●お使いになる前に必ずお読み下さい

ソフトウェアの使用条件と保障

■著作権

　本ソフトウェアは著作権法により保護されています。有限会社イエローソフトが本ソフトウェアの著作権を保有します。
　したがって、本ソフトウェアを弊社に無断で、売却、譲渡、賃貸またはその他いかなる方法であっても第三者に使用させることはできません。

■使用の制限

　お客様は、本ソフトウェアを１台のコンピュータに限ってインストールすることができます。また、バックアップを目的とする複製のみ許可します。

■保障

　弊社は、本ソフトウェアのディスクおよびマニュアルに物理的欠陥がある場合、ご購入日から３０日以内であれば無料で交換いたします。

■免責

　弊社は、前項に定める場合を除き、本ソフトウェアの使用、あるいは使用結果に対していかなる責任も負いません。
　弊社は、本ソフトウェアによって生成される２次的ソフトウェアに関して、いかなる制限も与えません。またいかなる保障もいたしません。

もくじ

第１章　概要-- 3
１．１　ファイルシステムとは-- 3
１．２　YS-FILE の特徴-- 3
１．３　未サポート情報--4
１．４　FAT フォーマットの制限--4
１．５　必要なシステム--4
１．６　ファイルシステムの大きさ--5
１．７　ファイルシステムアプリケーションの構成------------------------------6
１．８　システムコール一覧--8
１．９　ANSI 関数一覧-- 9
１．１０　重要な概念--10
１．１１　プログラム作成手順--17
第２章　デバイスドライバの作成---------------------------------18
２．１　デバイスドライバの作成--18
２．２　パーテーションテーブルへの対応------------------------------------22
第３章　コンフィグレーション-----------------------------------25
３．１　yfconfig.h の変更-- 25
３．２　ystdio.h の変更-- 31
３．３　コンフィグレーション後の再構築------------------------------------32
３．４　セーフシステム使用時のリカバリ処理について-------------------------32
第４章　システムコール詳細-----------------------------------33
第５章　ANSI 関数詳細-- 69
第６章　miniDOS--102
６．１　miniDOS とは-- 102
６．２　使用できるコンパイラ--102
６．３　miniDOS のコンパイル--102
６．４　miniDOS コマンド一覧-- 103
６．５　miniDOS コマンド詳細-- 104

第１章　概要

１．１　ファイルシステムとは

ファイルシステムとはディスク上のデータを"ファイル"として扱うためのソフトウェアです。Windows 上で作られたファイルはFAT ファイルシステムというファイルシステムによって保存されており、ファイルシステムを使うことによって、ユーザアプリケーションからこのファイルを読んだり、書いたりできます。逆にユーザアプリケーションで作成したファイルをWindows パソコンで読むことができます。

パソコン上のプログラムをC 言語で組む場合、fopen や fputs などの関数を普通に使っていますが、組込みマイコンのプログラムではそれが使用できませんでした。しかしファイルシステムを使うことによって、これらの関数が使用できるようになります。

１．２　YS-FILE の特徴

イエローソフトのファイルシステムであるYS-FILE には以下の特徴があります。

●Windows 互換 FAT12/FAT16/FAT32 をサポート
●階層ディレクトリ、複数ドライブ、相対パス対応
●ANSI 準拠のファイル入出力関数をサポート
●バッファサイズなどコンフィギュレーション可能
●電源断によるファイルシステムの修復機能
●サンプルとして簡単なディスクOS [miniDOS]が付属
●ソースコード付属
●ライセンスフリー

電源断によるファイルシステムの修復機能は、ファイルシステム(FAT)を修復するのであって、アクセス途中のデータを修復するのもではありません。FAT 領域の破壊によってすべてのファイルがアクセスできなくなってしまうことを避けます。

１．３　未サポート情報

以下の機能はサポートしていませんので使用には注意して下さい。

●NTFS
●ロングファイル名(VFAT)
ただし、ロングファイル名を含むディスクであっても問題なくアクセスできます。この場合、ロングファイル名はショートファイル名として認識されます。
●フォーマット機能
フォーマット機能はありません。フォーマットはパソコンで行って下さい。
●リアルタイム OS への対応
リアルタイム OS へは特に対応していません。したがってリアルタイムシステムで使う場合は、ファイルシステムをコールするタスクを一つに統一する、あるいは排他制御を行うなどする必要があります。

１．４　FAT フォーマットの制限

FAT12/FAT16/FAT32 を使用できますが､使用できるフォーマット形式に一部制限があります。制限は以下の通りですが、通常のディスクを Windows でフォーマットした場合は問題ありません。

1　セクタサイズは 512 バイト
2　FAT 領域が 2 つあること

１．５　必要なシステム

●イエローソフトの C コンパイラ YCH8 または YCSH
●YellowIDE Ver.6 以上
ターゲットの条件
●H8/300H、H8S、SH1、SH2 シリーズの CPU
●16K バイト以上の RAM

１．６　ファイルシステムの大きさ

下記はあくまでも目安です。ファイルシステムのバージョン、コンパイルオプションなどによって変わる場合があります。

最小構成　ANSI 関数を使いファイルコピープログラムを作った場合。FTA16 だけ有効、セーフシステム OFF の設定、バッファ最小。
（使用した関数、yfopen,yfclose,yfread,yfwrite)

	H8	SH
ROM	約 44K バイト	約 36K バイト
RAM	約 9K バイト	約 9K バイト

上記プログラムをシステムコールだけで作った場合
（使用した関数、yfOpen,yfCreate,yfClose,yfRead,yfWrite)

