TMM1000 LINUX クロス開発環境を用いたＣ言語入門

2003/07/16

東和メックス株式会社

■　商標

Red Hat ならびに Shadow Man ロゴは米国およびその他の国で Red Hat,Inc.の登録商標若しくは商標です。

■　注意

・既に Red Hat(R) Linux(R) 8.0 のレッドハットによる販売は終了しています
・Errata の提供は、2003 年 12 月 31 日で終了します。
・FTP 等で入手した製品にはレッドハット側で一切のお問い合わせに対応していません

■ 改変履歴

日付	内容
2003.7.16	・Red Hat 社の著作権情報追加

. INDEX

1.はじめに

本書は、東和 Linux ボード（TMM1000）を用いて、基本的な C 言語の説明、Linux でプログラムを作成し、コンパイルし、東和 Linux ボード（TMM1000）上での実行する手順の説明、及び Linux の基本的なコマンドの説明を記したものです。

開発環境の構築については「TMM1000 LINUX クロス開発環境の構築とソフトウェア開発」をご覧ください。

2 章ではプログラムのコンパイル方法と実行方法について書いてあります。ここでは必要な Linux のコマンドの説明も記してあります。

3 章では変数と配列について説明してあります。

4 章では繰り返し処理 for 文、while 文、条件分岐 if 文、switch～case 文の説明とこれらを用いたプログラム例を示してあります。

5 章ではシリアルポートからの入出力として open 関数、read 関数、write 関数について説明してあります。また、TMM1000 に付属のサンプルプログラムを変更したプログラム例を示してあります。

6 章では tmm1000 で用意した描画関数の使い方について書いてあります。この章で解説してある関数は tmm1000 でしか使えないので注意してください。

2.プログラムのコンパイルと実行

2-1.ユーザ”tmm1000”の作成

(a) スーパーユーザ”root”

Red Hat Linux をインストールし、”root”以外のユーザを追加していないという前提で説明します。”root”はスーパーユーザという特別な権限を持ったユーザです。このユーザは OS についてのあらゆる操作を実行することができます。また、全てのファイル、ディレクトリ（Windows でいうフォルダ）にアクセスすることができます。従って、重要なファイルを更新する等の操作により、OS を不安定にしてしまう可能性があります。よって、普通は”root”以外のユーザを作成して、これでログオンし、作業を行います。

(b) ユーザ"tmm1000"の作成とパスワードの設定　useradd　passwd　userdel

"root"でログオンしてください。

次に、ターミナルを起動してください。"**useradd**"はユーザを作成するコマンドです。これはスーパーユーザでないと実行できないコマンドです。スーパーユーザについては 2.1（a）節、（d）節を参照してください。

/usr/sbin/useradd　ユーザ名

＊ Red Hat Linux の場合は"/usr/sbin/"が useradd の前に必要です

所属するグループを指定する場合はオプションで"**-g**"を付けてグループ（グループについては 2-1（c）節を参照してください）を設定します。上記のようにユーザを作成するときにグループ名を省略した場合、ユーザ名と同じグループが作成され、これに属します。

/usr/sbin/useradd –g　グループ名　ユーザ名

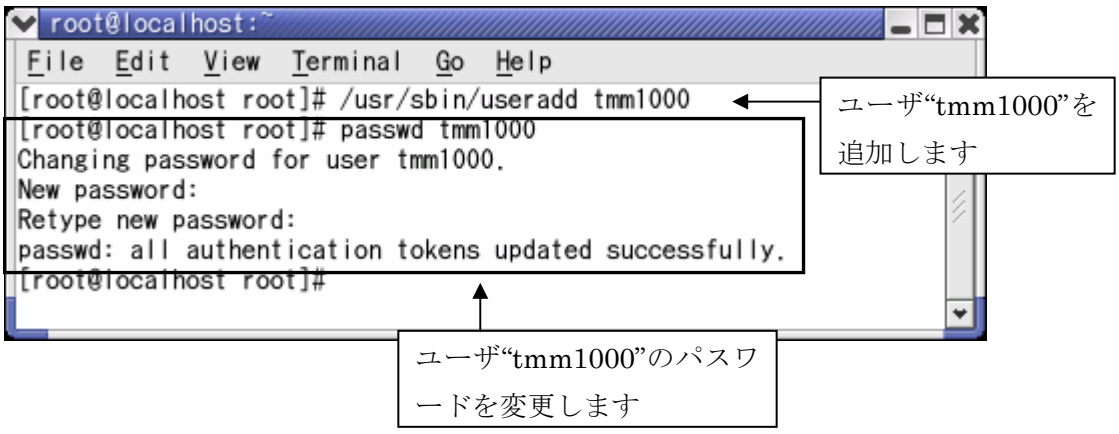
＊ Red Hat Linux の場合は"/usr/sbin/"が useradd の前に必要です

ターミナルに"**/usr/sbin/useradd tmm1000**"と入力して Enter を押してください。グループを省略しているので、ユーザ"tmm1000"はグループ"tmm1000"に属します。これで"tmm1000"というユーザが作成されました。

次にパスワードを設定します。"**passwd**"はパスワードを設定するコマンドです。ユーザ名を指定できるのはスーパーユーザのみです。スーパーユーザ以外のユーザは自分のパスワードのみ変更できます。また、スーパーユーザ以外のユーザは"(current) UNIX password:"とはじめに表示されるので、現在のパスワードを入力する必要があります。

passwd　ユーザ名

ターミナルに"**passwd tmm1000**"と入力して Enter を押して、ユーザ"tmm1000"のパスワードを入力してください。確認のためにパスワードは 2 回入力する必要があります。



パスワードが短すぎる（例えば”tmm”）、パスワードが単純すぎる（例えば”tmm1000”）などの場合は、”BAD PASSWORD：～”という警告がでます。～は入力したパスワードによって異なります。

スーパーユーザの場合は”Retype new password:”と表示されるので、ここでもう一度パスワードを入力すれば変更できます。

それ以外のユーザの場合は”New password”と表示されるので、警告が出ないパスワードを入力しなおしてください。

“**userdel**”はユーザを削除するコマンドです。**これはスーパーユーザでないと実行できないコマンドです。**

/usr/sbin/userdel　ユーザ名

＊ Red Hat Linux の場合は”/usr/sbin/”が useradd の前に必要です

作成したユーザ”**tmm1000**”を削除する場合はターミナルに”**/usr/sbin/userdel tmm1000**”と入力して Enter を押してください。

補足：

ユーザーのディレクトリは削除されませんが所有者であるユーザーの名前が表示されなくなります（cd ~tmm1000　とかもできなくなります）

(c) グループの作成　groupadd

Linux ではユーザはあるグループに属しています。

”groupadd”はグループを作成するコマンドです。このコマンドは**これはスーパーユーザでないと実行できないコマンドです。**

/usr/sbin/groupadd グループ名

＊ Red Hat Linux の場合は”/usr/sbin/”が useradd の前に必要です

詳しくは 2-5 節で説明しますが、ユーザは所属するグループのファイルにアクセスできます。例えば、ユーザ”tmm1000”は”tmm1000”というグループに属しているので、”tmm1000”というグループのファイルにアクセスにアクセスできます。

(d) ユーザの切り替え　su

ユーザ”tmm1000”を作成したら、一度ログオフして、”tmm1000”でログオンしてください。”tmm1000”はスーパーユーザではないので、mount などの特定のコマンドを実行することはできません。また、特定のフォルダにアクセスすることもできません。そこで、**必要なときだけスーパーユーザにユーザを切り替える**必要があります。

”su”はログオフしないで別のユーザに切り替えるコマンドです。

```
su ユーザ名
```

上記のようにターミナルに入力し、Enter を押してください。そしてパスワードを入力すればユーザを切り替えることができます。

ユーザ名を省略するとスーパーユーザに切り替えることができます。スーパーユーザに切り替える場合は、ターミナルに”**su**”と入力し、Enter を押して”root”のパスワードを入力します。

2-2.エディタ kate を用いたプログラムの編集

(a) カレントディレクトリの移動　cd

ディレクトリ”/home/tmm1000”にディレクトリを作成し、プログラムを作成します。はじめに、カレントディレクトリ（現在作業を行っているディレクトリ）を移動する必要があります。”**cd**”はカレントディレクトリを移動するコマンドです。

```
cd ディレクトリ
```

