なお、プログラミングにおいては高速性を追及するためにPC-98シリーズのパソコンの持つ機能に依存するプログラミングをすることがありますが、他機種にもこれらに対応する機能が見つかると思いますので、原則的にはこれから説明していく全ての機能が他機種へ移植出来るものと思っています。例えば富士通のパソコンFMRシリーズにたいしては文献2が、またダイナブック等のIBC PC互換機にたいしては文献3が参考になると思います。

本稿はMS-DOSとC言語に関して基本的な知識を既にもっておられる方を対象にして書かれています。もしそうでない方でC言語やこのようなプログラミングに興味を持っておられる方は、MS-DOSに関しては村瀬康治著「入門MS-DOS」[4]を、C言語に関してはL. Hancock & M. Krieger著/三浦明美訳「C言語入門」[5]を読まれることをお勧めします。どちらの書籍も初心者を対象にわかりやすく説明した名著であります。なお、既にBASICを修得されている方で、Cへの移行を考えられておられる人には文献6の書籍が役に立つと思われる。

第2節に、今回の開発に使用したハードウェアとソフトウェアの環境を記しています。第3節に本シリーズで作成する端末エミュレータが使用する通信制御仕様が簡単に説明されています。第4節で最も簡単なターミナルエミュレータの構造と実際のプログラミング法を解説しています。そして、第5節では第4節で作ったエミュレータを日本語対応にするためのプログラミングを説明しています。第6節では、ソースプログラムの2つのコンパイラによるコンパイル法を詳しく述べています。

2. ハードウェアとソフトウェアの環境

現在使用中のハードウェアは、NECのPC-9801VXです。メモリーは640キロバイトとなっています。本体には、4メガのRAMディスクが設定されています。これから紹介していくエミュレータを快適に使用するためには、RAMディスクを設定されておかれることをお勧めします。プリンターはNECのPC-PR201Vです。ディスプレイは、640×400ドットの専用高解像度のものがが必要です(カラーディスプレイが望ましい)。また、当然であります。モデムがRS-232Cケーブルでもってパソコン本体と結ばれています。

これから説明していくエミュレータプログラムでは、パソコンのモデルに依存する特殊な機能は使用しませんので、PC-9801シリーズの全機種(ただし、XAとLTは除く)とEPSONのPC-286とPC-386の両シリーズの全機種で動作すると思われます。ただし、PC-9801(無印)とPC-9801Eには漢字ROMが装着されている必要があります。ディスクドライブは2台が接続されている必要があります。プリンターに関しても、PC-PR101、PC-PR201シリーズか、これらのエミュレーションモードを持つ全ての機種が利用可能です。

本稿で使用するオペレーティングシステム(OS)はMS-DOSのVer. 3.3B[7]ですが、Ver. 3.1以上の版のものであればエミュレータは正しく動作します。ただ、MS-DOSはVer. 3.1のPS98-127-XXX以降では、プリンタードライバ(フ

ファイル名 PRINT. SYS) と RS-232C ドライバ (ファイル名 RSDRV. SYS) が別ファイルになっていますので、MS-DOS 構築ファイル "CONFIG. SYS" への両ドライバの登録が必要です。日本語を含むデータをホストコンピュータとやり取りするには、ATOK [8] や松茸 [9] などの日本語入力フロントエンドプロセッサが必要となってきます。それらの使用方法ならびに "CONFIG. SYS" への登録方法は、それぞれのマニュアルをご覧下さい。その他の環境設定は特に必要ありませんが、"CONFIG. SYS" ファイルにおいて "BUFFERS=20" 以上の指定を行なっておくと、ファイルの入出力が速くなります。

処理系としては、MS-C の Ver. 4.0 [10] 以上、もしくは Turbo C Ver. 1.5 [11] 以上を使用して下さい。今回のソースプログラムはこれらより以下の版のコンパイラでもコンパイル出来ませんが、次回以降のプログラムで上記の版よりサポートされた C の関数を使用しますので、古いバージョンのコンパイラを使用されておられる方は、早めに新しいバージョンに乗換えておいて下さい。ソースファイルの作成や修正には、Mif e s 98 [12] 等の MS-DOS 上で動くエディタが必要です。

3. パソコン ターミナルと通信仕様

今日では、ほとんど全てのパソコンに RS-232C 規格のコネクタが配置されています。RS-232C とはコンピュータと周辺装置とのインターフェースの規格です。この RS-232C コネクタにモデムを接続し、公衆電話回線や専用回線を通じてホストコンピュータを呼出すと大型コンピュータの専用ターミナルの代りとして、パソコンを使用することが出来ます。このようにパソコンを大型コンピュータにつないで使用することが、今日では一般化していますが、この主なる理由としては

- (1) パソコンが非常に安くなったこと。
- (2) パソコンをターミナルとして使用する場合、ソフトウェアでもってさまざまな便利な機能を必要に応じて追加することが出来る (インテリジェント端末化)。

などが考えられます。

本シリーズでは、インテリジェント端末をパソコンで作成することを目的として、そのソフトウェアの実際の表現法について述べていく予定です。このインテリジェント端末の通信制御仕様は、表 1 に示すとうりです。

表 1 目的とするインテリジェント端末の通信制御仕様

通信速度	9600ビット/秒以上
通信方式	全2重
同期方式	調歩同期
制御手順	無手順 (TTY手順)

この表に出てくるデータ通信に関する用語について、簡単に説明しておきます。

- ① 通信速度 : 1秒間に送ることの出来るデジタルデータの量です。単位はビット/秒 (bps) , またはキャラクタ/秒 (cps) で表せます。
- ② 全2重通信 : 通信を行なっている双方が、同時にデータを送受信出来る通信方式です。これに対して、トランシーバのように片方ずつ交互に通信する方式を半2重通信と言います。現在、パソコン通信の分野では全2重通信が一般的で半2重通信はほとんど見られません。
- ③ 調歩同期 : パソコンとホストコンピュータとの通信のタイミングをとる(これを同期をとると言う)時に、情報ビットの前後にスタートビットをストップビットを付加して送信し同期をとる方式です。この方式は回路構成が簡単なために装置が安価に出来ます。よって、一般のパソコン通信でよく用いられています。
- ④ 無手順 : 無手順とは、キーボードよりタイプしたデータがその瞬間にホストコンピュータへ流れる方式のことです。この方式では、データのやり取りのための制御も、誤りのための制御も行なわれません。そのために「たれ流し」とか「TTY手順」と呼ばれています。
TTYとはむかしあったテレタイプライター (Tele Typewriter) という名のホストコンピュータとの通信に関しては本質的にキーボード機能しかもたない端末名に由来しています。

通信に関する基本的な用語の内上で説明した以外のものとして、“半2重モード”と“全2重モード”と言うのがあります。これはパソコン端末のキーボードからキーインされたデータをホストコンピュータへ送信する時の処理の違いのことです。つまり、“半2重モード”とはパソコン端末よりキーインされたデータをホストコンピュータへ送信すると同時に、パソコンのディスプレイ画面にも出力するモードのことです。これはまた“ローカル エコー バック モード”とも呼ばれています。これに対して、パソコン端末よりキーインしたデータをホストコンピュータへ送信すると、ホストコンピュータがそのデータを端末側にエコー バックするモードのことを“全2重モード”と言います。このモードではパソコン側でこの送り返されてきたデータをディスプレイ画面に出力することによって、ユーザが自分の入力したデータを確認するようになっています。このモードですと低速の通信速度でホストコンピュータと接続されている場合には、エコー バックが遅くて使い物になりません。なお、九大大型計算機センターのMSPは半2重モードであり、UTS [13] は全2重モードです。この“半2重モード”と“全2重モード”は、上で説明したモデム間の論理的送受信路の方式である“半2重通信方式”と“全2重通信方式”とは別ものであることに注意して下さい。