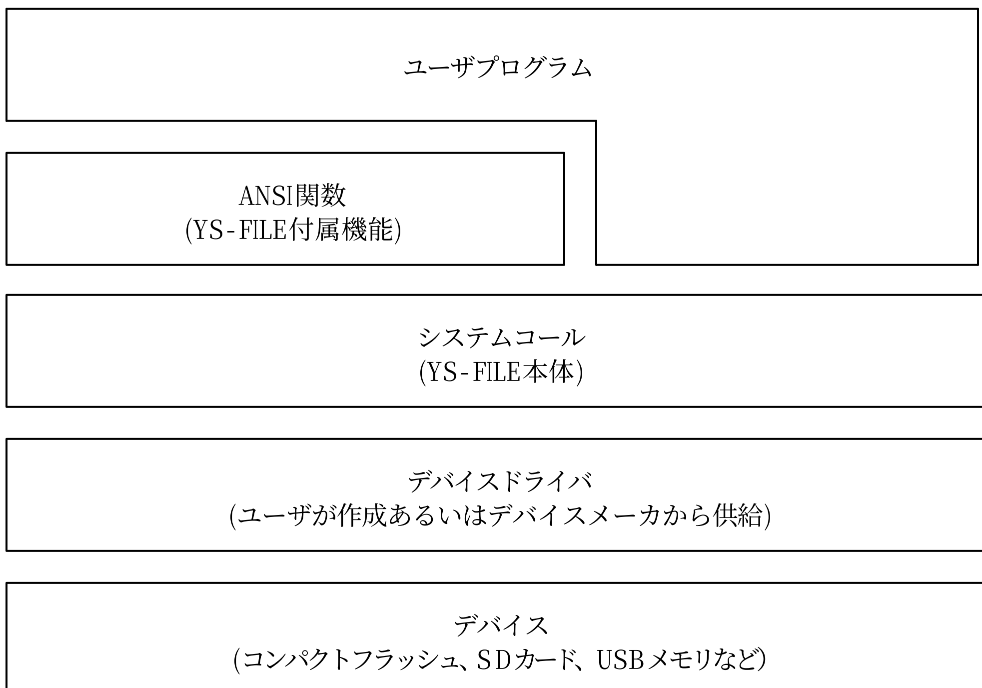
	H8	SH
ROM	約 37K バイト	約 30K バイト
RAM	約 8K バイト	約 8K バイト

サンプルプルグラムの miniDOS のプログラムサイズ(多くの ANSI 関数を使用した場合、すべての FAT を有効、セーフシステム ON の設定、バッファ最小）

	H8	SH
ROM	約 90K バイト	約 74K バイト
RAM	約 11K バイト	約 12K バイト

１．７　ファイルシステムアプリケーションの構成

ファイルシステムを使ったアプリケーションプログラムは以下のような階層構造をとります。下位の層から説明します。



●デバイス

コンパクトフラッシュやSDカードなど、物理的な実体です。マイコンとのインターフェース部分も含めてデバイスと呼びます。

●デバイスドライバ

デバイスとYS-FILEを結ぶ層です。デバイスドライバは実際にはセクタ単位のリード・ライトを行う関数のことです。（後で詳しく説明します）YS-FILEからはこのデバイスドライバを通してデバイスにアクセスします。したがって、デバイスドライバを作成できればどのようなデバイスであってもYS-FILEが使用できます。

デバイスドライバはデバイスに合わせてユーザが作成します。しかし、イエローソフトが販売するデバイスを使用する場合はイエローソフトがデバイスドライバを供給します。

●システムコール

YS-FILEの本体部分です。システムコールはファイルにアクセスするための最小限の機能を有したC言語の関数群です。この関数を呼び出すことによってファイルにアクセスできます。

●ANSI関数

C言語のstdio.hで定義される関数のうちファイルの入出力に関連した関数です。インターフェースが統一されているため移植性に優れています。

●ユーザプログラム

ユーザプログラムはANSI関数かシステムコールを直接呼び出すことによってファイルにアクセスします。ANSI関数を使うか、システムコールを使うかは状況によります。以下の判断基準に従って下さい。

ANSI関数にはない機能はシステムコールを使う
　例　ディレクトリの作成・削除、カレントディレクトリの変更など
高速性を追求するならシステムコールを使う
移植性、便宜性を追求するならANSI関数を使う

通常はANSI関数を使用します。高速性が必要な場合や大きなファイルを扱う場合などシステムコールを使います。

１．８　システムコール一覧

YS-FILE で使用できるシステムコールの一覧です。

システムコール名	機能
yfInitFileSystem	ファイルシステムの初期化
yfFindFirst	最初に一致するファイルの検索
yfFindNext	次に一致するファイルの検索
yfMakeDir	ディレクトリの作成
yfMakeFile	ファイルの作成
yfDelete	ファイルまたはディレクトリの削除
yfRename	ファイル名またはディレクトリ名の変更
yfGetAtr	ファイルまたはディレクトリの属性取得
yfSetAtr	ファイルまたはディレクトリの属性変更
yfGetTime	ファイルまたはディレクトリの時刻日付取得
yfSetTime	ファイルまたはディレクトリの時刻日付変更
yfGetCurDir	カレントディレクトリの取得
yfChangeCurDir	カレントディレクトリの変更
yfGetCurDrv	カレントドライブの取得
yfChangeCurDrv	カレントドライブの変更
yfGetOpenFileCount	現在オープンされているファイル数の取得
yfCreate	ファイルの作成とオープン
yfOpen	ファイルのオープン
yfClose	ファイルのクローズ
yfCloseAll	オープンされているファイルすべてのクローズ
yfWrite	ファイルのライト
yfRead	ファイルのリード
yfSeek	ファイルのシーク
yfGetBpbAdrs	BPB アドレスの取得
yfGetEmptyClust	空きクラスタ数の取得
yfGetLabel	ボリュームラベルの取得
yfSetLabel	ボリュームラベルの設定・変更
yfDelLabel	ボリュームラベルの削除
yfCheckFat	2 つの FAT 領域の比較
yfRecoveryFileSystem	ファイルシステムの修復
yfErrorMsg	エラーメッセージ文字列の取得