ターミナルに”**cd /home/tmm1000**“と入力して Enter を押して、カレントディレクトリを移動してください。

＊”/home/tmm1000”は”~tmm1000”と省略できます

(b) ディレクトリの作成　mkdir

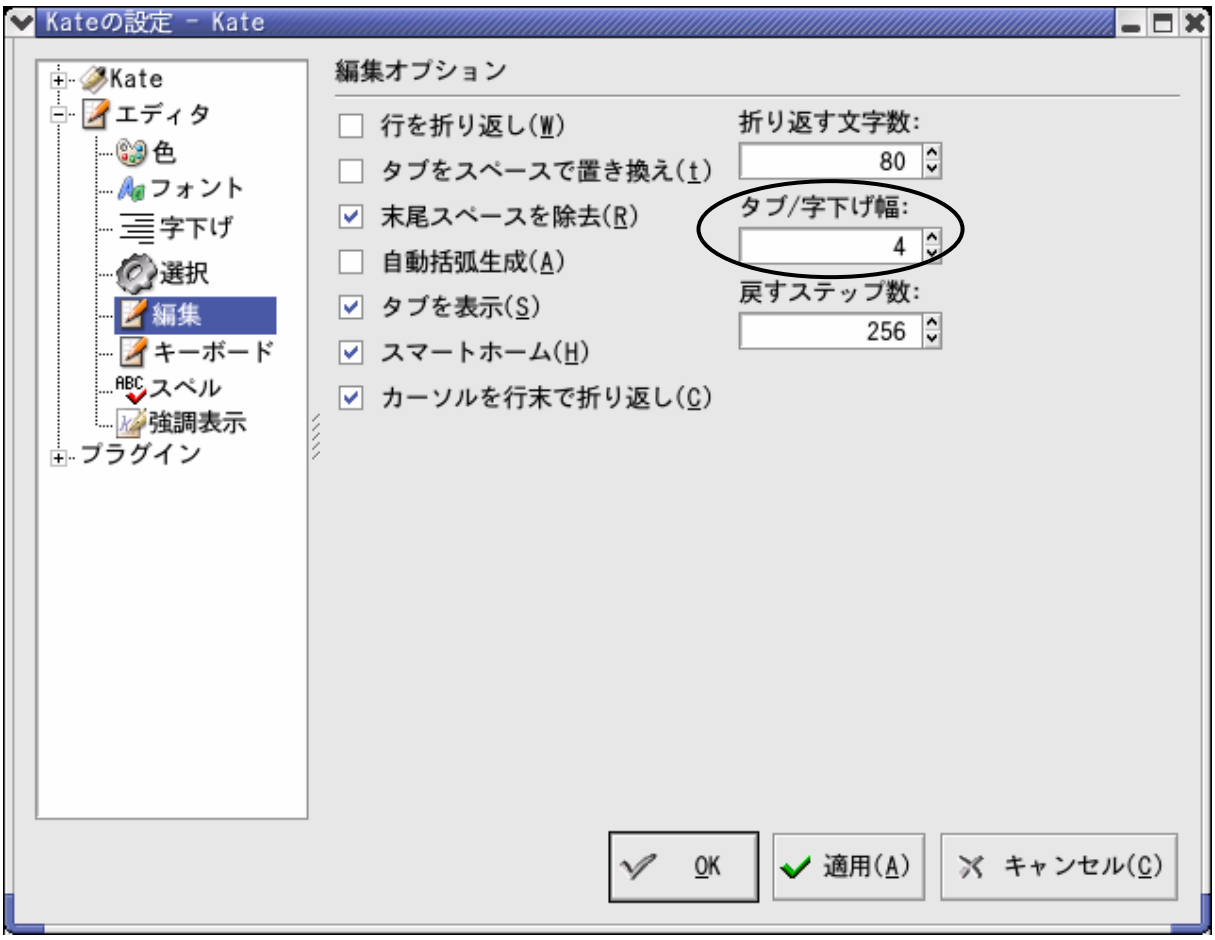
次に”/home/tmm1000”に”hello”というディレクトリを作成します。”**mkdir**”はディレクトリを作成するコマンドです。

```
mkdir ディレクトリ名
```

ターミナルに”**mkdir hello**”と入力して Enter を押して、ディレクトリを作成してください。

(c) エディタ”kate”の設定

ターミナルに”cd hello”と入力して Enter を押し、カレントディレクトリを移動してください。次にターミナルに”**kate**”と入力し、Enter を押し、エディタ kate を起動してください。



”**設定**”－ ”**kate を設定**”をクリックしてください。”**タブ／字下げ幅**”を **4**（標準的な字下げ幅です）に設定します。

設定が終了したら”適用”をクリックして、”OK”をクリックしてください。

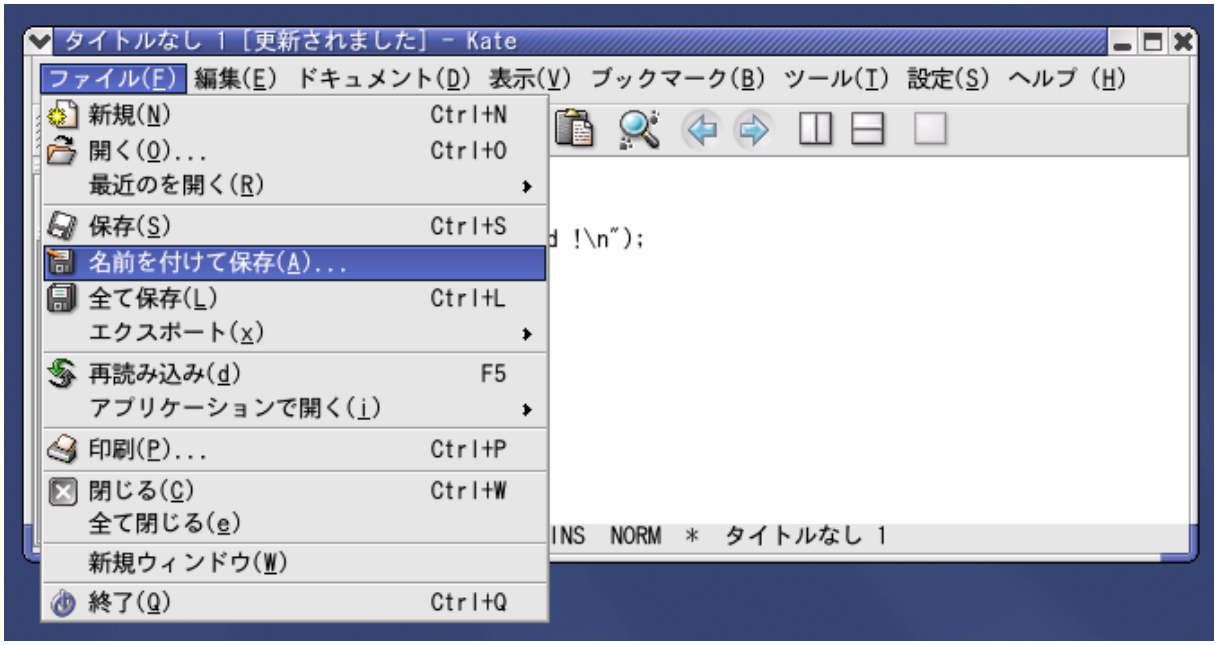
(d) プログラムの編集と保存

次にプログラムを編集します。C言語の説明は３章以降で行います。ここでは以下のプログラムを書いてください。ファイルの末尾に改行を入れないとエラーになるので注意してください。

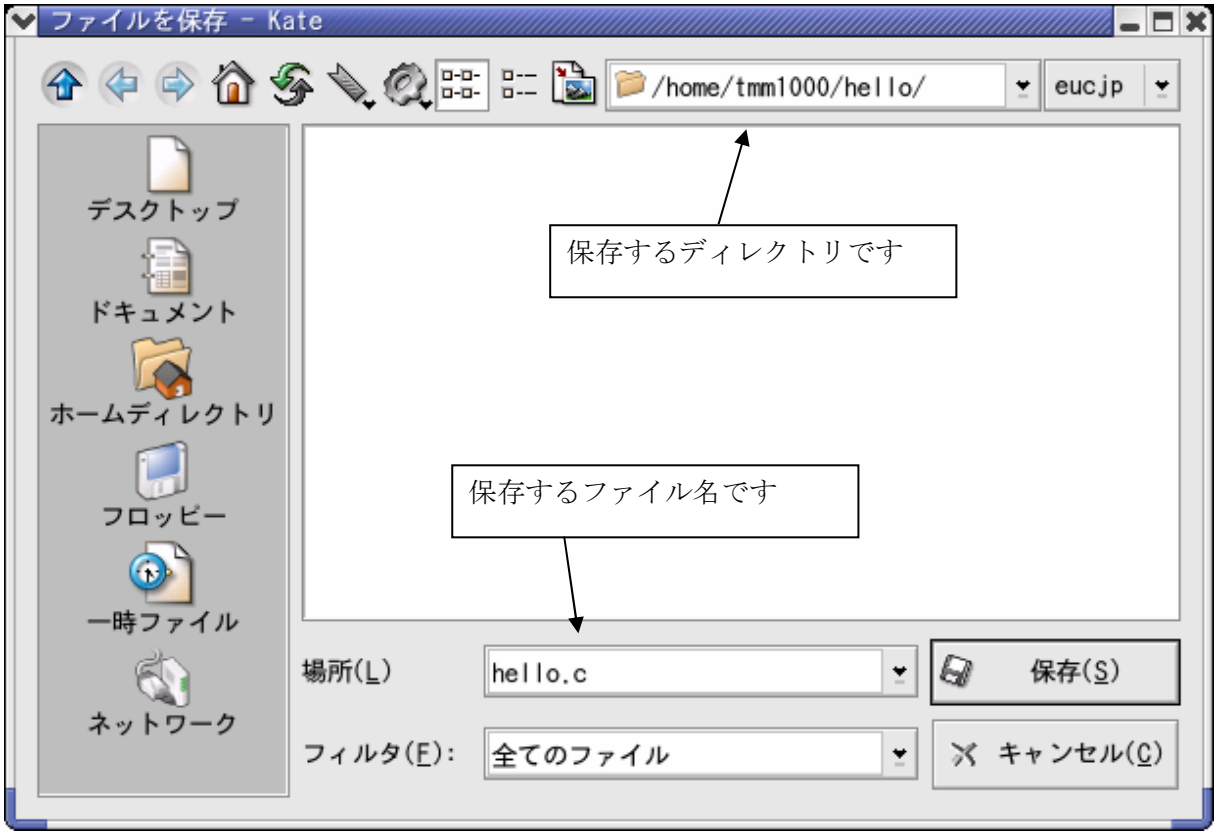
```
#include<stdio.h>
int main( )
{
        printf("Hello Wordl ! ＼n");
        return (0);
}
```

改行が必要です →

プログラムを書き終わったら保存してください。”**ファイル**”－ ”**名前を付けて保存**”をクリックしてください。



”**/home/tmm1000/hello**”に”**hello.c**”という名前で保存してください。



2-3.コンパイル（１）　－セルフ開発環境－

作成した”hello.c”をコンパイルします。Linux（x86 系）用の実行ファイル作成する場合とTMM1000（SH3）の実行ファイルを作成する場合ではコンパイルの方法が違います。Hello.cをコンパイルする場合はそれぞれ以下のコマンドになります。

Linux(X86 系)	gcc hello.c
TMM1000(SH3)	sh-linux-gcc hello.c

はじめに、Linux（x86 系）用の実行ファイルを作成して、Linux 上で実行します。

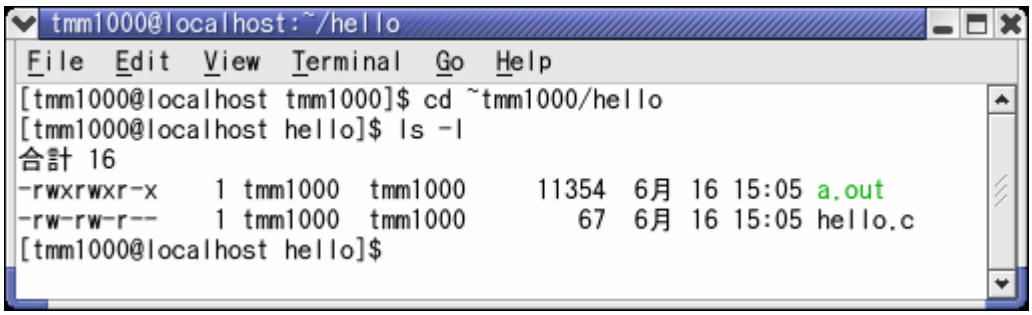
(a) カレントディレクトの確認　pwd

ここで、カレントディレクトリを確認してください。”**pwd**”はカレントディレクトリを表示するコマンドです。ターミナルに”**pwd**”と入力して、Enter を押してください。カレントディレクトリが”Hello.c”を保存したディレクトリ”**/home/tmm1000/hello**”になっているでしょうか。違う場合はターミナルに”cd ~tmm1000/hello”と入力し、Enter を押してカレントディレクトリを移動してください。