今後、本シリーズにおいて出てくるデータ通信に関する用語は、そのつど出来るだけ説明するように努める予定です。しかし、原稿のページ数の制約もありますので、より詳しい説明やデータ通信一般に関する解説は他の文献ゆずることもあります。そのような時は、例えば文献14や15を参照して下さい。

4. ダム ターミナル プログラム

ダム ターミナルとは単にホストコンピュータより受信したデータをディスプレイに表示し、パソコンのキーボードより入力された文字をホストコンピュータへ送信する機能しかもっていないターミナルのことです。これは図1に示されたような機能と構成でもって実現されます。しかし図1のなかに、ターミナル エミュレータの本質的なものの全てが入っています。あとはこれ

に種々の便利な機能を付加えて行くと、インテリジェント ターミナル エミュレータなるものが出来上がるわけです。

念のために、図1の流れを説明しておきます。まず第1に、RS-232Cポートを初期化してRS-232C通信機能が動作するようにします。次にホストコンピュータからRS-232Cポートにデータが送られて来ているかどうかをチェックします。もし受信データがあるとそれらの全てをRS-232Cポートより読み出し、同時にそれらをディスプレイ画面に書き出します。この作業後直ちにRS-232Cポートに、この作業中に送られてきたデータがないか調べに戻ります。ここまでで、ホストコンピュータから送られてくるデータの処理が出来ます。

もしRS-232Cポートに受信データがない時は、パソコンのキーボードから入力されたデータがキーボード バッファにあるかどうかをチェックします。キーボードからの入力データがないときは、RS-232Cポートの受信データのチェックに戻り、図1の上半分に示されている処理を行いません。キーボードからの入力データが有る時は、それを読み出すと同時にディスプレイにも書き出します。この作業が終ると直ちにRS-232Cポートのチェックに戻ります。以下、この過程の繰返しであります。この説明でお分かりのように、今回のエミュレータは“半2重モード” (“ローカル エコー バック モード”) になっています。

このように、このエミュレータは常にRS-232Cポートに受信されたデータの処理を最優先に行なっていることが分かって頂けると思います。このアルゴリズムによって高速の通信速度にも追従出来るようになります。もちろんRS-232Cポートから高速にデータを読み出すような機能と同時に読み出したデータを高速にディスプレイ画面に書き出す機能をもった関数を作る必要はありますが。

上で説明した機能の他に端末プログラムとして必要なものに、ブレイク信号をホストコンピュータへ送信する機能があります。これを加えるとターミナル エミュレータ プログラムに必要な機能の全てが揃ったこととなります。それらを図1に従って適当に組合せるとダム ターミナルの出来上がりです。そのためのソース リストが、リスト1からリスト4までに載せてあります。リスト1とリスト2に載っているものは、インクルード ファイルと呼ばれるファイルであります。これはコンパイル時にリスト3とリスト4のファイルに取込まれることによって、リスト1と2で定義された各種のパラメータやマクロがソースファイルにおいて置き換えられます。リスト1では一般的なパラメータの定義と関数の型の宣言がなされています。リスト2では、RS-232C関係のパラメータが定義されています。これらの意味は関数 `rs_init()` の説明のところで詳しく述べます。

リスト1から4までのソースプログラムをコンパイルして出来る実行形式ファイルは、例えば九大大型計算機センターのMSP用のエミュレータとして正しく動作します。また、“`param.h`”なるファイルで定義されているRS-232C関係のパラメータを適当に変えることによって、ほとんど全ての計算機のダム ターミナルとして使用可能です。ただし、ホストコンピュータがエコー バック機能をもっている時は、パソコンのディスプレイ画面に同じ文字が2つならんで表示されます。この場合の解決策は次回以降の稿で説明します。

なお、第1節にも書きましたがこれから紹介していきますプログラムは高速性を追求するために、多くの関数でPC-98シリーズのMS-DOSとPC-98のハードウェアに強く依存するプログラミングになっていることに注意して下さい。よって他機種に移植する時は、それぞれのハードに関する解説書とそれぞれの機種に特有なMS-DOSの機能の使用法を良く調べて下さい [例えば文献2もしくは3を参照]。繰返しになりますが、本稿はエミュレータの基本的構造を解説して行くものであって、移植性には特に気を配ってはいません。一般にエミュレータの色々な機能の高速性を追求すると、どうしてもパソコンの機種に依存するプログラミングになっ

てしまいます。

以下でリスト3とリスト4の中に現れる各々の関数のもつ機能を、簡単に説明します。

リスト3で現れる関数：

(1) `main ()`

関数 `rs_init ()` をコールし RS-232C ポートの初期化とパラメータの設定を行なった後、このエミュレータの本体である関数 `emulator ()` をコールします。もし RS-232C ポートの初期化と設定が正しく行なわれなかった時は、MS-DOS にぬけます。

(2) `emulator ()`

このエミュレータの心臓部です。RS-232C ポートを通じてのホストコンピュータからの受信データの処理 (関数 `recv_tss (len)` がこの機能を担う) と、パソコン端末からの入力データの送信 (関数 `send_tss (code)` がこの機能を担う) の制御を行なっています。この際、図1で説明したように受信データの処理を最優先で行なうようにプログラミングされています。

`CTRL` + `B` でもって関数 `rs_break ()` をコールして、ホストコンピュータにブレーク信号を送信します。一方、`CTRL` + `Q` でもってエミュレータを終了して、MS-DOSに戻るようになっています。ここで `CTRL` + `B` とは、`CTRL` キーを押しながら `B` キーを押すことを意味します (`CTRL` キーと `B` キーの同時押下)。

(3) `recv_tss (len)`

関数 `rs_receive ()` を用いて、RS-232C バッファ内にある "len" 個の1バイトデータを "in_code" に読み出します。次にその読み出したデータを関数 `sys_putchar (code)` をコールすることにより、ディスプレイに書き出します。ただし "in_code" がヌル (0x0:16進ゼロ) のときは、スキップします。(リスト3の行番号63)

(4) `send_tss (code)`

端末のキーボードより入力された1バイトデータ "code" を関数 `sys_putchar (code)` を用いてディスプレイに書き出すと同時に、関数 `rs_sendc (code)` をコールすることによってホストコンピュータに送信します。

リスト4で現れる関数：

(1) `rs_init ()`

Cの子プロセスを実行させる関数 `spawnv [10, 11]` を用いて、MS-DOSの外部コマンドである "SPEED.EXE" [7] を子プロセスとして実行させ、RS-232C ポートの初期化と、その起動を行ないます。子プロセスが正しく実行された時、この関数は "0" を返し、子プロセスが実行できなかったときは "-1" を返します。"SPEED.EXE" に渡される各種のパラメータはヘッダファイル "param.h"

(リスト2)で定義されています。つまり、初期化&起動されるのは標準RS-232Cポート(r0)であり、ボーレート9600ビット/秒(bps)、キャラクタ長7ビット(b7)、偶パリティ(pe)、ストップビット1(s1)、フロー制御ON(xon)です。これらのパラメータの詳しい意味と、パラメータの他の値での設定形式は文献7に詳しく説明されていますので、そちらを参照して下さい。

このプログラムでは" SPEED. EXE"なるファイルは、実行形式になった本エミュレータプログラムの入ったディスクのルートディレクトリになくはなりません。また、システムにRS-232Cドライバ(RSDRV. SYS) [7]が組込まれている必要があります。この2点を注意して下さい。

(2) rs_length ()

Cのソフトウェア割り込み関数 `int 86x [10, 11]` を用いてMS-DOSのサポートするRS-232CのBIOS [Basic Input Output System] をコール (割り込み番号 `0x19`) することにより、RS-232C受信バッファ内の受信データ長を読み出し、その値を返します。この関数と次の2つの関数 `rs_receive ()` と `rs_sendc (code)` についての詳細は、文献16を参照して下さい。