１．９　ANSI関数一覧

stdio.hで定義される以下の関数が使用できます。

関数名	機能
yfopen	ファイルのオープン
yfreopen	ファイルの再オープン
yfclose	ファイルのクローズ
ysetvbuf	YFILEへのバッファ割り当てと制御
ysetbuf	YFILEへのバッファ割り当て
yfflush	バッファのフラッシュ
yfgetc/ygetc	ファイルから一文字読み込み
yfgets	ファイルから一行読み込み
yfputc/yputc	ファイルへの一文字書き込み
yfputs	ファイルへの文字列書き込み
yungetc	ファイル入力バッファへの文字押し戻し
yfread	ファイルからのデータ配列読み込み
yfwrite	ファイルへのデータ配列書き込み
yfseek	ファイルのシーク
yrewind	ファイルの巻き戻し
yftell	ファイルの現在位置の取得
yfgetpos	ファイルの現在位置の取得
yfsetpos	ファイルの現在位置の移動
yremove	ファイルの削除
yrename	ファイル名の変更
yfeof	ファイル終端の検出
yferror	エラーの検出
yclearerr	エラー状態、ファイル終端状態のクリア
ytmpnam	一時ファイル名の生成
ytmpfile	一時ファイルの作成
yfscanf	ファイルからの書式付き入力
yfprintf	ファイルへの書式付き出力
yvfprintf	ファイルへの書式付き出力(可変個引数集付き)

１．１０　重要な概念

●ファイル名

ファイル名は半角８文字までです。

ファイル名の後にピリオドを続けて拡張子を記述できます。拡張子は半角３文字までです。

名前と拡張子には英数、カナ、全角文字が使用できます。ただし、以下の文字は使用できません。

縦線(|)
ドット(.)
コロン(:)
ダブルコーテーション(")
不等号(<>)
セミコロン(;)
円記号　(\)
ピリオド(.)　ただし、拡張子とファイル名の区切りとしては可
空白、タブ
ワイルドカード文字(?、*)

●ワイルドカード

システムコールでのファイル名指定にワイルドカードを使用できる場合があります。ワイルドカードの使用によって複数のファイルを一回で指定できます。
（システムコールでワイルドカードが利用できるのは yfFindFirst です）

ワイルドカードには?と*の２種類あります。

?　疑問符は任意の一文字に置き換えられます。例えば以下の例は

TEST?.DAT

次のようなファイルと一致します。

〇TEST1.DAT
〇TEST2.DAT
〇TEST3.DAT
〇TESTA.DAT
×TEST12.DAT

*　アスタリスクは0文字以上の任意の文字列に置き換えられます。例えば以下の例は

TEST*.DAT

次のようなファイルと一致します。

〇TEST1.DAT

〇TEST123.DAT

〇TEST.DAT

●ドライブ名

　アルファベット一文字（大文字、小文字問わず）とコロン(:)でドライブを識別します。ドライブは接続された順番にA: B: C:　.... となります。

●ドライブ番号

　システムコール関数の中にはドライブをドライブ名ではなく番号で識別するものがあります。それをドライブ番号と言います。A: B: C: ....の順に0、1、2...と番号を割り付けます。

●ディレクトリ名

　Windowsの時代になって現代では「フォルダ名」と表現した方がわかりやすいかも知れません。ディスクの中にディレクトリを作成して、その中にファイルをまとめることができます。またディレクトリの中にさらにディレクトリを作ることができます。

　ディレクトリの名前の規則はファイル名と同じです。ただし、拡張子の概念はありません。

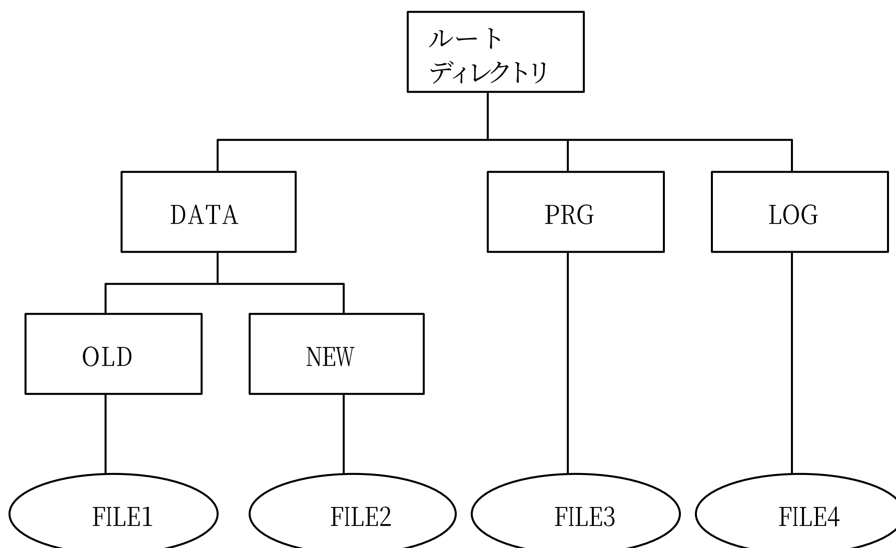
●ルートディレクトリ

　ディレクトリ構造の最上位に位置するのがルートディレクトリです。ルートディレクトリに名前はなく、単に[\]で表されます。

●パス名

同じ名前のファイル名であっても格納されているディレクトリが違えば、違うファイルとして扱われます。

したがって、ファイルを指定する場合、ファイル名の先頭にドライブ名、ディレクトリ名を円記号(\)で区切って続けます。



上記はあるドライブ(A:ドライブとします）のディレクトリ構造を表した一例です。四角がディレクトリを表し、楕円がファイル名を表します。
ファイルFILE1〜FILE4を正確に表現すると以下のようになります。