(b) カレントディレクトリ内のファイル情報の表示　ls

カレントディレクトリに”hello.c”が保存されているかを確認します。”**ls**”はカレントディレクトリ内のファイルの情報を表示するコマンドです。ターミナルに”**ls**”と入力し、Enter を押してください。”hello.c”が保存されていることが確認できましたでしょうか。もし、保存されていない場合はもう一度エディタで保存してください。

オプションで”**-l**”を付けるとファイルの詳細情報を表示できます。ターミナルに”**ls -l**”と入力して Enter を押してください。ディレクトリのファイルの詳細情報が表示されます。



（タイプ）	（モード）	（リンク数）	（所有者）	（グループ）	（サイズ）	（更新日時）	（ファイル名）
d	rwxr-xr-x	3	tmm1000	tmm1000	1024	May 1 21:27	Hello

タイプはファイルの種類を表し、“ー”は通常のファイル、“d”はディレクトリ、“l”はシンボリックリンク（Windows の言うショートカット）です。

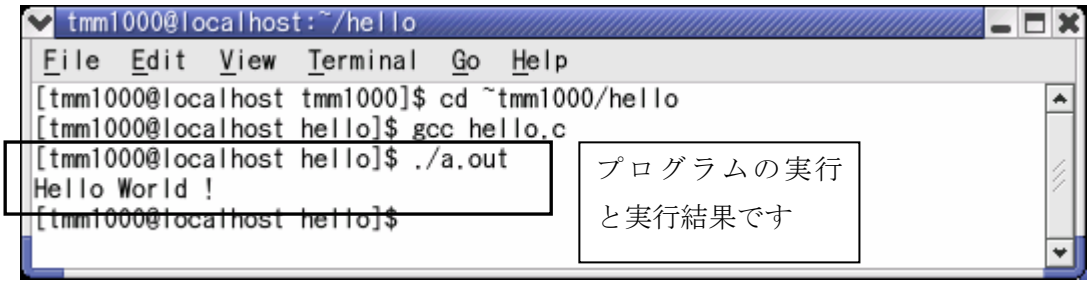
モードはファイルのアクセス権限を表し、最初の 3 文字が所有者に許可される操作、次の 3 文字はファイルが所属するグループに属するユーザに許可される操作、最後の 3 文字がその他のユーザに許可される操作です。**“r”は読み出し、“w”は書き込み、“x”は実行**です。“**rwxr-xr-x**” の場合は所有者が読み込み、書き込み、実行、ファイルが所属するグループに属するユーザとその他のユーザが読み込み、書き込みの権限を持つという意味です。

(c) プログラムのコンパイルと実行

Linux（x86 系）用の実行ファイルを作成するために、”**gcc hello.c**”と入力して Enter を押してください。コンパイルが正常に終わると”hello.c”と同じディレクトリに”**a.out**”というファイルが作成されます。

これを実行するにはターミナルに”**./a.out**”と入力して Enter を押してください。”Hello World !”と出力されます。

”a.out”以外の名前にするにはオプションで”-o”を付けて名前を指定します。”hello.out”という名前にするにはターミナルに”gcc -o hello.out hello.c”と入力して Enter を押してください。



この場合は Linux（x86 系）でソフトウェアを開発し、Linux（x86 系）でソフトウェアを動作させています。このようにソフトウェアの開発を、ソフトウェアを動作させるシステムで行うことを**セルフ開発環境**といいます。

2-4.コンパイル（２）　－クロス開発環境－

ターミナルに”**sh-linux-gcc hello.c**”と入力して Enter を押してください。2-3（c）節と同様に”**a.out**”というファイルが作成されます。2-3（c）節と同様にして”a.out”以外の名前にすることができます。

(a) コンパクトフラッシュを使用可能にする　mount

フロッピーディスク、コンパクトフラッシュなどの外部記憶装置は接続しただけでは使用できません。マウントという作業が必要になります。

はじめに**マウントポイントと呼ばれるディレクトリを作成します**。これはディレクトリ”**/mnt**”に作成されることが多いので、ここに作成します。

ターミナルに”**cd /mnt**”と入力して Enter を押して、カレントディレクトリを移動してください。次に、ターミナルに”**mkdir cf**”と入力し、Enter を押して、マウントポイント”**cf**”を作成してください。

“**mount**”はあるデバイス（フロッピーディスク、コンパクトフラッシュなど）をマウントポイントから読み書きできるようにするコマンドです。**これはスーパーユーザでないと実行できないコマンドです。**

mount　デバイス　マウントポイント

TMM1000 では１つのコンパクトフラッシュを 3 つの領域に区切って使用しています。これらの１つ１つの領域をパーティションといいます。

第１パーティションは Windows でも読み書きできます。第 2 パーティションには OS（Linux）が入っています。第 3 パーティションは主にアプリケーションを入れるのに使用されています。

よって、第 3 パーティションに作成した”a.out”をコピーします。ターミナルに”**mount /dev/sda3 /mnt/cf**”と入力して Enter を押してください。これでコンパクトフラッシュの第 3 パーティションがマウントポイント”**/mnt/cf**”で読み書きできるようになります。

＊第 1 パーティション、第 2 パーティションは以下の通りです。 第 1 パーティション　　/dev/sda1　　第 2 パーティション　　/dev/sda2

(b) 実行ファイル”a.out”のコンパクトフラッシュへのコピー　cp

作成した”a.out”をコンパクトフラッシュへコピーします。”**cp**”はファイルをコピーするコマンドです。

cp　コピー元ファイル　コピー先ディレクトリ

ターミナルに”**cp a.out /mnt/cf/bin**”と入力し、Enter を押して、”a.out”を”/mnt/cf/bin”にコピーしてください（第 3 パーティションのディレクトリ”bin”にアプリケーションを入れることが多いからです)。

ディレクトリをコピーする場合はオプションで“**-R**”を付けます。

cp –R コピー元ディレクトリ　コピー先ディレクトリ

(c) コンパクトフラッシュを安全に取り外す　umount

フロッピーディスク、コンパクトフラッシュなどはアンマウントをしないと安全に取り外すことができません。”**umount**”はアンマウントを行うコマンドです。**これはスーパーユーザでないと実行できないコマンドです。**

umount　マウントポイント

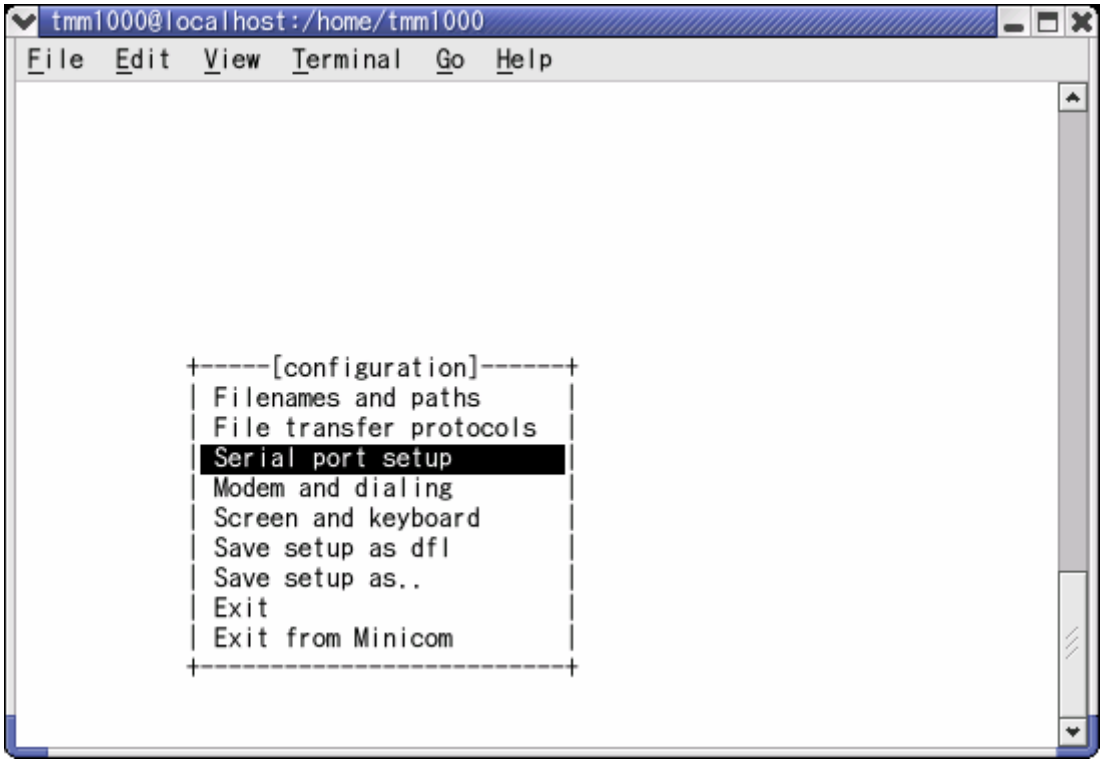
ターミナルに”**umount /mnt/cf**”と入力して Enter を押してください。この後にコンパクトフラッシュを取り外してください。

「デバイスを使用中です」というエラーが出る場合があります。この原因の１つにカレントディレクトリがマウントポイント、もしくはその中のディレクトリになっているということがあります。その場合はカレントディレクトリを移動してください。（例えば”cd ~tmm1000”）