(3) rs_receive ()

Cのソフトウェア割り込み関数 `int 86x` を用いてMS-DOSのサポートするRS-232CのBIOSをコールすること (割り込み番号 `0x19`) により、RS-232C受信バッファにある受信データを1バイト読み出し、その値を返します。この関数は受信データがなければ受信するまで待ちますので、使用時に注意が必要です。

(4) rs_sendc (code)

Cのソフトウェア割り込み関数 `int 86x` を用いてMS-DOSのサポートするRS-232CのBIOSをコールすること (割り込み番号 `0x19`) により、送信データ "code" をRS-232Cポートに出力します。RS-232Cの初期化で、データ長が7ビットに設定されているなら、"shift in (0x0f)" コード、もしくは "shift out (0x0e)" コードの付加と、8ビットデータの7ビット変換を行ってから送信します。

ここで、"shift in (SI)" コードと "shift out (SO)" コードとは、7ビット データでローマ字 (アスキー文字) と半角のカタカナが混在した時、両者を切りわけるコードです。すなわち、SIに続くコードはローマ字と、SOに続くコードはカタカナと約束されています。つまり、この関数を用いるとデータ長が7ビットの場合でも、半角のカタカナの送受信が正しく行なわれるようになっています。

(5) rs_break ()

Cのポート出力関数 `outp [10, 11]` を用いて、RS-232Cのポート番号 "0x32"へ"0x3f"なるデータを出力します。これによってRS-232Cポートへブレイク信号が出力されます。適当なポーズをおいて、ブレイク信号出力状態を通常状態

に戻す必要があります。これはポート番号"0x32"へ"0x37"なるデータを出力することによって行なわれます。これらのポート番号と出力データについての詳細は、PC-9801シリーズのハードウェア マニュアル、例えば文献17を見て下さい。

高速のCPUをもつパソコン使用時には、`pause(30)`ではポーズ時間が不足します。そのようなパソコンを使用する時は、ヘッダファイル"`tss_min.h`"の内で定義されている`BREAK_TIME`を"1"より大きな適当な値に設定してください。また、この関数が呼ばれたことを確認するために、ベルを鳴らすようにしています(リスト4の行番号83)。

(6) `pause(times)`

適当な時間だけポーズする関数です。機種によって、パラメータ `i_max` の値を適当に変更して下さい。

(7) `inkey()`

Cの標準的なMS-DOSシステムコール関数 `intdos[10, 11]` を用いて、キーボード バッファに入力された文字を取りだし、そのコードを返します。文字が入力されていない時は、"0"が返されます。MS-DOSシステムコールについての詳細は、文献18を見て下さい。

(8) `cls()`

Cのソフトウェア割り込み関数 `int86[10, 11]` を用いてMS-DOSの拡張システムコール(割り込み番号 `0xDC`)を行なって、ディスプレイ全体をクリアします。MS-DOSの拡張システムコールについての詳細は、文献18と19を見て下さい。

(9) `sys_putchar(code)`

Cのソフトウェア割り込み関数 `int86` を用いてMS-DOSの拡張システムコール(割り込み番号 `0xDC`)を行なって、ディスプレイに1バイトデータ"`code`"を出力します。漢字データを出力する場合は、シフトJISコードの第1バイト、第2バイトの順に入力する必要があります。

リスト4に現れる関数の多くはNEC版のMS-DOSのBIOSコールとPC-98拡張システムコールを使用しています。よって、他機種への移植の際はそれぞれの機種で上記の機能を再現するBIOSコールを使用して下さい[例えば、文献2もしくは3を参照]。また、どうしてもMS-DOSの標準システムコールのみでRS-232C関係の機能を実現しなければならない場合には、文献20にそのサンプル プログラムが載っています。この時はパソコンの機種依存性はなくなりますが、エミュレータの高速性が失われます。

5. 日本語対応バージョン

さて、第3節で作られたダム ターミナル エミュレータに、今後色々な機能を付加えてインテリジェント化していきたいと思っています。今回はその第1弾として、ダム ターミナルを日本

語対応版にしてみたいと思います。ここで言う日本語とは、漢字、ひらかな、全角記号等のいわゆる全角文字を指しています。

アスキー文字は1バイトのデータで表現されていますが、漢字等の全角文字は2バイトで1文字が表現されているため、2バイトを1文字として取り扱う必要があります。しかし、第3節で作った文字データを取り扱う関数は、受信にせよ送信にせよ全て1バイト単位でしか取り扱えません。その上、送受信のデータの中に日本語コードとアスキーコードのアルファベットが混在している場合には、そのデータの中から日本語を識別しなくてはなりません。これらのために、日本語データを取り扱う時は第3節のプログラミングに比べて少しばかり複雑になります。

漢字コードはJIS規格のもとで定められています。そこでは、漢字シフトコードKI（漢字IN）とKO（漢字OUT）を使って1バイト系コードと日本語が混在しても識別出来るようになってきます。すなわち、KIコードの後に2バイトの日本語コードが続き、KOコードによって1バイト系のコードに戻します。JIS規格ではエスケープ（ESC）コードに続く2バイトのコード（エスケープシーケンス）でKIとKOが定義されています。表2に一般によく使用されている日本語KI、KOコードをまとめてあります。

表2. 漢字コードと対応する漢字シフトコード

漢字コード	漢字シフトコード名称	KIコード	KOコード
JIS	JOIS	ESC, 0X24, 0X40	ESC, 0X28, EX48
	旧JIS	ESC, 0X24, 0X40	ESC, 0X28, 0X4a
	新JIS	ESC, 0X24, 0X42	ESC, 0X28, 0X4a
シフトJIS			

表2の最下段のカラムに記されているシフトJISコードは、MS-DOSが採用している漢字コードです。このコード体系を採用すると、日本語文字はその第1バイト目をチェックすることによって1バイト系の文字と区別することが可能になってきます。そのために、KI、KOコードは必要なくなっています。

さて、パソコンのOSであるMS-DOSが日本語コードとしてシフトJISコードを採用しているために、ホストコンピュータがシフトJISコード以外の漢字コードを採用している時は、シフトJISコードとJISコード間の変換作業が必要となってきます。つまり、ホストコンピュータが漢字コードとしてJISコードを使用している時は、ホストコンピュータからパソコンに送られてくる日本語データはKIコードを取除いた後、シフトJISコードに変換しないと日本語が正しくディスプレイに出力されません。逆に、パソコンから日本語データをホストコンピュータに送信する時はシフトJISコードからJISコードへの変換作業を行ない、その後漢字2バイトデータの前にKIコードを付けて送信することが必要となってきます。JISコード

シフトJISコード間の変換法は、文献21に詳しく解説されていますから参照して下さい。

一方、九大大型計算機センターのUTS [13] ようにホストコンピュータが漢字コードとしてシフトJISを採用している時 [22] は、1バイトデータと漢字データの違いを意識する必要がなくなります。パソコンのキーボードより入力されたデータは、何も加工することなくホストコンピュータへ送信するだけでよく、受信時も単に受取ったデータをそのままディスプレイに書き出すだけでよくなります。よって、この場合はリスト1から4までのプログラムのままで日本語のデータに正しく対応出来ます。

このような考えのもとでダムターミナル用の関数を変更したものと、日本語データの処理のために新しく追加した関数のソースやパラメータ用のステートメントをリスト5からリスト8に載せています。つまり、リスト5にはリスト1に追加すべきステートメントを、リスト2に追加すべき変数の定義をリスト6に載せています。そして、リスト7にはリスト3に追加すべき関数と取り換えられるべき関数のソースプログラムを載せており、リスト8にはリスト4に載っているソースに追加すべき関数を載せています。