A:\DATA\OLD\FILE1

A:\DATA\NEW\FILE2

A:\PRG\FILE3

A:\LOG\FILE4

このように　ドライブ名　ディレクトリ名....　ファイル名を円記号(\)で区切ってつなげます。

●カレントドライブ

ドライブ名、ディレクトリ名、ファイル名を続けて表すと、文字列が長くなり面倒です。そこでカレントドライブというのを設定します。

カレントドライブとは、「現在のドライブ」、「起点となるドライブ」という意味で、カレントドライブにより、ドライブ名を省略することができます。ドライブ名が書略された場合はカレントドライブが選択されているのもとして解釈されます。

そうすると前ページのファイル名は以下のように表されます。

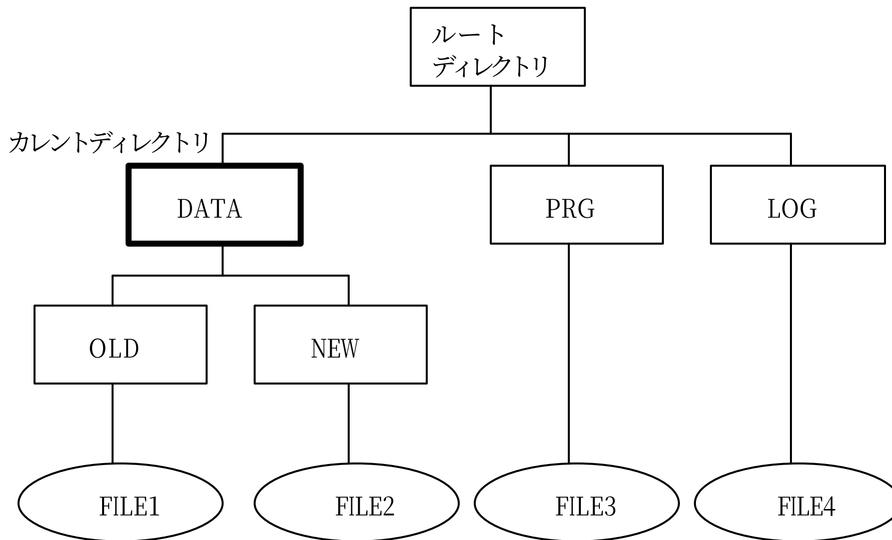
\DATA\OLD\FILE1
\DATA\NEW\FILE2
\PRG\FILE3
\LOG\FILE4

カレントドライブではないドライブのファイルを表すときはドライブ名を指定するようにします。カレントドライブは yfChangeCurDrv システムコールで変更できます。初期状態のカレントドライブはA:（ドライブ番号0)です。

●カレントディレクトリ

カレントドライブと同じ概念で、「現在のディレクトリ」、「起点となるディレクトリ」をカレントディレクトリといいます。カレントディレクトリが設定されていると、そのディレクトリから下の部分だけ表示すればファイルを指定できます。

例えば前図で、DATA がカレントディレクトリだとします。



そうするとFILE1とFILE2は次のように表すことができます。

OLD\FILE1
NEW\FILE2

FILE3 と FILE4 はカレントディレクトリの下にないため依然としてルートディレクトリから表します。

もし、OLDがカレントディレクトリであるならばFILE1は単にFILE1とだけ記述すればよくなります。

FILE1

カレントディレクトリは yfSetCurDir システムコールで変更できます。初期状態のカレントディレクトリはルートディレクトリです。

●親ディレクトリ

カレントディレクトリを基準にして一つ上のディレクトリを親ディレクトリといい、ピリオド２つで表します。[..]

この親ディレクトリの記号[..]を使えば、すべてのファイルをカレントディレクトリを基準にして表すことができます。

カレントディレクトリが\DATA の場合

OLD\FILE1

NEW\FILE2

..\PRG\FILE3

..\LOG\FILE4

●自ディレクトリ

親ディレクトリはピリオド２つで表されますが、ピリオドが一つの場合[.]は自分自身のディレクトリを表します。しかし、自分自身のディレクトリをわざわざ表す必要性はなく、この記号がファイルの指定で使われることはあまりありません。

●相対パスと絶対パス

ファイルを指定する際、ルートディレクトリからたどってファイルを指定する方法を絶対パス指定といいます。カレントディレクトリがどこであろうとファイルを一意に指定することができます。

これに対して、カレントディレクトリを基準にファイルを指定する方法を相対パス指定と言います。相対パス指定ではファイルを簡素に指定できますが、カレントディレクトリがどこにあるか常に意識する必要があります。