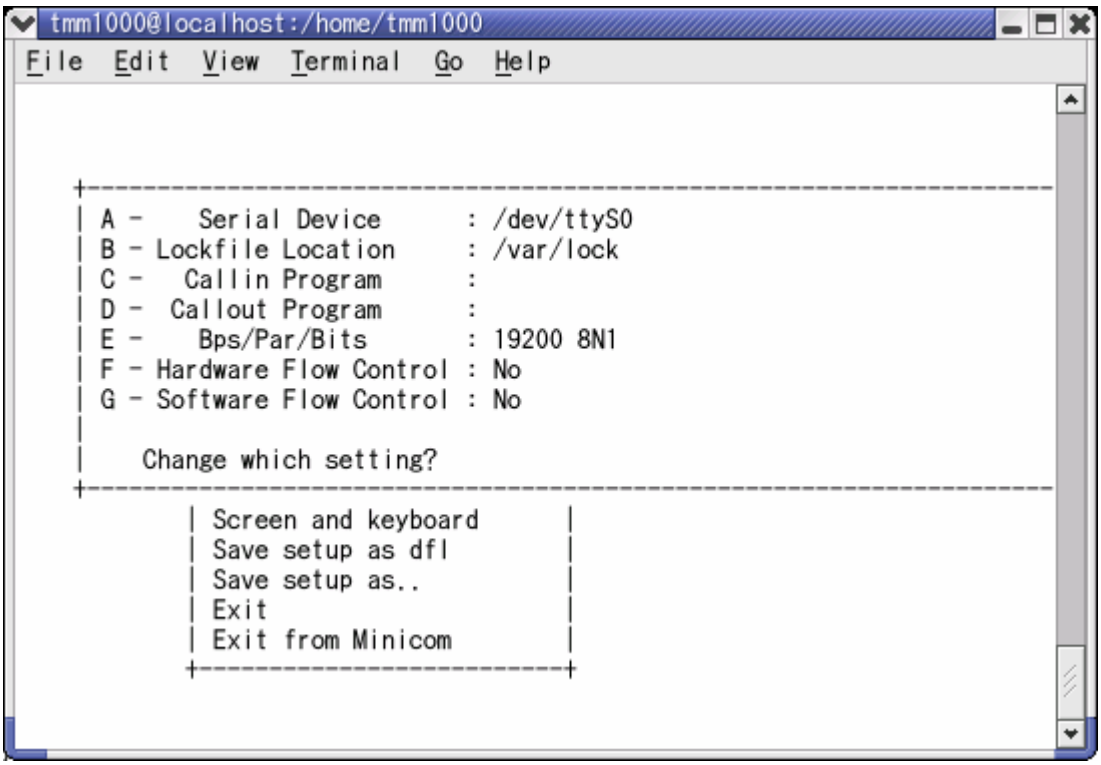
(d) シリアルポート経由での TMM1000 との通信　　minicom の設定

TMM1000 とパソコンを RS232C のクロスケーブルで接続してください。シリアルポートで TMM1000 と通信するために、”minicom”というアプリケーションを使います。

スーパーユーザになり、ターミナルに”minicom –s”と入力して Enter を押し、minicom の設定画面を出してください



“**Serial port setup**”を選択し、Enter を押してください。



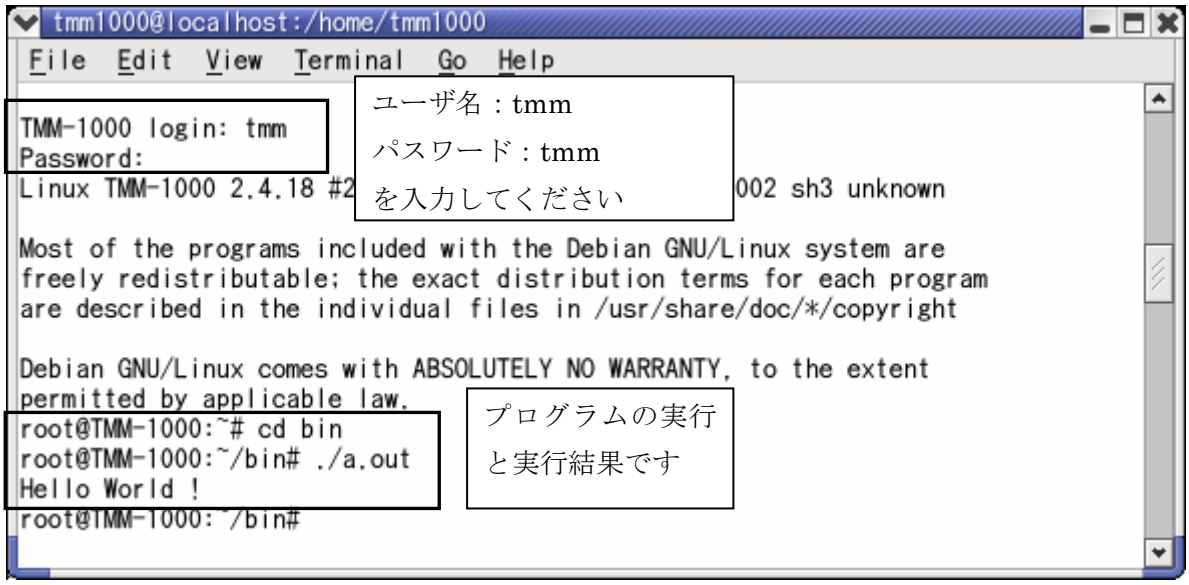
上図のように設定を変更してください。設定の変更が終了したら Enter を押して、はじめの画面に戻ってください。”**Save setup as dfl**”を選択して Enter を押して、設定を保存してください。

“**Exit from Minicom**”を選択して Enter を押し、minicom を終了させてください。また、”**Exit**”を選択して、Enter を押した場合は、設定を終了し、”minicom”になります。”minicom”を終了させるには”**Ctrl+A**”を押してから”x”を押します。”**Leave Minicom?**”というメッセージが出るので、”**Yes**”を選択し、Enter を押してください。

(e) TMM1000 上でのプログラムの実行

データをコピーしたコンパクトフラッシュを TMM1000 にセットして、電源を入れてください。　スーパーユーザになり、ターミナルに”**minicom**”と入力し、Enter を押し、minicom を起動してください。

はじめに TMM1000 にログインします。**ユーザ名：tmm、パスワード：tmm** を入力してログインしてください。次に”**cd bin**”と入力し、Enter を押して、”a.out”をコピーしたディレクトリに移動してください。”**./a.out**”と入力し、Enter を押して、プログラムを実行してください。”Hello World !”と出力されます。

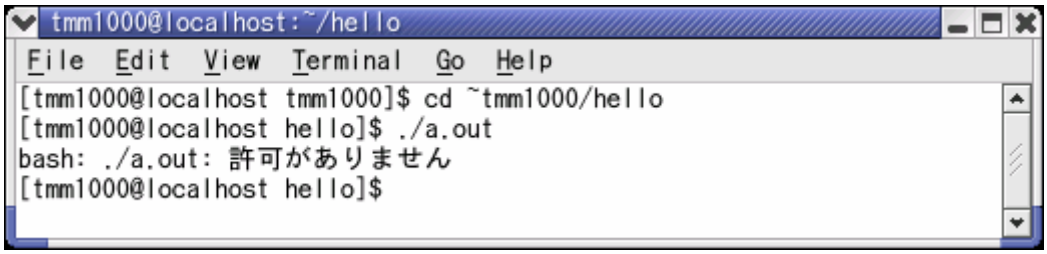


この場合は Linux（x86 系の CPU）でソフトウェアを開発し、TMM1000（SH3 の CPU）でソフトウェアを動作させています。このようにソフトウェアの開発を、ソフトウェアを動作させるシステムとは違うシステムで行うことを**クロス開発環境**といいます。

2-5.ファイルのアクセス権限と所有者・所属するグループ　chmod、chown

(a) ファイルのアクセス権限の変更　chmod

プログラムを実行したときに下図のように“許可がありません”というエラーが出る場合があります。



これにはファイルのアクセス権限が関係しています。2-3（b）節で述べたようにファイルに誰がどのようなアクセス権限（読み出し、書き込み、実行）を持つかを定めています。この中で<u>実行する権限を持たせることを「実行ビットを立てる」と言います</u>。

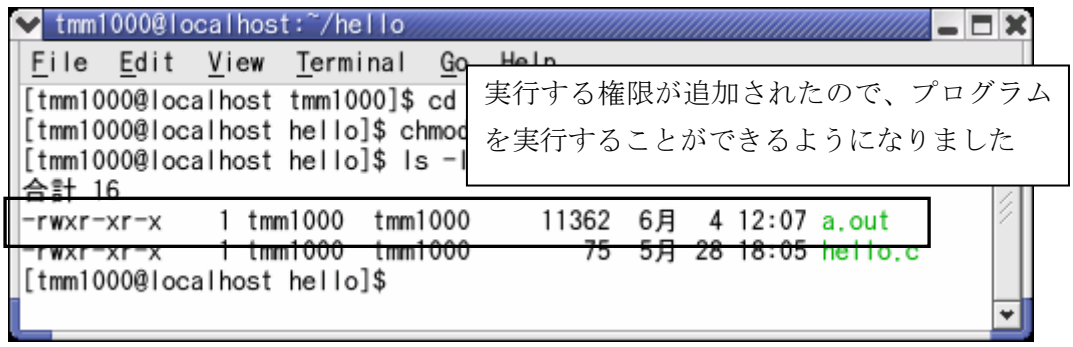
ターミナルに”**ls -l**”と入力して Enter を押して、ファイルの所有者、グループ、アクセス権限を確かめてください。

ファイルの所有者が”**root**”、グループが”**root**”でアクセス権限が”**rwxr-xr—**“の場合、そのファイルは所有者（ユーザ”root”）とファイルが所属するグループ（”root”に属するユーザしか実行することができません。

例えば、ユーザ”tmm”（所属グループ”tmm”）はファイルの所有者でもファイルが所属するグループのユーザでもないので、このファイルを実行することができません。この場合はその他のユーザのアクセス権限に”x（実行）”を持たせれば実行できるようになります。

ただし、スーパーユーザは全ての権限を持つユーザなので、全てのユーザに実行が許可されない場合（例えばアクセス権限が”---------“など）を除いてはファイルを実行できます。

アクセス権限に”x（実行）”を追加するために、ターミナルに”**chmod 755 a.out**”と入力して Enter を押してください。



“**chmod**”はアクセス権限を変更するコマンドです。記号と 3 桁の 8 進数の数字による設定方法があります。記号による設定方法は以下の通りです。

chmod [ugoa] [＋－＝] [rwx]

[ugoa] は実行権限を変更する対象を表します。**[＋－＝]** は後に記述された実行権限をどのようにするのかを表します。最後の **[rwx]** は実行権限を表します。