ここでは漢字シフトコードとしては、九大大型計算機センターのMSPが採用しているJOISコードを取っています。よって、JOIS以外の漢字コードに対応するためには、ヘッダーファイル"param.h"の内の"KI_CODE2"と"KO_CODE2"の値を、例えば表2に従って適当なものに変える必要があります。

更に、この版では受信データと送信データが一目で区別出来るように、color(c)なる関数を用いて、受信データを緑の文字(リスト7の行番号19)で、送信データを白い文字(リスト7の行番号95)でディスプレイに表示するようにしています。

これらの新しいソースファイルをコンパイルすると、日本語のデータがアスキーデータと同じようにホストコンピュータとやり取りできます。

以下で、リスト7とリスト8に載っている関数の機能を簡単に説明していきます。

リスト7に載っている関数：

これらはリスト3に追加する関数と置き換える必要がある関数です。

(1) reci_tss(len)

内容が変更された関数です。

ホストコンピュータからの受信データの中に"ESC"コードがないかをチェックし、もしあった時は、"fg_reci_esc"なる変数を"1"にセットし、"ESC"コードが受信されたことを記憶します。(このような、ある状態を記録した変数をフラグと呼びます)。そうでない時は、関数output_crt()の内で、ダムターミナルの時と同じような処理をします。

一方、"ESC"コードを受信すると、続いて2バイトの漢字識別コードが送られて来ますので、それらの余分なコードを取除く処理が必要となってきます。これは関数esc_seq(code)を呼ぶことによって行なわれます。

(2) esc_seq(code)

新しく追加された関数です。

この関数によって、"ESC"のあとに送られて来るKIとKOに関する2バイトのコードが取除かれます。この時KIコードの第2バイト目が来ると"fg_reci_

kanji"と"fg_hilo"なる2つのフラグを共に"1"とします(リスト7の行番号57と行番号58)。K0のコードの第2バイト目が来ると"fg_reci_kanji"なるフラグを"0"に戻します(リスト7の行番号65)。

データ長8ビットで通信している時には、KIコードとK0コードの第2バイト目のデータである"0x24"と"0x28"が、8ビット目のビットが立てられて送られてくる可能性があります。その場合にも正しく対応出来るようにしたものが、リスト7の行番号54と行番号62のステートメントです。

(3) output_crt (code)

新しく追加された関数です。

受信データ"code"がアスキー文字の時 (fg_reci_kanji=0) は、そのままディスプレイ画面に書き出します。一方漢字データの時は、その第1バイトをグローバル変数"jcode_hi"に一旦ストアし、そして次に送られてくる第2バイト目をグローバル変数"jcode_lo"にストアします。漢字の2バイトデータが揃ったところで、関数to_shift () をコールすることによりJISコードをシフトJISコードへ変換し、その後ディスプレイ画面へ出力します。

(4) send_tss (code)

内容が変更された関数です。

まず、関数sjis_1st (code) をコールして、キーボードより入力された1バイトデータがシフトJIS漢字の第1バイト目であるかどうかをチェックします。もしそうであると、キーボードよりシフトJIS漢字の第2バイト目を読み込みます。そしてまずそれらをディスプレイ画面に書き出します。次にこのシフトJIS漢字の2バイトデータを関数to_jis () を呼ぶことによってJIS漢字コードに変換します。そしてそれらをホストコンピュータへ送信するのですが、その前に関数to_kanji () を呼んでKIコードをホストコンピュータへ送っておきます。ただし、この漢字データの直前に送られたデータが漢字であった時は、再度KIコードを送らないようにしております(リスト7の行番号102)。

一方、入力された1バイトコードがシフトJIS漢字の1バイト目でなかった場合は、そのコードをそのままディスプレイに書くと同時にホストコンピュータへ送信します。その時、直前に送信されたデータが漢字であった時は関数to_alpha () をコールして前もってK0コードを送信しておく必要があります。

リスト8に載っている関数：

これらはリスト4に追加する関数と置き換える必要がある関数です。

(1) rs_break ()

内容が変更された関数です。

ブレーク信号をホストコンピュータへ送信した後、エスケープシーケンスと漢字に関するフラグを元にもどしておく必要があります(リスト8の行番号20~22)。

(2) to_shift ()

新しく追加された関数です。

J I Sコードの漢字データ (`jcode_hi` = 第1バイト目, `jcode_lo` = 第2バイト目) をシフトJ I Sコードに変換します。その結果, シフトJ I Sコードの第1バイト目が `jcode_hi` に, 第2バイト目が `jcode_lo` に返されます。

(3) `to_jis ()`

新しく追加された関数です。

シフトJ I Sコードの漢字データ (`jcode_hi` = 第1バイト目, `jcode_lo` = 第2バイト目) をJ I Sコードに変換します。その結果, J I Sコードの第1バイト目が `jcode_hi` に, 第2バイト目が `jcode_lo` に返されます。

(4) `sjis_1st (code)`

新しく追加された関数です。

"code"なるキャラクタがシフトJ I Sコードの漢字第1バイト目であるかどうかをチェックします。もしそうであると, この関数は"1"を返し, そうでない時は"0"を返します。

(5) `to_kanji ()`

新しく追加された関数です。

送信データが漢字の時, その前に付加する3バイトのK Iコード (エスケープシーケンス) をホストコンピュータに送信します。

(6) `to_alpha ()`

新しく追加された関数です。

送信データがアスキー文字の時, その前に付加する3バイトのK Oコード (エスケープシーケンス) をホストコンピュータに送信します。

(7) `color (c)`

新しく追加された関数です。

Cのソフトウェア割り込み関数 `int86` を用いてMS-DOSの拡張システムコール (割り込み番号 `0xDC`) を行なって, 文字の属性を変更します。文字の属性とは色, プリンキングや下線など文字を修飾するための情報を言います。"c"に図2に示すような8ビット情報を与えて, この関数をコールするとそれに対応した文字の属性が以後に続く文字に対して適用され, 次の変更まで有効です。この点の詳細は, 文献23を参照して下さい。

6. コンパイルの方法

まず, `Mif es 98 [12]` などのエディタを用いて, ダムターミナルの場合リスト1~4までのプログラムを入力して下さい。その時, リスト1~4に対応してそれぞれ"t s s _

min.h", "param.h", "tss_min1.c", "tss_min2.c"なるファイル名を付けます。ここで最初の2つのファイル"tss_min.h", "param.h"はインクルード ファイルであり、残りの2つのファイル"tss_min1.c"と"tss_min2.c"がエミュレータ本体のソースファイルです。ソースファイルが2つに分れていまして、分割コンパイルを行なう必要があります。コンパイルの方法は、1つのソースファイルでも複数のソースファイルでも変わりありません。分割コンパイルは、リンクする際に複数のオブジェクトを指定するだけでよいのです。

これらのソースプログラムはMS-C Ver. 4.0 [10] 以上、もしくはTurbo C Ver. 1.5 [11] 以上のラージ モデルでコンパイルする必要があります。これは次のような理由によります。つまり、次回以降色々な便利な機能を付加していく予定ですが、そのためにプログラム サイズがかなり大きくなります。そこでCコンパイラのメモリモデルの1つであるスモール モデル内におさめることが難しくなってきますので、最初からラージ モデルを使用しているためです。

例えば、MS-C Ver. 5.1 [24] でコンパイルする時は

```
cl /AL /c /J tss_min1.c ☐
```

```
cl /AL /c /J tss_min2.c ☐
```

として下さい。ここで記号 ☐ は改行キーを押すことを意味します。"cl"はMS-Cの3つのコンパイラ バスを実行するためのコンパイラ ドライバです。すると、2つのオブジェクト ファイル"tss_min1.obj"と"tss_min2.obj"が作られます。次にリンク"link.exe"を用いて、これら2つのオブジェクトとライブラリを結合する必要があります。つまり、

```
link tss_min1+tss_min2, tss_min.exe, NUL, llibce ☐
```