１．１１　プログラム作成手順

以下の順番に従ってファイルシステムアプリケーションを作成します。

１　YS-FILE のインストール

２　デバイスドライバの作成（イエローソフト製のデバイスを使用する場合不要）

３　YS-FILE のコンフィグレーション

４　ユーザプログラムの作成

基本的に「SD カード開発セット　導入の手引き」または「USB メモリ開発セット　導入の手引き」にしたがって開発を進めて下さい。弊社以外のデバイス使用する場合は、デバイスドライバの作成が必要です。次の章の説明を読んで下さい。

第２章　デバイスドライバの作成

YS-FILE は､SD カード開発セット/USB メモリ開発セットの一部として販売されています。（単体では販売していません。）　開発セットの SD カード IO モジュールまたは USB メモリ IO モジュールを使用する場合はデバイスドライバの作成は不要です。しかし多少の変更が必要になります。詳しくは「SD カード/USB メモリ開発セット　導入の手引き」を読んで下さい。

ここでは､YS-FILE をユーザ独自のデバイスや他のデバイス（例えばコンパクトフラッシュ、RAM ディスクなど）で使用するための説明をします。

YS-FILE はデバイスドライバを書き換えることによってどんなデバイスにも応用することができます。
また､「SD カード/USB メモリ開発セット　導入の手引き」も合わせて読んで下さい。

２．１　デバイスドライバの作成

「SD カード/USB メモリ開発セット　導入の手引き」の「第３章　デバイスドライバの移植と動作テスト」においてファイル DEVICE.C の中の関数を変更することによってどんなデバイスにも対応することができます。

- void yfBusInit(void)
- void yfTimerInit(void)

「導入の手引き」のマニュアルの通り変更して下さい。

- int yfInitDevice(void)

引数　　なし
戻り値　成功＝０
　　　　失敗＝１

デバイスを初期化する命令を記述します。この関数はプログラム実行後、一度だけ呼ばれます。必要なければ空の関数にして下さい。

● int yfInit(int DriveNo)

引数　　ドライブ番号

戻り値　成功＝0

　　　　失敗＝1

　デバイスを初期化する命令を記述します。yfInitDeviceとの違いは、この関数は引数にドライブ番号を持つことと、この関数はデバイスが交換される度に呼ばれることです。この関数が必要なければ空の関数にして下さい。

● int yfDiskEject(int DriveNo)

引数　　ドライブ番号

戻り値　成功＝0

　　　　失敗＝1

　デバイスが抜き取られた時の処理を記述して下さい。必要なければ空の関数にして下さい。

● int yfReadSector(int DriveNo, DWORD sect, BYTE *buf)

引数　　DriveNo＝ドライブ番号

　　　　sect＝セクタ番号

　　　　buf＝読み込んだセクタデータの格納先

戻り値　成功＝0

　　　　失敗＝1

　デバイスからセクタを読み込む命令を記述して下さい。なお、アドレス buf は YS-FILE が領域を確保しますが4バイト境界が保証されています。したがって、WORDやDWORDサイズで転送を行っても問題ありません。その方が高速です。ただしキャストは必要です。

● int yfWriteSector(int DriveNo, DWORD sect, BYTE *buf)

引数　　DriveNo＝ドライブ番号

　　　　sect＝セクタ番号

　　　　buf＝書き込むデータの格納先

戻り値　成功＝0

　　　　失敗＝1

　デバイスのセクタに書き込む命令を記述して下さい。なお、アドレス buf は YS-FILE が領域を確保しますが4バイト境界が保証されています。したがって、WORDやDWORDサイズで転送を行っても問題ありません。その方が高速です。ただしキャストは必要です。

● int yfWriteSectorM(int DriveNo, MultiSectBuf *Msb, int count)

引数　　DriveNo＝ドライブ番号

　　　　Msb＝連続セクタ情報

　　　　count＝連続するセクタの数

戻り値　成功＝0

　　　　失敗＝1

　デバイスの連続するセクタにデータを一度に書き込む命令を記述して下さい。Msb は連続セクタ情報を格納する構造体配列のアドレスで、この構造体配列は次のような構造をもちます。

```
typedef struct _MultiSectBuf {
        DWORD    sect;
        DWORD    *buf;
} MultiSectBuf;
MultiSectBuf Msb[];
```

　例えば、連続するセクタ 101,102,103 に buf1、buf2、buf3 のデータを書き込む場合は、YS-FILE は構造体配列 Msb に次のようなデータをセットして、この関数を呼びます。

```
        MultiSectBuf Msb[3] = {
                        {101, buf1},
                        {102, buf2},
                        {103, buf3},
                        }
```

　なお、デバイスに連続セクタを一度に書き込む機能がない場合、あるいはあっても 1 セクタ書き込みを繰り返した場合と速度的に大差ない場合は､1 セクタの書き込み関数を使って作成して下さい。

例

```
int yfWriteSectorM(int DriveNo, MultiSectBuf *Msb, int count)
{
	int i;
	unsigned long sect = Msb[0].sect;
	for (i = 0; i < count; i++) {
		if (yfWriteSector(DriveNo, sect++, Msb[i].buf)) {
			return 1;
		}
	return 0;
}
```

あるいは後述するYS-FILEのコンフィグレーションで、以下のシンボルをコメントアウトします。　そうするとYS-FILEはこの関数を使用しません。

```
/*********************************************************************
*****    マルチセクタライト                                        *****
*********************************************************************/
#define MULTI_SECTOR_WRITE
```

なお構造体MultiSectBufはsddrv.h(SD版)またはusbdrv.h(USB版)でインクルードされたysfile.hで定義されています。したがってsddrv.h/usbdrv.hをインクルードしなければ、代わりにysfile.hをインクルードして下さい。