[ugoa]

u	所有者
g	ファイルが所属グループに属するユーザ
o	その他のユーザ
a	すべて

[＋－＝]

＋	後に記述した権限を追加します
－	後に記述した権限を削除します
＝	後に記述した権限に設定します

[rwx]

r	読み出し
w	書き込み
x	実行

3 桁の 8 進数の数字を用いた設定方法は以下の通りです。

chmod [755]

左から所有者、グループ、その他の実行権限を表します。数字と実行権限の関係は以下の通りです。

0	000	---
1	001	--x
2	010	-w-
3	011	-wx

4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

先ほどファイル“a.out”のアクセス権限を、“**rwxr-xr-x**”（所有者が読み込み、書き込み、実行、グループとその他が読み込み、書き込み）に変更しました。これは以下の 2 通りのコマンドで実行できます。

chmod u=rwx,g=rx,o=rx a.out
chmod 755 a.out

(b) ファイルの所有者・所属するグループの変更　chown

“**chown**”はファイルの所有者とグループを変更するコマンドです。**このコマンドはスーパーユーザでないと実行できません。**

以下のように所有者とグループ名を指定します。所有者のみを変更する場合は“**:（グループ名）**”を省略してください。逆にグループ名のみを変更する場合は所有者を省略し“**:（グループ名）**”のみを入力してください。

chown （所有者）:（グループ）

2-6.ファイルの移動･名前の変更　mv

ファイルの移動または、名前の変更を行うコマンドです。

ファイルの移動を行う場合：**mv 移動するファイル　移動先ディレクトリ**
ファイルの名前を変更する場合：**mv　名前を変更するファイル　変更後のファイル名**

2-4（b）節では”cp”コマンドを使ってファイルをコピーしましたが、ここでは”mv”コマンドを用いてファイルを移動する方法を書きます。

ファイル“**a.out**”をディレクトリ“**/mnt/cf/bin**“に移動する場合は、ターミナルに“**mv a.out /mnt/cf/bin**“と入力して Enter を押します。

コンパイルして作成されたファイルの名前はオプションを付けて名前を指定しない限り、“**a.out**”で固定です。これではファイルの内容が分かりづらいので、名前を変更することを考えます。

“**a.out**”の名前を“**hello.out**”に変更する場合は、ターミナルに“**mv a.out hello.out**“と入力して Enter を押します。

2-7.ファイルの削除　rm

ディレクトリ、ファイルを削除します。

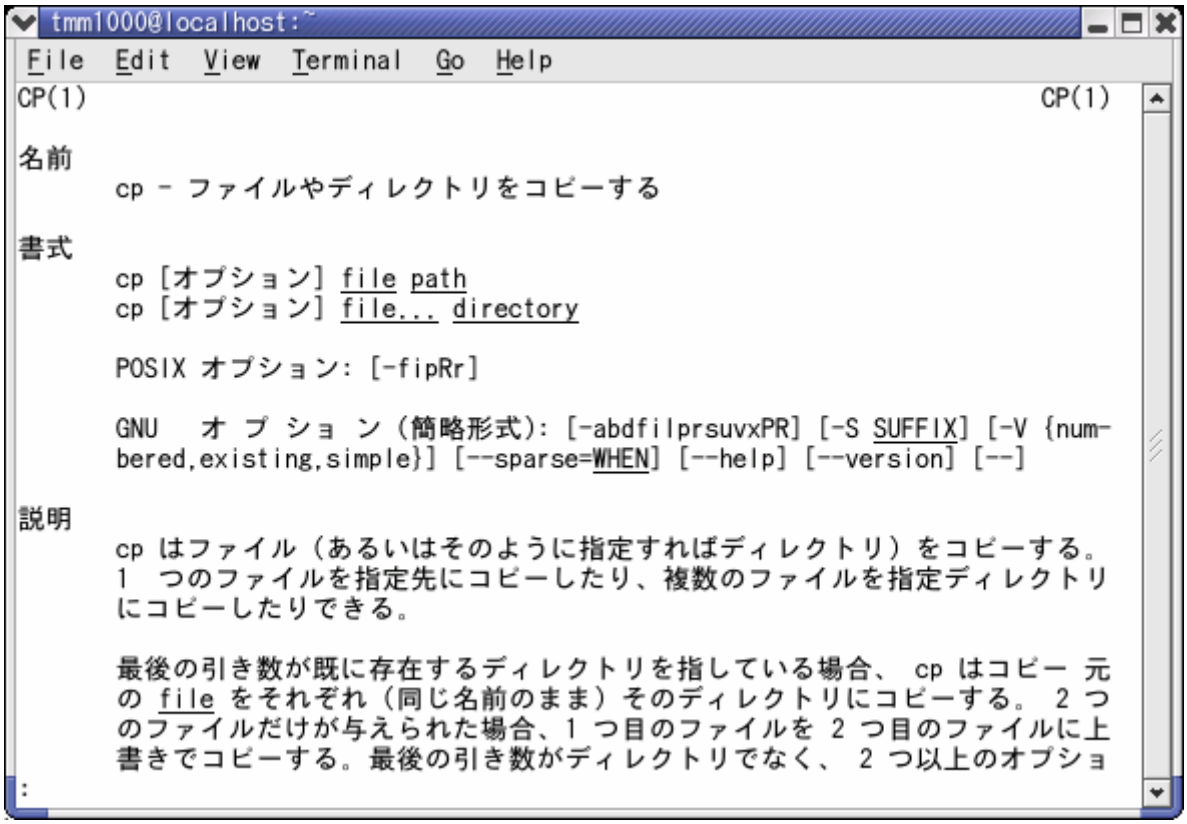
ファイル“**a.out**”を削除する場合は、ターミナルに“**rm a.out**“と入力して Enter を押します。ディレクトリとその中身を全て削除する場合はオプションで“**-r**”を付けます。ディレクトリ“**hello**”を削除する場合はターミナルに“**rm -r hello**“と入力して Enter を押します。

2-8.オンラインマニュアルの表示　man

オンラインマニュアルを表示するコマンドです。cp コマンドのマニュアルを表示するにはターミナルに”man cp”と入力して Enter を押してください。

キーボードの”↑”、”↓”で上下にスクロールします。”**q**”を入力すればターミナルに戻ります。

Linux のコマンドの他に C 言語の printf 文などのオンラインマニュアルもあります。



３.変数と配列

3-1.変数

（a）変数の型

変数はその名の通り様々な値を取ることができます。変数は計算を行うときに使用したり、for 文などの繰り返し文のカウンタに使用したり、条件分岐の条件に使用したりと様々な使い方があります。

変数には整数を表すもの、小数（浮動点少数）を表すもの、文字を表すものなどがあります。これらを変数の型といいます。基本的な変数の型を以下に示します。

型	データの種類	byte 数
char	文字	1
int	整数	2 or 4
float	単精度浮動点小数	4
double	倍精度浮動点小数	8

int 型はコンピュータ、コンパイラによって、2 byte になったり 4byte になったりするので、int 型は移植する際にトラブルの原因になります。よって、long int 型、short int 型 を使うほうが好ましいです。long int 型は 4byte、short int 型は 2byte です。それぞれ long、short と省略できます。

(b) 変数の宣言と初期化

変数を使うためには使う変数を宣言する必要があります。変数の宣言は以下のようにします。

変数の型　変数名;

int i;　int 型の変数 i を宣言する場合

このままでは、変数 i の値は定まっていません。つまり変数 i の値はどのようになっているのかわかりません。4-1 節の printf 文を用いて表示して確認してみてください。

この状態ではプログラムを正常に動作させることはできないので、**初期化を行い変数の値を定めます。**このときの変数の値を**初期値**といいます。

変数の初期化は変数を宣言するのと同時に行います。

変数の方　変数名　=　初期値

int i = 0;　　整数型の変数 i を宣言すると同時に初期値 0 で初期化する場合

また、宣言した変数を一度も使用しないと警告が出ます。コンパイラによってはコンパイルが完了し、実行ファイルが作成されますが、エラーになり、コンパイルが完了しない場合もあります。

3-2.配列の宣言と初期化

同じ種類のデータを取り扱う場合変数名を a1、a2、･･･、a100 と宣言するのは大変です。そこで、配列を使います。配列の宣言は以下のように行います。

データ型　配列名[要素数]

int a[100];　　100 個の要素を持つ int 型の配列 a を宣言する場合

これで、下図のように a[0],a[1]、･･･、a[99]という 100 個の変数を用意できます。a[0]、a[1]などの配列の１つ１つのことを要素と言います。

a[0]	a[1]	･･･	a[99]

配列の初期化は以下のようにします。

データ型　配列名[要素数]={要素、要素、要素、･･･}

int a[3] = {1,3,5};

これで、a[0]を 1 で、a[1]を 3 で、a[2]を 5 で初期化されます。また、以下のようにすることもできます。

int a[3] = {0};

このようにすると、a[0]を 0 で初期化し、**残りの値を指定していない要素は全て 0 で初期化されます**。

3-3.文字列の取り扱い（char 型の配列）

C 言語では文字列を扱うのに char 型の配列を使います。また、文字列の最後には NULL 文字（＼0）が付くので、配列の数は“文字列＋１”用意する必要があります。なお、”＼”は Windows では”¥”になります。

char 型の配列は 3-2 節の方法以外にも以下のようにして初期化できます。

```
char 変数名[要素数]＝”初期化する文字列”
char strBuff[256] = “Hello World !”;
```

これで、256 個の要素を持つ char 型の配列 strBuff を宣言し、”Hello World !”で初期化することができます。

memset 関数を用いても初期化することができます。これは標準ライブラリ関数なので、プログラムのはじめに、”#include <string.h>”と記述する必要があります。標準ライブラリ関数については 4-1 節を参照してください。

```
strBuff[256];
memset(strBuff,0x00,256);
```

これで、strBuff の 256 個の要素全てに”0x00”を代入することができます。“0x00”の”0x”は 16 進数という意味です。”0x00”は 16 進数の”00”で、NULL 文字を表します。

memset 関数を説明するにはポインタについて説明する必要がありますが、本書ではポインタの説明を行っていないので、詳しい説明は省略させていただきます。

4.”Hello World !”と表示するプログラム

4-1.printf 関数を用いたプログラム（標準ライブラリ関数）

文字を表示するために **printf 関数**を使います。これはコンソール（コンピュータの入出力機器のことで、キーボードとディスプレイが一般的です）に文字を出力する関数です。

printf(“書式指定文字列”,変数,・・・)