として下さい。ここで、"llibce"はラージ モデル用のライブラリです。これで"tss_min.exe"なる実行形式プログラムが出来ます。漢字対応版の場合にも、リスト1から4までのソースを第4節に記した説明に従って変更&追加後、上と同じ方法でコンパイル出来ます。

上のような分割コンパイルによるプログラム開発を自動化するユーティリティ"make.exe" [10, 24] を用いるとコンパイルとリンクを効率良く行なうことが出来ます。具体的には、

まずエディタを用いて次のようなテキスト ファイルを作ります。

```
tss_min1.obj : tss_min1.c tss_min.h param.h
cl /AL /c /J tss_min1.c

tss_min2.obj : tss_min2.c tss_min.h param.h
cl /AL /c /J tss_min2.c

tss_min.exe : tss_min1.obj tss_min2.obj

link tss_min1+tss_min2, tss_min.exe, NUL, llibce
```

そして、これを例えば"tss_min.mak"なるファイル名でセーブします。そして

```
make tss_min.mak
```

と入力すると、コンパイルとリンクが一度で行なわれ、"tss_min.exe"なる実行形式プログラムが出来ます。また、この方法ですと修正が行なわれたファイルのみコンパイル/リンクが実行されますので、プログラム開発が手際良く行なわれるようになります。上で表れた、コンパイラドライバ("cl")とリンカ("link")のパラメータの詳細やmakeファイルの文法等については、MS-Cのマニュアル[10, 24]を参照して下さい。

Turbo Cでコンパイルする時は、プロジェクトファイルを用いて統合開発環境バージョンTCでコンパイルすると、MS-CでMAKEファイルを用いるのと同じように効率良くプログラム開発が出来ます。そのためには、まず次のようなテキストファイルを作ります。

```
tss_min1.c (tss_min.h, param.h)
```

```
tss_min2.c (tss_min.h, param.h)
```

そして、これを例えば"tss_min.prj"なるファイル名でセーブします。次に、"tc.exe"を立上げ、メニュー画面より"Project"を選択してファイル"tss_min.prj"を登録します。こうしておいてから、**f・9** (make) キーを押すと、"tss_min.prj"の内容に従い、コンパイルとリンクが一度に行なわれます。なお、実行形式ファイル名は"tss_min.exe"となります。これらの点についての詳細は、文献11を参照して下さい。

7. おわりに

C言語によるTSS端末エミュレータプログラミングの解説の最初のものとして、最も基本的なエミュレータと、その1つの拡張としての漢字対応版の2つの版のエミュレータを紹介しました。今回のプログラムでは、RS-232Cポートの初期化を"SPEED"コマンドを用いて行なっています。そのために、エミュレータ実行時に"SPEED.EXE"なるファイルが必要となっています。これはRS-232C関係のパラメータを変更する必要が起った時に、この方法のほうが初心者の方にとって分かりやすいと思ったからです。もちろんこのRS-232Cポートの初期化は、rs_length()やrs_receive()などの関数と同じようにRS-232CのBIOSに割り込み番号0x19のソフトウェア割り込みを実行することによって実現出来ます[16, 25]。この方法については次回詳しく説明したいと思います。また、日本語PFD、日本語ASPEN、ファイル転送(特にホストコンピュータがデータの終結コードとしてヌル(0x0)を送ってこない時の対応策)、エコーバックがある場合等への対応機能も次回解説する予定です。

本稿にソースプログラムの全てを掲載しておりますが、これらを打ち込むのが面倒な方には郵送によるコピーサービスを行ないます。希望される方は、フロッピーディスクを返信用封筒(宛名、切手付き)と一緒に下記住所あてにお送り下さい。メディアは問いません。

〒840 佐賀市本庄町一番地
佐賀大学 理工学部 物理学教室
武政 尹士

この解説シリーズがこれから自分の計算機環境に合うように、既存のエミュレータに手を加えたいと思っておられる方や、新しく自作のエミュレータを作ろうと思っておられる方たちのお役に立てば幸いです。なお、私たちはプロのプログラマでもなければ、Cの専門家でもありませんので、プログラミング上改良すべき点は多々あると思われます。また、本稿にも不十分な点や勉強不足な点が多く見られると思われます。お気づきのことがありましたら、ぜひご指摘をお願い致します。

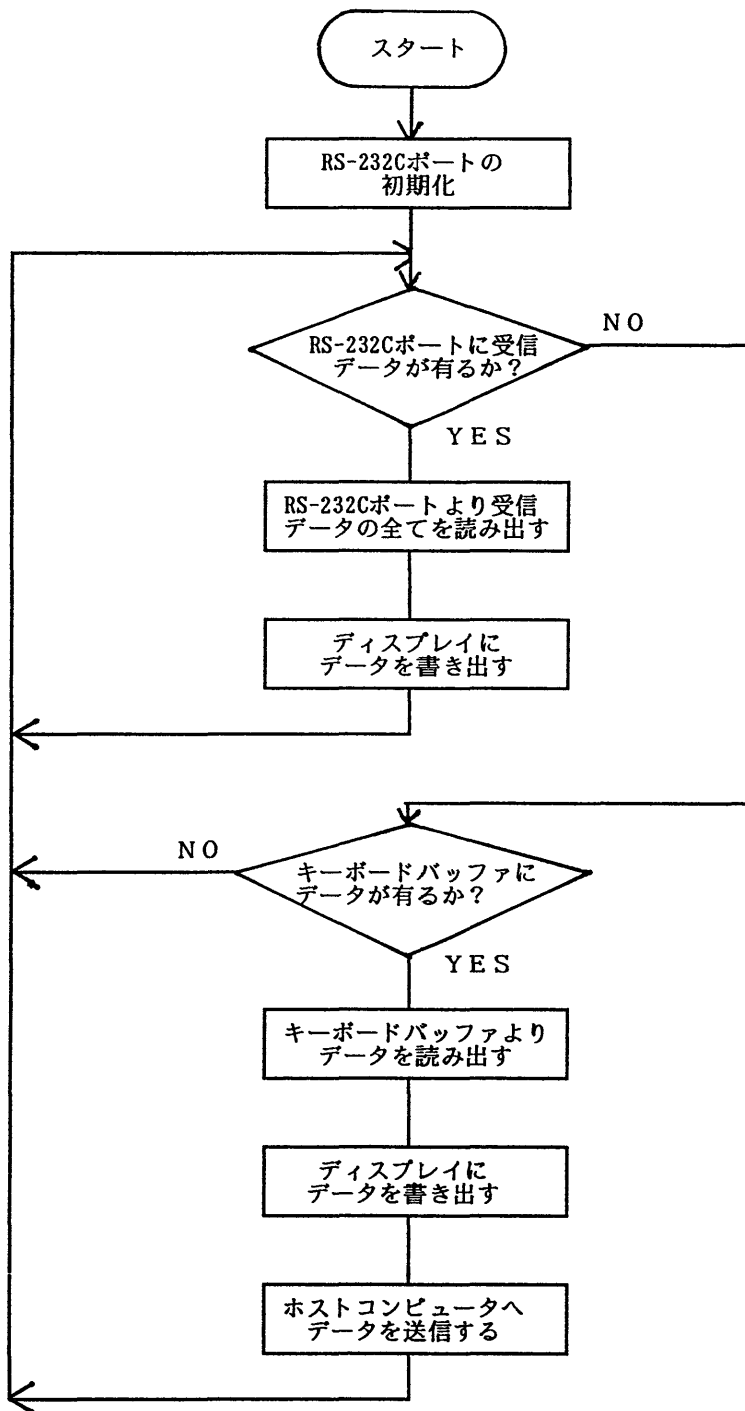
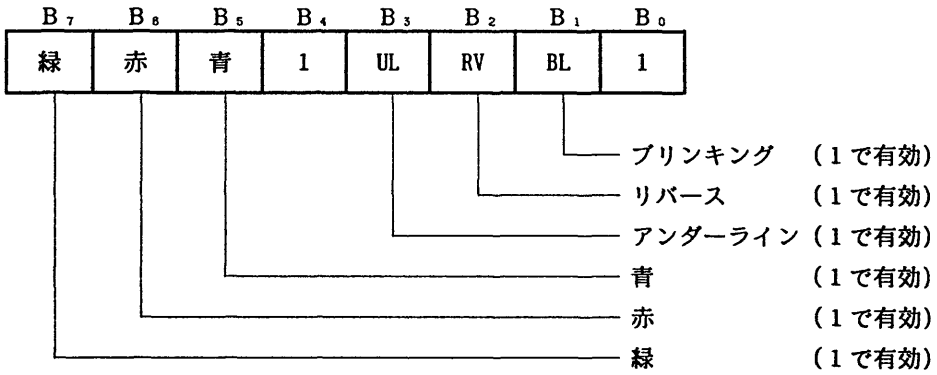