● int yfCheckDiskInsert(int DriveNo)

引数　　DriveNo＝ドライブ番号

戻り値　ディスクが挿入されている場合＝1

　　　　ディスクが挿入されていない場合＝0

デバイスにディスクが挿入されているか、いないかを調べる関数です。RAMディスクのように常に挿入されている場合などは常に1を返して下さい。

この関数は関数void yfTimerInit(void)で設定されたインターバルタイマ割り込みによって起動されます。

２．２　パーテーションテーブルへの対応

パーテーションテーブルはディスクの最初のセクタにあり、ディスクを論理的に複数のディスクとして扱うための情報が格納されています。ディスクをパーテーションで分けて使わない場合もパーテーションテーブルが最初のセクタにある場合があります。

デバイスドライバの作成ではyfReadSectorとyfWriteSectorでセクタ番号を扱いますが、このセクタ番号はディスクの物理的なセクタ番号ではなく、各論理ドライブの論理的なセクタ番号である必要があります。

そのためパーテーションテーブルの構造を知って、論理セクタ番号を物理セクタ番号に変換する操作が必要です。

パーテーションテーブルの構造

パーテーションテーブルはディスクの最初のセクタのオフセット1BEHから64バイトにわたりあります。パーテーションテーブルは各16バイトのエントリ4つからなり、各エントリには論理ディスクの開始セクタ番号が格納されています。

パーテーションテーブルの場所

各エントリの構造

0	1　2　3	4	5　6　7	8　9　A　B	C　D　E　F
	CHS開始セクタ		CHS終了セクタ	LBA開始セクタ リトルエンディアン	LBA終了セクタ リトルエンディアン
アクティブフラグ		パーテションタイプ			

プログラミング方法

ディスクが交換される度に呼ばれる初期化関数yfInitにおいてパーテーションテーブルを読み込みます。

```
//パーテーションテーブルの読み込み
if (sdSectorRead(buf, 0) != SD_OK) {
        return 1;
}
//パーテーションテーブルでない
if (buf[511] != 0xAA || buf[510] != 0x55) {
        return 1;
}

//パーテーションエントリの読み込み
for (i = 0; i < 4; i++) {
        j = 16*i;
        sec = (((unsigned long)buf[j + (0x01be + 8 + 0)]) << 0)
                + (((unsigned long)buf[j + (0x01be + 8 + 1)]) << 8 )
                + (((unsigned long)buf[j + (0x01be + 8 + 2)]) << 16 )
                + (((unsigned long)buf[j + (0x01be + 8 + 3)]) << 24 );
        ParStartSect[i] = sec;
}
```

最初のsdSectorRead(buf, 0)はイエローソフトのSDカードIOモジュールの物理セクタを読み込む関数です。他のデバイスの場合は違う関数になりますが、ここで最初のセクタ512バイトを読み込みます。

次にセクタの最後2バイトに0x55、0xAAが書かれていることを確認します。

最後に各エントリのLBA開始セクタを読み込んで配列ParStartSect[]に格納します。ここで注意することはLBA開始セクタはリトルエンディアンで書き込まれていますのでビックエンディアンに変換することです。

これで各論理ディスクの開始セクタが配列ParStartSectに格納されます。

セクタのリード、ライト関数ではこの配列 ParStartSect を参照して、論理セクタ番号を物理セクタ番号に変換します。その例を以下に示します。

```
/*************************************************************
*****   セクタのリード                                  *****
*****   引き数          sect    セクタ番号              *****
*****                   buf     データの格納ポインタ    *****
*****   戻り値          成功    0                       *****
*****                   失敗    0 以外の値              *****
*************************************************************/
int yfReadSector(int DriveNo, DWORD sect, BYTE *buf)
{
        sect+=ParStartSect[DriveNo];
        if (sdRead(buf, 1, sect) == SD_OK) {
                return 0;
        }
        return 1;
}
```

sect += ParStartSect[DriveNo]の部分が論理セクタ番号を物理セクタ番号に変換する部分です。

以上、すべての関数を目的のデバイスに合わせて変更すれば、YS-FILE が使用できるようになります。

この後の、メイクの仕方、およびテストプログラムの実行に関しては「導入の手引き」を読んで下さい。

第３章　コンフィグレーション

YS-FILE のコンフィグレーションに関しては「SD カード/USB メモリ開発セット　導入の手引き」で少し説明しましたが、ここでさらに詳しく説明をしておきます。

YS-FILE はソースコードが供給され再構築することによってユーザシステムに合わせて最適化を行うことができます。コンフィグレーションはヘッダファイルの設定値を変えることによって簡単に行うことができます。

「SD カード/USB メモリ開発セット　導入の手引き」の「第４章　YS-FILE の構築とテストプログラムの実行」で以下のファイルを書き換えることによってコンフィグレーションを行います。

yfconfig.h　　YS-FILE 本体のコンフィグレーション
ystdio.h　　ANSI 関数のコンフィグレーション

ystdio.h は C コンパイラのインクルードディレクトリにあることに注意して下さい。(YellowIDE6 の場合は\YellowIDE6\INCLUDE\H8 または\YellowIDE6\INCLUDE\SH)

３．１　yfconfig.h の変更

①FAT の選択

```
/****************************************************************
*****     FAT の選択                                      ******
****************************************************************/
//#define         YSFILE_FAT12
#define  YSFILE_FAT16
#define  YSFILE_FAT32
```

使用しない FAT システムは#define 文をコメントにします。これによりプログラムを小さくすることができます。特に FAT12 は現在では使われることが少ないのではずしておくのが良いでしょう。