書式指定文字列ですが、これには 3 種類あります。＼（Windows の場合は¥）ではじまるもの。％ではじまるもの。その他です。

＼ではじまるものは改行、タブなどを表します。“＼n”は改行を“＼t”はタブを表します。他にも種類がありますが、よく使うのはこの２つです。

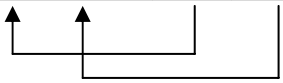
％ではじまるものは変数を指定の形式に変換して表示します。

%d	10 進数の整数
%x	16 進数の整数
%o	8 進数の整数

%f	浮動点小数
%s	文字列
%c	文字

int 型の変数 a の値が 100 の場合以下のように記述すると、10 進数と 16 進数に変数 a の値が変換され、”100　64”と表示されます。

```
printf(“%d  %x  “ ,  a  ,  a  );
```



その他のものは普通の文字です。これはそのまま表示されます。

printf 関数は<u>標準ライブラリ関数</u>と呼ばれるものです。これを使用するにはプログラムのはじめに、”**#include <stdio.h>**”と記述する必要があります。

<u>標準ライブラリ関数は使用するときに”#include <～>”と記述する必要があります。～は使用する関数によって違います。</u>

2 章で示したサンプルを例として示します。これは”Hello World !”とコンソールに表示するプログラムです。出力結果は 2-4（E）節を参照してください。

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

4-2.sprintf 関数を用いたプログラム

次は文字列（char 型配列）を用いて”Hello World !”を表示するプログラムを作成します。char 型の配列 strBuff を用意し、これに文字列を書き込み、printf 関数で出力するプログラムを作成します。

sprintf 関数は char 型配列に文字列を書き込む関数です。これは標準ライブラリ関数なので、プログラムのはじめに、”**#include <stdio.h>**”と記述する必要があります。

sprintf(char 型配列名,”文字列”);

sprintf(strBuff,”Hello World!”);
文字列方の配列 strBuff に”Hello World !”と書き込む場合

サンプルプログラムを以下に示します。

```
#include<stdio.h>
int main( )
{
      char strBuff[14];
      memset(strBuff,0x00,14);
      sprintf(strBuff,”Hello World !”);
      printf(“%s＼n”,strBuff);

      return (0);
}
```

4-3.繰り返し処理　for 文を用いたプログラム

ある条件を満たす間ある処理を繰り返す場合に for 文を用います。

```
for( 式１ ; 式２ ; 式３){
      繰り返したい処理
}
```

式１：初期値を入力します
式２：この条件を満たす間繰り返します
式３：{ } 内の処理の最後に実行する式です

条件式は以下の通りです。条件式を満たしている場合を“真”といい、満たしていない場合を“偽”といいます。

a = = b	a と b が等しい
a > b	a は b より大きい
a < b	a は b より小さい

a >= b	a は b 以上
a <= b	a は b 以下

２つの条件式の“かつ”と“または”を使うこともできます。

(条件式1)&&(条件式2)	かつ(条件式1)と(条件式2)が共に真ならば真
(条件式1)‖(条件式2)	または(条件式1)と(条件式2)のどちらかが真ならば真

for 文を用いたプログラム例

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

int main( )
{
      int i = 0;
      char strBuff[14];
      memset(strBuff,0x00,14);

      for( i = 0 ; i <= 2 ; i++){
            sprintf(strBuff,”Hello World !”);
            printf(“i = %d＼t%s＼n”,i,strBuff);
      }
      return (0);
}
```

for 文の動作について説明します。変数 i の初期値は 0 で、i が 2 以下の場合{ }内の処理を繰り返します。{ }内の処理の最後に変数 i をインクリメント（1 を加える）します。つまり、この場合は{}内の処理を 3 回繰り返します。これを実行すると以下のようになります。

```
i = 0    Hello World !
i = 1    Hello World !
i = 2    Hello World !
```

”i++”ですが、これは変数 i をインクリメントする（１を加える）という意味です。また、”i--”は変数 i をデクリメント（１を引く）です。

4-4.繰り返し処理　while 文を用いたプログラム

条件式の条件を満たす間｛ ｝内の処理を繰り返します。

```
while( 条件式 ){
    繰り返したい処理
}
```

while 文を用いたプログラム例

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

int main( )
{
    int i = 0;
    char strBuff[14];
    memset(strBuff,0x00,14);

    while( i <= 2 ){
        sprintf(strBuff,"Hello World !");
        printf("i = %d＼t%s＼n",i,strBuff);
        i++;
    }
    return (0);
}
```

while 文の動作について説明します。i が 2 以下の場合{ }内の処理を繰り返します。i は初期値 0 で初期化し、while 文の最後で i をインクリメントしています。よって、この場合は{}内の処理は 3 回繰り返します。出力結果は 4-3 節と同様です。

4-5.繰り返し処理から抜ける　break 文

繰り返し文（for,while など）、switch 文から強制的に抜けるときに使います。”**break;**”と記述したらその場所で繰り返し文（for 文,while 文など）、switch 文から抜けます。

break 文を用いたプログラム例

```
#include<stdio.h>
int main( )
{
      int i = 0;
      while( i <= 2 ){
            printf(“before＼n”);
            break;
            printf(“after＼n”);
            i++;
      }
      return (0);
}
```

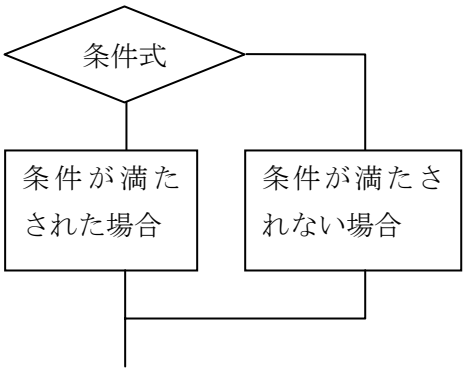
実行すると”before”のみが表示されます。

”before”を表示する printf 文の後に break 文で while 文から抜けているので、break 文の後の”after”を表示する printf 文は実行されません。

4-6.条件分岐　if 文

条件式を満たすかどうかで分岐します。else 以下を省略すると条件を満たさない場合は何も処理をしないという意味になります。

```
if( 条件式 ){
    実行文（条件が満たされた場合）
}else{
    実行文（条件が満たされない場合）
}
```



if 文を用いたプログラム例

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

int main( )
{
      int i = 0;
      char strBuff[14];
      memset(strBuff,0x00,14);

      for( i = 0 ; i <= 2 ; i++){
            if( i <= 1 ){
                  sprintf(strBuff,"Hello World !");
                  printf("i = %d＼t True＼t %s＼n",i,strBuff);
            }else{
                  printf("i = %d＼t false＼n",i);
            }
      }
      return (0);
}
```

for 文によって変数 i が 0 から 2 まで変化します。変数 i が 1 以下がどうかで分岐します。

実行結果

```
i = 0     True     Hello World !
i = 1     True     Hello World !
i = 2     false
```

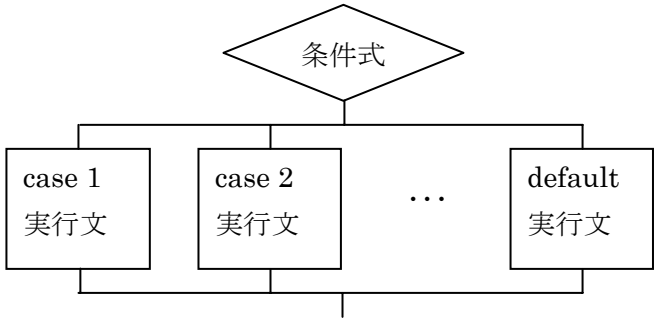
i が 1 以下で条件式を満たしている場合

i が 1 より大きく、条件を満たしていない場合

4-7.条件分岐　switch～case 文

式の値に一致する case 文を実行します。break 文を省略するとそれ以降の全ての実行文を実行してしまいます。

```
switch( 式 ){
    case　定数式１：
        実行文;
        break;
    case　定数式２ ：
        実行文;
        break;
            ・
            ・
            ・
    default：実行文;
}
```



switch～case 文を用いたプログラム例

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

int main( )
{
      int i = 0;
      char strBuff[14];
      memset(strBuff,0x00,14);

      for( i = 0 ; i <= 3 ; i++){
            switch ( i ){
                  case 0 :
                        sprintf(strBuff,"Hello World !");
                        printf("i = %d＼t case 0＼t %s＼n",i,strBuff);
                        break;
                  case 1 :
                        printf("i = %d＼t case 1＼n",i);
                        break;
                  default :
                        printf("i = %d＼t default＼n",i);
                  }
      }
      return (0);
}
```

変数 i の値が 0 のときは case 0 を、1 のときは case 1 をそれ以外の場合は default を実行します。

実行結果

```
i = 0     case 0     Hello World !
i = 1     case 1
i = 2     default
i = 3     default
```

i が 0,1 以外の場合は default を実行します

4-8.関数

４章のプログラムは全て main 関数（必ず必要な関数で、一番はじめに実行される関数です）のみで記述しました。４章のような小規模なプログラムなら良いのですが、ある規模以上のプログラムを main 関数だけで記述しようとすると理解しづらいプログラムになってしまいます。

また、似たような処理を行う場合は関数としてその処理をまとめれば理解しやすいプログラムになります。

関数には引数と戻り値があります。関数は渡された引数を用いて処理を行い戻り値を返します。”a+b”を計算しその値を返す関数”add”は以下のようになります。

```
int add(int a,int b)
{
      return a+b;
}
```

変数 c に関数 add を用いて、3 と 4 の和を代入する場合は以下のように記述します。

```
c = add(3,4);
```

プログラムのはじめに以下のように記述することを関数のプロトタイプ宣言と言います。これを行うとコンパイラが引数と戻り値のチェックを行います。

```
int add(int a,intb);
```

関数を使うためには使われる前にその関数本体が記述されているか、プロトタイプ宣言をされている必要があります。例１のように main 関数の前に関数”add”が記述されています。この場合はプロトタイプ宣言を行わなくてもエラーになりません。しかし、例２のように main 関数の後に関数”add”が記述されている場合はエラーになります。

関数”add”を main 関数の前に記述した場合

```
#include<stdio.h>

int add(int a,int b);        //関数のプロトタイプ宣言は省略しても平気です

int add(int a,int b)
{
      return a+b;
}
int main( )
{
      int c;
      c = add(3,5);
      printf(“%d＼n”);
      return (0);
}
```