図 1. エミュレータプログラムの基本的な処理の流れ図



B ₇	B ₆	B ₅
緑	赤	青

0	0	0	黒
0	0	1	青
0	1	0	赤
0	1	1	紫

1	0	0	緑
1	0	1	水色
1	1	0	黄
1	1	1	白

例：黄色でプリンキング = 16進数 D3 (2進数 11010011)

図2. 文字属性を設定する8ビットデータ

参考文献

1. 渡辺健次, 武政尹士, 「日本語PFDエミュレータ - PC-9801シリーズ対応 -」, 九州大学大型計算機センター広報, Vol. 21, No. 5 (1988), p. 371.
 渡辺健次, 武政尹士, 「高速多機能TSSPFD端末エミュレータ(グラフィック機能追加) - PC-9801シリーズ対応 -」, 九州大学大型計算機センター広報, Vol. 22, No. 1 (1989), p. 37.
 武政尹士, 渡辺健次, 「高速多機能TSSPFD端末エミュレータ(UTS対応版) - PC9801シリーズ対応」, 九州大学大型計算機センター広報, Vol. 22, No. 3 (1989), p. 209.
 武政尹士, 渡辺健次, 「PC-9801シリーズ対応高速多機能TSS端末エミュレータ: TSSPFD (APSEN, パソコン間対応版)」, 九州大学大型計算機センター広報, Vol. 23, No. 6 (1990), p. 611.
2. インタープログ編, 「富士通FMRシリーズ 徹底解析マニュアル 増補改訂版」, (1990), (株)ピー・エヌ・エヌ.
3. T. Hogan/SE編集部訳編, 「プログラマーのためのPCソースブック」, (1990), (株)翔泳社.
4. 村瀬康治, 「入門MS-DOS」, (1985), (株)アスキー.
5. L. Hancock, M. Krieger/三浦明美(訳), 「C言語入門」, (1984), (株)アスキー.
6. R. J. Traister/野中浩一(訳), 「BASIC to C」, (1986), (株)アスキー.
7. 日本電気(株), 「PC-9800シリーズ MS-DOS 3.3 ユーザーズ リファレンス マニュアル」, (1988).
8. ジャスト システム(株), 「一太郎 Ver. 3.0 活用編」, (1987).
9. 管理工学研究所, 「日本語ワードプロセッサ 松 専門機能編」, (1987).
10. アスキーマイクロソフト, 「Microsoft C ユーザーズ ガイド Ver. 4.0」 (1987), (株)アスキー.
11. Borland International Inc., 「Turbo C ユーザーズ ガイド Ver. 1.5」, (1987), (株)マイクロソフトウェア アソシエイツ.
12. メガソフト(株), 「Mifes 98 Ver. 4.0 ユーザーズ マニュアル」, (1988).
13. 松尾文碩, 「FACOM M-780上のUTS - 世界最高速のUNIX -」, 九州大学大型計算機センター広報, Vol. 20, No. 5 (1987), p. 389.
14. 川田良文, 堤守政, 安藤八郎, 「端末からセンターを利用するためのデータ通信入門」, 名古屋大学大型計算機センター ニュース, Vol. 20, No. 2 (1989), p. 120.
15. 石坂充弘, 「情報通信プロトコル」, (1989), (株)オーム社.
16. 柿園昭俊, 植松早苗, 「MS-DOS メニュー ブック(2)」, (1986), (株)技術評論社.
17. 日本電気(株), 「PC-9800シリーズ PC-9801VX ハードウェア マニュアル」, (1986).

18. 日本電気(株), 「PC-9800シリーズ MS-DOS 3.3 プログラマーズ リファレンス マニュアル Vol. 1 & 2」, (1988).
19. 山崎福馬, 小沢真樹, 「MS-DOS プラス Ver. 2.1/3.1 完全対応」, (1986), (株)ピー・エヌ・エヌ.
20. 瀬藤一起, 鷹野澄, 「MS-DOS」, (1989), 共立出版(株).
21. アスキー書籍編集部編, 「MS-DOS プログラミング テクニック」, (1986), (株)アスキー.
22. 二村祥一, 「UTSの日本語処理機能」, 九州大学大型計算機センター広報, Vol. 20, No. 5 (1987), p. 450.
23. 藤田英時, 幸田敏記, 「PC-Techknow 9800」, (1985), (株)システムソフト.
24. アスキーマイクロソフト, 「Microsoft C ユーザーズ ガイド Ver. 5.1」 (1988), (株)アスキー.
25. 白田耕作, 「PC-98 拡張システムコール詳説」, (1985), インターフェイス 4月号別冊付録.

リスト 1

```

1: /*-----*/
2: * * *
3: * * *
4: * * * Terminal Emulator Program : TSS_min.H * * *
5: * * *
6: * * *
7: * * * <<<<< Microsoft C & Turbo C version >>>>> * * *
8: * * *
9: * * *
10: /*-----*/
11:
12:
13: /*-----*/
14: /* * * * 入力キーコード * * * */
15: /*-----*/
16:
17: #define CTRL_B 0x02
18: #define CTRL_Q 0x11
19:
20:
21: /*-----*/
22: /* * * * システム & 拡張システムコール用機能番号 * * * */
23: /*-----*/
24:
25: #define EXT_DOS 0xdc
26: #define RS232C_BIOS 0x19
27:
28:
29: /*-----*/
30: /* * * * 基本的常数の定義 * * * */
31: /*-----*/
32:
33: #define TRUE 1
34: #define FALSE !TRUE
35:
36: #define BREAK_TIME 1
37:
38:
39: /*-----*/
40: /* * * * マクロ定義 * * * */
41: /*-----*/
42:
43: #define ring_bell() sys_putchar( 0x07 )
44:
45:
46: /*-----*/
47: /* * * * 型宣言 * * * */
48: /*-----*/
49:
50: typedef unsigned char BYTE;
51:
52:
53: /*-----*/
54: /* * * * 関数の型宣言 * * * */
55: /*-----*/
56:
57: int rs_init( void );
58: int rs_length( void );
59: BYTE rs_recieve( void );
60: BYTE inkey( void );
61: void rs_sendc( BYTE );
62: void rs_break( void );
63: void pause( int );
64: void cls( void );
65: void sys_putchar( BYTE );
66: void emulator( void );
67: void send_tss( BYTE );
68: void reci_tss( int );
69:
70: /* ===== END of TSS_min.H ===== */