②セーフシステムの選択

```
/****************************************************************
*****    セーフシステム                                    ******
*****    電源断からの回復システムを有効にする              ******
****************************************************************/
#define         SAFE_SYSTEM
```

セーフシステムは書き込み途中の電源断による、あるいは割り込みなどによる暴走からファイルシステムを回復するシステムです。しかし、セーフシステムが動いていると、ディスクへの書き込み回数が増え、速度が遅くなります。以下の場合にはセーフシステムは必要なく、この#define 文をコメントにします。

1　ディスクの読み込みしか行わない
2　装置全体の安全性が高く書き込み途中の電源断は起こらない
3　たとえ電源断によってファイルシステムが壊れても、非常に希なことなので特に問題としない。

注意

セーフシステムはファイルシステム自身を守るものであって、書き込み途中のデータを守るものではありません。書き込み途中で電源断が起こった場合、書き込みデータは失われます。しかし、ファイルシステムは正常に回復されディスク全体がアクセスできなくなってしまうことはありません。またディスクが物理的に破壊された場合については回復できません。

③ドライブの数

```
/****************************************************************
*****    ドライブの数                                      ******
*****    無用に多くしないでください                        ******
****************************************************************/
#define MAX_DRIVE_NO    1
```

システム全体のドライブ数を設定します。ドライブ数が倍になれば使用するメモリも倍になります。したがって無用に多くしないで最小値を設定して下さい。

④一度にオープンできるファイルの数

```
/***************************************************************
*****    一度にオープンできるファイルの数                    *****
*****    無用に多くしないでください                          *****
***************************************************************/
#define MAX_OPEN_FILE    6
```

同時にオープンできるファイル数を指定します。あくまでも”同時”にオープンできるファイル数です。クローズしたファイルはカウントされません。この数値が1増えると70〜80 バイトのメモリを消費します。したがって無用に多くしないで下さい。

また ANSI 関数を使う場合は ANSI 関数のコンフィグレーションでも同時にオープンできるファイル数を増やします。詳しくは後述します。

⑤ファイル名の最大長

```
/***************************************************************
*****    パス+ファイル名の最大長                             *****
****************************************************************/
#define MAX_FILE_PATH    511
```

ディレクトリ部分も含めたファイル名の最大長を指定します。この数値を削ることによって多少メモリを節約できますが、あまり変わりません。ディレクトリ階層が深くなってファイル名が長くなってしまった場合にここの数値を大きくします。

⑥バッファのサイズ

YS-FILE はセクタの読み込みにバッファを介して行います。バッファは、用途別にデータセクタバッファ、FAT セクタバッファ、ディレクトリエントリセクタバッファの３つがあります。それぞれ別個にセクタバッファの個数(512 バイト単位）を設定できます。

```
//セクタバッファの数　512 バイト単位　2 以上を推奨
#define MAX_SECTOR_BUFFER        1                    //1 以上
```

MAX_SECTOR_BUFFER でデータセクタバッファの個数を指定します。最低1です。2 以上を推奨します。

```
//FAT バッファの数　512 バイト単位　最低2 4 以上を推奨
#define MAX_FAT_BUFFER             2                //2 以上
```

MAX_FAT_BUFFER で FAT セクタバッファの個数を指定します。FAT は一つのディスクに 2 つあるので最低2です。4 以上を推奨します。

```
//ディレクトリエントリバッファの数 512 バイト単位　2 以上を推奨
#define MAX_DIRENT_BUFFER          1                //1 以上
```

MAX_DIRENT_BUFFER でディレクトリエントリセクタバッファの個数を指定します。最低1です。2 以上を推奨します。

限りあるメモリを 3 種類あるセクタバッファにどのように配分すれば良いかの目安は以下の通りです。

1　大きいファイルを効率よくアクセスしたい場合はデータセクタバッファ(MAX_SECTOR_BUFFER)を優先的に増やす。
2　ディスクの中に多数のファイルがあり、どれもよくアクセスする場合は FAT セクタバッファを増やす。
3　サーチを速くしたい場合はディレクトリエントリバッファを増やす。
4　書き込みの方が多い場合はFAT セクタバッファを増やす。

⑦空きクラスタバッファの数

```
//空クラスタバッファの数　5 以上を推奨
#define MAX_EMPTY_CLUST_BUFFER     5                //1 以上
```

ファイルまたはディレクトリを作成する際にYS-FILE は空きクラスタをサーチしますが、この数値を上げることによりサーチを速くすることができます。最低１です。５以上を推奨します。あまり大きくしすぎるとかえって遅くなることがあります。

⑧連続セクタ書き込みの許可

```
/*************************************************************
*****    マルチセクタライト                                  *****
*************************************************************/
#define MULTI_SECTOR_WRITE
```

YS-FILE は連続するセクタは一度に書き込みを行います。デバイスが連続セクタの書き込みに対応していない場合、あるいは対応していても 1 セクタずつの書き込みと速度的に大差ない場合はこの#define 文をコメントにします。

⑨ライトリミッタ

```
/*************************************************************
*****    ライトリミッタ                                      *****
*************************************************************/
//#define        WRITE_LIMIT       (32*1024)
```

システムコールの引数に不正な値を使用して大きなファイルを作成してしまうことを制限します。例えばyfWrite にて書き込みサイズに間違って-1 という値を渡してしまった場合、巨大なファイルを作ってしまいます。