関数”add”を main 関数の後に記述した場合

```
#include<stdio.h>

int add(int a,int b);        //関数のプロトタイプ宣言が必要です

int main( )
{
      int c;
      c = add(3,5);
      printf(“%d＼n”);
      return (0);
}

int add(int a,int b)
{
      return a+b;
}
```

4-9.定数

定数は変数と違い、途中で値を変化させることができません。定数の宣言は以下のように行います。

```
#define 定数名　値
```

```
#define PI 3.1415　　定数名：PI　　値：3.1415
#define DeviceName "/dev/ttySC1"　定数名：DeviceName　　値："/dev/ttySC1"
```

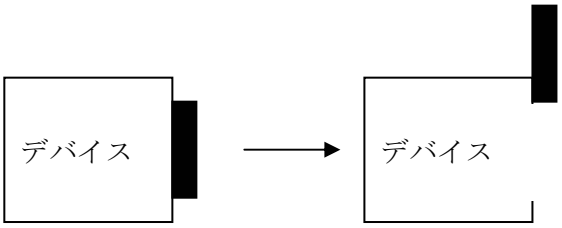
定数を使うメリットは保守が容易になること、定数の値の意味が明確になるということがあります。定数を使わないで記述した場合は、値を修正するときに全ての箇所の修正をする必要があります。定数を用いた場合は定数の宣言部分を修正するだけで済みます。

また、”3.1415”より定数を用いて”PI”と表示した方が、”/dev/ttySC1”より定数を用いて”DeviceName”とした方が値の意味が明確になります。

5.シリアルポートからの入出力

5-1.デバイスのオープン

デバイスはオープンしないと使用することができません。オープンしていないデバイスは扉が閉まっているようなものです。この状態ではデータの読み書きはできません。デバイスをオープンすれば、データの読み書きができるようになります。



(a) サンプルプログラムでのファイルの開き方

サンプルプログラム" **sample.tar.gz**"を展開すると"**disp**"、"**serial**"、"**print**"ができます。この中のディレクトリ"**serial**"に保存されているサンプルプログラム（"sample.c"）を開いてください。

サンプルプログラムの 31 行目から 67 行目に"**tty_init( )**"という関数が記述してあります。これは定数"**DeviceName**"のデバイスをオープンする関数です。デバイスが正常にオープンされた場合に 0 を返します。また、オープンしたあとは **ttyd** という名前でデバイスをコントロールできます。

サンプルプログラム内では以下のように if 文を使ってデバイスが正常にオープンできたかどうかをチェックしています。

```
if( tty_init( ) == 0 ){
～～～～～～～～～～
～～～～～～～～～～
}else{
            //!< デバイスのオープンに失敗
printf("¥n------- DeviceOpenError -------¥n");
ans = -1;
          }
```

(b) open 関数

デバイス"/dev/ttySC1"を読み書き用でオープンには以下のようにします。以下のようにopen 文を使用すると"ttyd"という名前でデバイスをコントロールできるようになります。

```
int ttyd;
ttyd = open(/dev/ttySC1, O_RDWR | O_NOCTTY | O_NONBLOCK );
```

open(デバイス名・ファイル名 , フラグ , モード)

＊複数のモードを記述する場合は"|"で区切ります

"open"はファイルのオープンに失敗すると－1 を返します。

フラグに記述するフラグの一部を以下に述べます。

O_RDWR は読み書き用という意味です。O_RDONLY、O_WRONLY がそれぞれ読み込み専、書き込み専用という意味です。この 3 つはどれか 1 つのみ記述します。

O_NOCTTY はターミナルデバイスの場合で、プロセス（OS から見たときに、実行中の１つの１つのプログラムをプロセスと言います）が制御端末を持たない場合でもそれが制御端末にならないということです。

O_NONBLOCK ファイルを非停止（non-blocking）で開きます。

O_CREAT はファイルが存在しなかった場合は作成します。このときに作成するファイルのアクセス権限をモードで指定します。

O_CREAT を使用しなかった場合はこのモードは省略します。モードを以下に示します。

S_IRWXU	所有者は読み込み、書き込み、実行の許可を持ちます
S_IRUSR (S_IREAD)	所有者は読み込みの許可を持ちます
S_IWUSR (S_IWRITE)	所有者は書き込みの許可を持ちます
S_IXUSR (S_IEXEC)	所有者は実行の許可を持ちます
S_IRWXG	グループに属するユーザは読み込み、書き込み、実行の許可を持ちます
S_IRGRP	グループに属するユーザ読み込みの許可を持ちます
S_IWGRP	グループに属するユーザ書き込みの許可を持ちます
S_IXGRP	グループに属するユーザ実行の許可を持ちます
S_IRWXO	その他のユーザは読み込み、書き込み、実行の許可を持ちます
S_IROTH	その他のユーザは読み込みの許可を持ちます
S_IWOTH	その他のユーザは書き込みの許可を持ちます
S_IXOTH	その他のユーザは実行の許可を持ちます

5-2.デバイスのクローズ

オープンとは逆の操作で、デバイスを使用できなくします。デバイスを使用した後は必ずクローズする必要があります。**ttyd** でオープンしたデバイスをクローズするには以下のようにします。

```
close( ttyd );
close(デバイス名);
```

サンプルプログラムの 19 行目から 30 行目に”**tty_close( )**”という関数が記述してあります。これは定数”**DeviceName**”のデバイスをクローズする関数です。

5-3.デバイスからデータを読み込む　read 関数

デバイスからデータを読み込むために read 関数を用います。5-2 節の例ではデバイス”/dev/ttySC1”を”ttyd”で操作できるようにしました。このデバイスから 1 byte のデータを文字列 strInput 読み込む場合は以下のようにします。

```
read( ttyd, strInput, 1);
read(デバイス名,char 型配列名,読み込む byte 数);
```

また、read 関数は読み込んだ byte 数を返します。

5-4.デバイスへの書き込み　write 関数

(a) 文字列の byte 数を取得する　strlen 関数

デバイスに書き込む場合は write 関数を使いますが、このときに書き込む byte 数を指定する必要があります。**strlen 関数**は文字列の byte 数を戻り値として返します。これは標準ライブラリ関数なので、プログラムのはじめに、”**#include <string.h>**”と記述する必要があります。

文字列”strBuff”の長さを求めるには以下のようにします。文字列の byte 数を格納する変数”len”を用意します。

```
int len = 0 ;
len = strlen( strBuff );

strlen(char 型配列名);
```

(b) デバイスへの書き込み

デバイス"ttyd"に文字列"strBuff"を書き込む場合は以下のようにします。はじめに書き込む byte 数を求めます。

```
int len = 0;
len = strlen( strBuff );
write( ttyd, strBuff, len );
```

write(デバイス名,char 型配列名,書き込む byte 数);

この write は read と同様に書き込んだ byte 数を返すので、以下のように、strBuff の byte 数と書き出した byte 数が等しいかをチェックすることで、データが正常に書き出せたかどうかをチェックできます。

```
if( write( ttyd, strBuff, len ) != len ){
        printf( "------- Write Error -------¥n" );
        ans = -1;
}
```

5-5.プログラム例

(a) プログラム例の概要

このプログラムはキーボードから文字を 1 文字入力して（入力した後に Enter を押してください)、それが数字の場合は”You hit （入力された数字）”を表示し、数字以外の文字の場合は”Not Number”と表示するものです。

はじめにサンプルプログラムの 11 行目と 12 行目、45 行目～54 行目までを”**#if 0**”と”**#endif**”でくくり、コンパイルしないようにしてください。

11 行目と 12 行目

```
#if 0
static   struct   termios oldtio;
static   struct   termios newtio;
#endif
```

45 行目から 54 行目

```
#if 0
tcgetattr( ttyd,&oldtio ); /* save current port settings */
    //!< 新しいポートの設定をカノニカル入力処理に設定する

    ～～～～～～～～～～～～～
    ～～～～～～～～～～～～～

    tcflush( ttyd, TCIFLUSH);
    tcsetattr( ttyd, TCSANOW, &newtio );
#endif
```

(b) 無限ループを使ってシリアルデバイスからの入力を待つ方法

シリアルデバイス（今回のプログラムではキーボード）からの入力を待つために、無限ループを使います。while(1)とありますが、このようにすると常に条件式を満たしている状態になり、無限に{}内の処理を繰り返します。

キーボードからの入力有無の判断は read 関数が読み込んだ byte 数を返すことを利用して判断します。read 関数の値が 1 以上の場合、break 文で無限ループから抜けます。

```
while(1){
    if( read( ttyd, strInput, 1) >= 1 ){
        break;
    }
}
```

以下にプログラム例を示します。サンプルプログラムを変更して作成したので、main 関数のみを示します。

```
int main( )
{
    int ans = 0,len = 0;
     char     strBuff[256],   //シリアルデバイスに出力する文字列
              strInput[3];    //シリアルデバイスから入力された文字列を格納
              memset(strBuff,0x00,256);
              memset(strInput,0x00,3);

     printf("¥n------- DeviceOpenStart -------¥n");
     if( tty_init( ) == 0 ){      //!< デバイスオープン
          printf ("------- Hit Number -------¥n");

          while(1){
               if( read( ttyd, strInput, 1) >= 1 ){
                    break;
               }
          }
```