```

リスト 2

```

1: /*-----*/
2: *
3: *
4: *           Terminal Emulator Program : PARAM.H
5: *
6: *
7: *           <<<<<  Microsoft C & Turbo C version  >>>>>
8: *
9: *
10: /*-----*/
11:
12:
13: /*-----*/
14: /*           R S - 2 3 2 C 関係のパラメータ
15: /*-----*/
16:
17: #define  SPEED_PATH           "xyzspeed.exe"
18: #define  RS_PORT_NO          "r0"
19: #define  BAUD_RATE           "9600"
20: #define  CHAR_LEN            "b7"
21: #define  PARITY               "pe"
22: #define  STOP_BIT            "s1"
23: #define  XONOFF              "xon"
24:
25: /* =====                END of PARAM.H                ===== */

```

リスト 3

```

1: /*-----*/
2: *
3: *
4: *           Terminal Emulator Program : TSS_min1.C
5: *
6: *
7: *           <<<<<  Microsoft C & Turbo C version  >>>>>
8: *
9: *
10: /*-----*/
11:
12:
13: #include <dos.h>
14: #include <process.h>
15: #include "tss_min.h"
16: #include "param.h"
17:
18:
19: /* -----                main()                ----- */
20:
21: void main( void )
22: {
23:     if ( rs_init() == -1 )
24:         exit( -1 );
25:     else
26:         emulator();
27:     exit( 0 );
28: }
29:
30: /* -----                emulator()                ----- */
31:
32: void emulator( void )
33: {
34:     int len;
35:     BYTE key_code;
36:
37:     cls();
38:     while ( TRUE ) {
39:         if ( ( len = rs_length() ) != 0 ) reci_tss( len );
40:         if ( ( key_code = inkey() ) == 0 ) continue;

```

リスト 3 (続)

```

41:     switch ( key_code ) {
42:         case CTRL_B :
43:             rs_break();
44:             break;
45:
46:         case CTRL_Q :
47:             return;
48:
49:         default :
50:             send_tss( key_code );
51:     }
52: }
53: }
54:
55: /* -----      reci_tss( len )      ----- */
56:
57: void reci_tss( int len )          /* 受信データの処理 */
58: {
59:     BYTE in_code;
60:
61:     do {
62:         while ( len-- > 0 ) {
63:             if ( ( in_code = rs_receive() ) == 0x0 ) continue;
64:             sys_putchar( in_code );
65:         }
66:     }
67:     while ( ( len = rs_length() ) != 0 ) ;
68: }
69:
70: /* -----      send_tss( code )      ----- */
71:
72: void send_tss( BYTE code )        /* 送信データの処理 */
73: {
74:     sys_putchar( code );
75:     rs_sendc( code );
76: }
77:
78: /* =====      END of TSS_min1.C      ===== */

```

リスト 4

```

1:  /*-----*
2:  *
3:  *
4:  *          Terminal Emulator Program : TSS_min2.C
5:  *
6:  *
7:  *          <<<<<  Microsoft C & Turbo C version  >>>>>
8:  *
9:  *
10: /*-----*/
11:
12:
13: #include <dos.h>
14: #include <process.h>
15: #include "tss_min.h"
16: #include "param.h"
17:
18:
19: /* -----      rs_init()      ----- */
20:
21: int rs_init( void )              /* RS - 2 3 2 C の初期化 */
22: {
23:     char *rs_param[ 8 ];
24:
25:     rs_param[0] = SPEED_PATH;
26:     rs_param[1] = RS_PORT_NO;
27:     rs_param[2] = BAUD_RATE;

```

リスト 4 (続)

```

28:   rs_param[3] = CHAR_LEN;
29:   rs_param[4] = PARITY;
30:   rs_param[5] = STOP_BIT;
31:   rs_param[6] = XONOFF;
32:   rs_param[7] = "☹0";
33:   return ( spawnv( P_WAIT, SPEED_PATH, rs_param ) == -1 ) ? -1 : 0;
34: }
35:
36: /* ----- rs_length() ----- */
37:
38: int rs_length( void )          /* 受信データ長の取得 */
39: {
40:     union REGS inregs, outregs;
41:     struct SREGS segregs;
42:
43:     segread( &segregs );
44:     segregs.ds = 0x0060;
45:     inregs.h.ah = 0x04;
46:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
47:     return outregs.x.cx;
48: }
49:
50: /* ----- rs_receive() ----- */
51:
52: BYTE rs_receive( void )       /* 1バイト受信 */
53: {
54:     union REGS inregs, outregs;
55:     struct SREGS segregs;
56:
57:     segread( &segregs );
58:     segregs.ds = 0x0060;
59:     inregs.h.ah = 0x02;
60:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
61:     return outregs.h.al;
62: }
63:
64: /* ----- rs_sendc( code ) ----- */
65:
66: void rs_sendc( BYTE code )    /* 1バイト送信 */
67: {
68:     union REGS inregs, outregs;
69:     struct SREGS segregs;
70:
71:     segread( &segregs );
72:     segregs.ds = 0x60;
73:     inregs.h.ah = 0x01;
74:     inregs.h.cl = code;
75:     int86x( RS232C_BIOS, &inregs, &outregs, &segregs );
76: }
77:
78: /* ----- rs_break() ----- */
79:
80: void rs_break( void )        /* ブレーク信号の送信 */
81: {
82:     outp( 0x32, 0x3f );
83:     ring_bell();
84:     pause( 30 * BREAK_TIME );
85:     outp( 0x32, 0x37 );
86: }
87:
88: /* ----- pause( times ) ----- */
89:
90: void pause( int times )      /* タイマー */
91: {
92:     int i, j;
93:     static int i_max = 1000;
94:
95:     for ( j = 0; j < times; j++ )
96:         for ( i = 0; i < i_max; i++ )

```


リスト 4 (続)

```

97:         ;
98:     }
99:
100: /* ----- inkey() ----- */
101:
102: BYTE inkey( void )          /* キーボードからの入力 */
103: {                          /* データの取り出し */
104:     union REGS inregs, outregs;
105:
106:     inregs.h.ah = 0x06;
107:     inregs.h.dl = 0xff;
108:     intdos( &inregs, &outregs );
109:     return outregs.h.al;
110: }
111:
112: /* ----- cls() ----- */
113:
114: void cls( void )          /* ディスプレイ画面の全消去 */
115: {
116:     union REGS inregs, outregs;
117:
118:     inregs.h.cl = 0x10;
119:     inregs.h.ah = 0x0a;
120:     inregs.h.dl = 2;
121:     int86( EXT_DOS, &inregs, &outregs );
122: }
123:
124: /* ----- sys_putchar( code ) ----- */
125:
126: void sys_putchar( BYTE code ) /* ディスプレイへの1バイト */
127: {                          /* コードの出力 */
128:     union REGS inregs, outregs;
129:
130:     inregs.h.cl = 0x10;
131:     inregs.h.ah = 0x00;
132:     inregs.h.dl = code;
133:     int86( EXT_DOS, &inregs, &outregs );
134: }
135:
136: /* ===== END of TSS_min2.C ===== */

```

リスト 5

```

1: /******
2:
3:     リスト 1 に追加するパラメータと関数の型の宣言
4:
5: *****/
6:
7:
8:
9: /*-----*/
10: /*          入力キーコード          */
11: /*-----*/
12:
13: #define ESC          0x1b
14:
15:
16: /*-----*/
17: /*          表示色、表示属性コード          */
18: /*-----*/
19:
20: #define GREEN          0x81
21: #define WHITE          0xe1
22:
23:

```

リスト 5 (続)

```

24: /*-----*/
25: /*          関数の型宣言          */
26: /*-----*/
27:
28: int     esc_seq( BYTE );
29: int     sjis_1st( BYTE );
30: void    to_shift( void );
31: void    to_jis( void );
32: void    to_kanji( void );
33: void    to_alpha( void );
34: void    output_crt( BYTE );
35: void    color( int );
36:
37: /* =====          END of TSS_min.H          ===== */