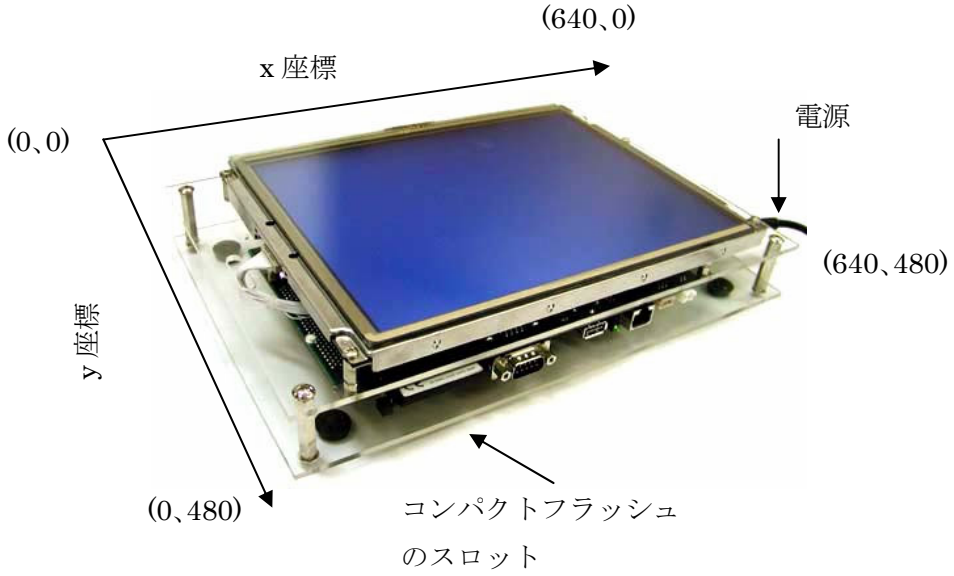
```
        if( ( '0' <= strInput[0] ) && ( strInput[0] <= '9' ) ){
            sprintf( strBuff,"You hit¥t");
            len = strlen( strBuff );
            if( write( ttyd, strBuff, len ) != len ){       //!< デバイスに Write
                printf( "------- Write Error -------¥n" );
                ans = -1;
            }

            len = strlen( strInput );
            if( write( ttyd, strInput, len ) != len ){      //!< デバイスに Write
                printf( "------- Write Error -------¥n" );
                ans = -1;
            }
        }else{
            sprintf( strBuff,"Not Number");
            len = strlen( strBuff );
            if( write( ttyd, strBuff, len ) != len ){       //!< デバイスに Write
                printf( "------- Write Error -------¥n" );
                ans = -1;
            }
        }
    }else{
        //!< デバイスのオープンに失敗
        printf("¥n------- DeviceOpenError -------¥n");
        ans = -1;
    }
    printf("¥n------- DeviceCloseEnd -------¥n");
    tty_close( );                                           //!< デバイスクローズ

    return( ans );
}
```

6 .tmm1000 のディスプレイ表示関数

6-1.LCD 画面の座標

この章からは tmm1000 のディスプレイ表示関数について説明します。この章以降に出てくるライブラリ関数はtmm1000のみで使えるのもであり、一般的なC言語では使えません。

はじめに、ＬＣＤ画面の座標について説明します。座標の原点は下図の位置です。位置の目安として、電源コードを接続するコネクタ、コンパクトフラッシュのスロットを記述しました。



6-2.表示色

TMM1000 は 16bit の 16 進数で色を表現しています。これは 16 桁の 2 進数に直せます。左の 5 桁が R（赤)、真ん中の 6 桁が G（緑)、右の 5 桁が B（青）の強さを表します。

1 1 1 1 1	1 1 1 1 1 1	1 1 1 1 1
R	G	B

例えば赤は 2 進数で“11111 000000 00000”となります。これを 16 進数に直します。はじめに、次のように 4 桁で区切り 10 進数に変換します。

“1111 1000 0000 0000”　左の 4 桁”1111”は 10 進数で表すと”16”です。よって、16 進数では”f”です。次の 4 桁”1000”は 10 進数で表すと”8”で、16 進数でも”8”です。

よって、赤は 16bit の 16 進数で表すと”f800”です。なお、16 進数は”0xf800”のように 0x を付けて表現します。

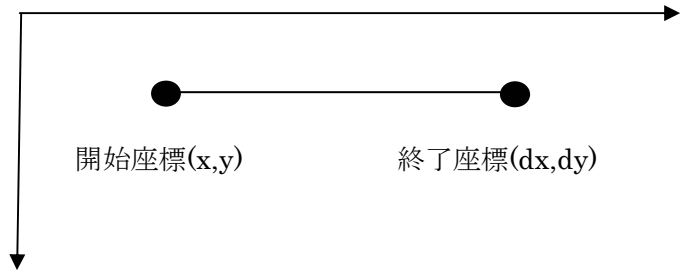
6-3.ディスプレイ表示関数

dsp_init(void)

ＬＣＤ画面の初期化を行います。ＬＣＤ画面を使用するときに一番はじめに使用してください。ここで、フォントがロードされます。

dsp_line(int x , int y , int dx ,int dy , int col)

画面に線を描画します。指定するパラメータは開始の座標（x,y）と終了の座標（dx,dy）と色です。色は 16bit の 16 進数で指定します。ただし、斜め線は現在サポートしていません。

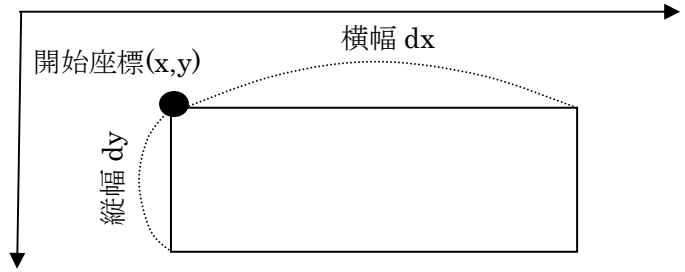


x	開始 x 座標
y	開始 y 座標
dx	終了 x 座標

dy	終了 y 座標
col	色 16bit の 16 進数で指定

dsp_box(int x , int y , int dx ,int dy , int col)

画面に四角形を描画します。下図のように開始座標(x,y)と横幅 dx、縦幅 dy と色を指定します。



x	開始 x 座標
y	開始 y 座標
dx	x 方向（縦方向）の幅

dy	y 方向（縦方向）の幅
col	色 16bit の 16 進数で指定

dsp_boxf(int x , int y , int dx ,int dy , int col , int mode)

画面に四角形を塗りつぶして描画します。パラメータの指定は dsp_box と同様です。

x	開始 x 座標
y	開始 y 座標
dx	x 方向(横方向)の幅

dy	y 方向(縦方向)の幅
col	色 16bit の 16 進数で指定
mode	固定で指定(SMODE_PSET)

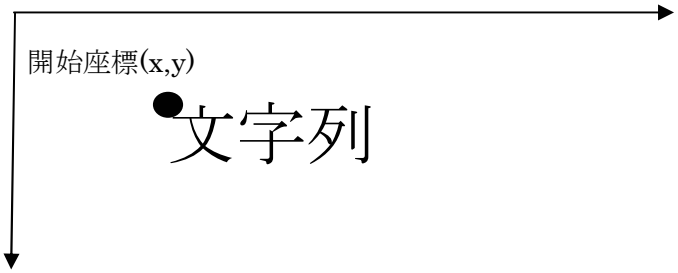
ConvertEUC2SJIS(char *burr);

文字コードを EUC から S-JIS に変換します。char 型の配列 strBurr の文字列の文字コードを変換する場合は以下の通りです。

ConvertEUC2SJIS(strBuff);

dsp_text(int x , int y , int font ,int yoko , int tate , int col , char *dat)

画面に文字を描画します。数のように文字を書き始める座標(x,y)を指定します。font については拡張用に用意したパラメータなので現在は"0"で固定しています。yoko、tate はそれぞれ 0 が標準、1 が倍角です。文字の色は 16bit の 16 進数で指定します。表示する文字ですが、あらかじめ文字コードを S-JIS に変換しておいてください。

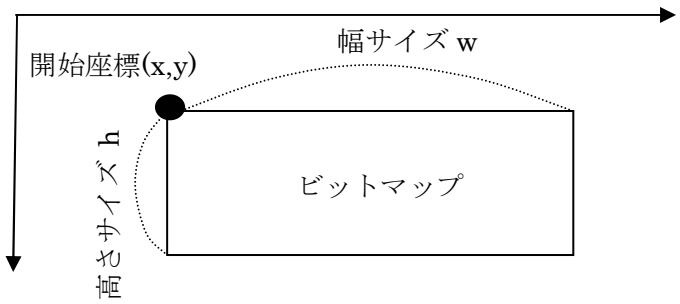


x	開始 x 座標
y	開始 y 座標
font	'0'で固定(拡張用)
yoko	文字の横倍角指定(0:標準 1:倍角)

tate	文字の縦倍角指定(0:標準 1:倍角)
col	色 16bit の 16 進数で指定
dat	sprintf(dat,"databuff") 表示する文字

disp_image(int x , int y , int w ,int h , char *filename)

ビットマップ（画像ファイルで拡張子は bmp です）を描画します。開始座標(x,y)とビットマップの幅と高さをピクセルで指定します。



x	開始 x 座標
y	開始 y 座標
w	ビットマップの幅サイズ（ピクセル）

h	ビットマップの高さサイズ（ピクセル）
filename	sprintf(filename,"sample.bmp") 表示するビットマップを指定

dsp_bell(frec,time)

スピーカーから音を鳴らします。

frec	周波数

time	時間

6-4.図形描画プログラムの例

ディレクトリ”**disp**”に保存されているサンプルプログラムを変更して、タッチパネルを押すと、押した位置に直線、四角などの図形を描画するプログラムを作成します。

はじめに、サンプルプログラムの 38 行目 65 行目までを”**#if 0**”と”**#endif**”でくくり、コンパイルしないようにしてください。（これでディスプレイには何も描かれなくなります）

```
#if 0
//!< ******* 直線線画
dsp_line(20,20,480,20,COL_RED );
/* Not support Slash*/
            ～～～～～～～～～～～～～～～～
            ～～～～～～～～～～～～～～～～
dsp_image(166,240,308,231,"sample01.bmp" );//!< イメージ線画
//!< 65535 色用
#endif
```

タッチパネルの座標の取得ですが、サンプルプログラムに”**int　get_touch(int *x,int *y)**”という関数があるので、これを使います。タッチパネルが押されていると“**0**”を返します。変数 x と y にタッチパネルの座標を取得します。

76 行目～83 行目を参照してください。

```
ret = get_touch( &x,&y );
if( ret == 0 ){
    dsp_line( x,y, x+20 ,y , COL_GREEN );
    usleep(10000);
}else{
```

上記のように if 文でタッチパネルが押されたかどうかを判断します。

dsp_line(x,y, x+20 ,y , COL_GREEN);のように記述すれば、タッチパネルに触れた場所から横方向に 20 ドットの長さの線が引けます。

この他にも 6-3 節の描画関数を使って様々な図形を試してみてください。

お問合せ先

東和メックス株式会社　市場開発本部　法人営業部　担当：増田、笹岡
TEL：03-3816-7864
E-mail：info@towa-meccs.co.jp

下記ホームページにて、詳細情報を随時更新しております。

■東和ＳＨボード　ＴＭＭ１０００
http://www.towanet.com/seihin/sh_board/index.html

■東和メックス株式会社
http://www.towa-meccs.co.jp