```

リスト 6

```

1: /*-----*/
2:
3:          リスト 2 に追加するパラメータ
4:
5: /*-----*/
6:
7:
8: /*-----*/
9: /*          J O I S 用 K I , K O コード          */
10: /*-----*/
11:
12: #define KI_CODE1          0x24
13: #define KI_CODE1_S        0xa4
14: #define KI_CODE2          0x40
15: #define KO_CODE1          0x28
16: #define KO_CODE1_S        0xa8
17: #define KO_CODE2          0x48
18:
19: /* =====          END of PARAM.H          ===== */

```

リスト 7

```

1: /*-----*/
2:
3:          リスト 3 に追加する関数と変更する関数
4:
5: /*-----*/
6:
7:
8: int     fg_send_kanji = 0, fg_reci_kanji = 0, fg_hilo, fg_reci_esc = 0;
9:
10: BYTE    jcode_hi, jcode_lo;
11:
12:
13: /*          -----          reci_tss( len )          -----          */
14:
15: void     reci_tss( int len )          /* 受信データの処理 */
16: {
17:     BYTE    in_code;
18:
19:     color( GREEN );
20:     do {
21:         while ( len-- > 0 ) {
22:             in_code = rs_receive();
23:             switch ( fg_reci_esc ) {

```

リスト7 (続)

```

24:         case 0 :
25:             switch ( in_code ) {
26:                 case ESC :
27:                     fg_reci_esc = 1;
28:                     continue;
29:
30:                 case 0 :           /* ヌルコードの読み飛ばし */
31:                     continue;
32:
33:                 default :
34:                     output_crt( in_code );
35:                     continue;
36:             }
37:
38:         case 1 :
39:             fg_reci_esc = 0;
40:             if ( esc_seq( in_code ) != 0 ) len = rs_length();
41:             continue;
42:         }
43:     }
44: }
45: while ( ( len = rs_length() ) != 0 ) ;
46: }
47:
48: /* ----- esc_seq( code ) ----- */
49:
50: int esc_seq( BYTE code )           /* エスケープシーケンスの */
51: {                                   /* 処理 */
52:     switch ( code ) {
53:         case KI_CODE1 : ;
54:         case KI_CODE1_S :
55:             while ( rs_length() == 0 ) ;
56:             rs_receive();           /* K I コード ( 2 バイト目) */
57:             fg_reci_kanji = 1;     /* の読み飛ばし */
58:             fg_hilo = 1;
59:             return 1;
60:
61:         case KO_CODE1 : ;
62:         case KO_CODE1_S :
63:             while ( rs_length() == 0 ) ;
64:             rs_receive();           /* K O コード ( 2 バイト目) */
65:             fg_reci_kanji = 0;     /* の読み飛ばし */
66:             return 1;
67:
68:         default :
69:             fg_reci_kanji = 0;
70:             return 0;
71:     }
72: }
73:
74: /* ----- output_crt( code ) ----- */
75:
76: void output_crt( BYTE code )       /* 受信データのディスプレイ */
77: {                                   /* 画面への出力 */
78:     if ( fg_reci_kanji == 0 ) sys_putchar( code ); /* アスキーデータ */
79:     else {                           /* 漢字データ */
80:         fg_hilo = 1 - fg_hilo;
81:         if ( fg_hilo == 0 ) jcode_hi = code;
82:         else {
83:             jcode_lo = code;
84:             to_shift();
85:             sys_putchar( jcode_hi );
86:             sys_putchar( jcode_lo );
87:         }
88:     }
89: }
90:
91: /* ----- send_tss( code ) ----- */
92:

```

リスト 7 (続)

```

93: void send_tss( BYTE code )          /* 送信データの処理 */
94: {
95:     color( WHITE );
96:     if ( sjis_1st( code ) == 1 ) {   /* 漢字データ */
97:         jcode_hi = code;
98:         jcode_lo = inkey();
99:         sys_putchar( jcode_hi );
100:        sys_putchar( jcode_lo );
101:        to_jis();
102:        if ( fg_send_kanji != 1 ) to_kanji();
103:        rs_sendc( jcode_hi );
104:        rs_sendc( jcode_lo );
105:        fg_send_kanji = 1;
106:    }
107:    else {                             /* アスキーデータ */
108:        sys_putchar( code );
109:        if ( fg_send_kanji != 0 ) to_alpha();
110:        rs_sendc( code );
111:        fg_send_kanji = 0;
112:    }
113: }
114:
115: /* ===== END of TSS_min1.C ===== */

```

リスト 8

```

1: /* =====
2:
3:             リスト 4 に追加する関数と変更する関数
4:
5: ===== */
6:
7:
8: extern int fg_reci_kanji, fg_hilo, fg_reci_esc;
9: extern BYTE jcode_hi, jcode_lo;
10:
11:
12: /* ----- rs_break() ----- */
13:
14: void rs_break( void )                /* ブレーク信号の送信 */
15: {
16:     outp( 0x32, 0x3f );
17:     ring_bell();
18:     pause( 30 * BREAK_TIME );
19:     outp( 0x32, 0x37 );
20:     fg_reci_kanji = 0;                /* 漢字に関する */
21:     fg_hilo = 0;                     /* フラグを */
22:     fg_reci_esc = 0;                 /* おろす */
23: }
24:
25: /* ----- to_shift() ----- */
26:
27: void to_shift( void )                /* J I S → シフト J I S 変換 */
28: {
29:     if ( jcode_hi % 2 == 1 ) {
30:         jcode_lo += 0x1f;
31:         if ( jcode_lo >= 0x7f ) jcode_lo++;
32:     }
33:     else jcode_lo += 0x7e;
34:     jcode_hi = ( jcode_hi - 0x21 >> 1 ) + 0x81;
35:     if ( jcode_hi > 0x9f ) jcode_hi += 0x40;
36: }
37:
38: /* ----- to_jis() ----- */
39:
40: void to_jis( void )                 /* シフト J I S → J I S 変換 */

```

リスト 8 (続)

```

41: {
42:   jcode_hi -= ( jcode_hi > 0x9f ) ? 0xb1 : 0x71;
43:   jcode_hi = jcode_hi * 2 + 1;
44:   if ( jcode_lo > 0x9e ) {
45:     jcode_lo = jcode_lo - 0x7e;
46:     jcode_hi++;
47:   }
48:   else {
49:     if ( jcode_lo > 0x7e ) jcode_lo--;
50:     jcode_lo -= 0x1f;
51:   }
52: }
53:
54: /* ----- sjis_1st( code ) ----- */
55:
56: int sjis_1st( BYTE code )          /* シフト J I S 漢字の */
57: {                                  /* 第 1 バイト目の検出 */
58:   code &= 0xff;
59:   if ( ( code >= 0x81 && code <= 0x9f ) ||
60:       ( code >= 0xe0 && code <= 0xfc ) )
61:     return 1;
62:   else
63:     return 0;
64: }
65:
66: /* ----- to_kanji() ----- */
67:
68: void to_kanji( void )              /* 漢字 I N コードの送信 */
69: {
70:   rs_sendc( ESC );
71:   rs_sendc( KI_CODE1 );
72:   rs_sendc( KI_CODE2 );
73: }
74:
75: /* ----- to_alpha() ----- */
76:
77: void to_alpha( void )              /* 漢字 O U T コードの送信 */
78: {
79:   rs_sendc( ESC );
80:   rs_sendc( KO_CODE1 );
81:   rs_sendc( KO_CODE2 );
82: }
83:
84: /* ----- color( c ) ----- */
85:
86: void color( int c )                /* 表示文字の色、属性の設定 */
87: {
88:   union REGS  inregs, outregs;
89:
90:   inregs.h.cl = 0x10;
91:   inregs.h.ah = 0x02;
92:   inregs.h.dl = ( BYTE ) c;
93:   int86( EXT_DOS, &inregs, &outregs );
94: }
95:
96: /* ===== END of TSS_min2.C ===== */

```