そのような不具合を避けるのがライトリミッタです。初期状態ではライトリミッタは不許可になっています。コメントをはずして WRITE_LIMIT を許可して下さい。

⑩yfprintf、yfscanf の効率化

yfprintf と yfscanf は大きな関数です。プログラムを少しでも小さくするための設定です。

```
/*************************************************************
*****    コンパイラ付属の sprintf を使用                      *****
*************************************************************/
//#define              PRINTF_COMLIB
//#define              PRINTF_BUFSIZE   128
```

YS-FILE は yfprintf 関数を自前で持っていますが、コンパイラ付属の vsprintf 関数を利

用することによって重複を避け、プログラムを小さくすることができます。

PRINTF_COMLIB と PRINTF_BUFSIZE のシンボルを両方同時に定義して下さい。PRINTF_BUFSIZE は vsprintf で一時的に文字列を格納するサイズです。したがって、ここで定義した値以上の文字を出力することはできません。

```
/*************************************************************
*****     yfprintf 関数での浮動小数点の使用                *****
*************************************************************/
#define          PRINTF_FLOAT


/*************************************************************
*****     yfscanf 関数での浮動小数点の使用                 *****
*************************************************************/
#define          SCANF_FLOAT
```

yfprintf 関数等を大きな関数にしている一つの原因は浮動小数点演算の扱いです。浮動小数点を使用しないのであれば上記２つのシンボル定義をコメントにして削除して下さい。そうするとプログラムが小さくなります。

ただし、PRINTF_COMLIB を定義している場合は、コンパイラ付属の vsprintf 関数が使用されるため、影響はありません。

３．２　ystdio.hの変更

ANSI 関数に関するコンフィグレーションは ystdio.h を書き換えることによって行います。

注意

YellowIDE6 を使用している場合 ystdio.h は\YellowIDE6\INCLUDE\H8 と\YellowIDE6\INCLUDE\SHの両方にインストールされます。H8、SHの両方で使用するには両方を書き換える必要があります。

①バッファサイズ

```
#define YBUFSIZ 512                  //バッファサイズ 512バイト以上512バイト単位で確保すると効率がよい
```

バッファサイズを指定します。バッファサイズは 1 以上の値であれば任意ですが、512バイト以上、512バイト単位で確保すると効率が良いです。

②押し戻せる文字の個数

```
#define YBACKSIZ        1            //押し戻せる文字の個数
```

yungetc()関数によって押し戻せる文字の最大個数を指定します。最低１です。yungetc()関数を使用しないのであれば１のままにしておいて下さい。

③同時にオープンできるファイル数

```
#define YFOPEN_MAX      6            //最大同時オープンファイル数
```

同時にオープンできるファイル数を指定します。同時にオープンできるファイル数はyfconfig.hでもありましたが、そこで設定した値以下にします。通常は同じ値を指定します。

３．３　コンフィグレーション後の再構築

yfconfig.h と ystdio.h の変更が終了したら、YellowIDE のメニューの（プロジェクト）→（再構築）で再構築して下さい。（メイク）ではなく（再構築）であることに注意して下さい。（メイクだど必要なファイルがコンパイルされない場合があります。）

３．４　セーフシステム使用時のリカバリ処理について

セーフシステムが有効な状態で、電源断などのために FAT ファイルシステムに異常が見つかった場合、YS-FILE のデフォルトの設定では、メッセージを表示したあと、修復が始まるようになっています。

しかし、メッセージを独自のものに変更したい、あるいは修復は時間がかかるので、メッセージだけ表示させたいなどの選択をしたい場合があります。

その場合はデバイスドライバ作成のプロジェクトで DEVICE.C の以下の関数を変更します。

```
/*******************************************************************
*****   ファイルシステムに異常が見つかった場合の処理をここに書く*****
*****   なお、修復するには yfRecoveryFileSystem()を呼ぶ         *****
*****   引数     DriveNo ドライブ番号                           *****
*****   戻り値   なし                                           *****
*******************************************************************/
void yfFileSystemError(int DriveNo)
{
        puts("ファイルシステム修復します");
        yfRecoveryFileSystem(DriveNo);
        puts("修復完了");
}
```

yfRecoveryFileSystem 関数は修復のための関数です。この関数を呼ばなければ修復はされません。

第４章　システムコール詳細

４．１　エラーコード一覧

定義シンボル	数値	意味
ERR_NODISK	1	ディスクなし
ERR_NOFILE	2	ファイルなし
ERR_NOFORMAT	3	フォーマットされていない
ERR_FATFULL	4	FAT 領域に空きなし
ERR_UNSPTFORMAT	5	未サポートのフォーマット
ERR_ILLNAME	6	名前が不正
ERR_ILLDRIVE	7	不正なドライブ
RRR_DUPNAME	8	既にこの名前はある
ERR_DISKCHANGE	9	ディスクが変更された
ERR_READERROR	10	リードエラー
ERR_NODIR	11	ディレクトリなし
ERR_NOPATH	12	パスが存在しない
ERR_OPENFULL	13	これ以上オープンできない
ERR_NOOPEN	14	ファイルがオープンされていない
ERR_CURDIR	15	対象はカレントディレクトリ
ERR_OPENEDFILE	16	対象はすでにオープンされている
ERR_READONLY	17	対象はリードオンリーファイル
ERR_NOTEMPTY	18	空のディレクトリでない
ERR_ILLPARA	19	不正なパラメータ
ERR_WRITELIMIT	20	ライトリミッタによる制限
ERR_MSGMAX	20	エラーメッセージの数

yfInitFileSystem

ファイルシステムの初期化

書式

void yfInitFileSystem(void)

引数

なし

戻り値

なし

説明

　ファイルシステムを初期化します。ファイルシステムを使用する前に一回だけ実行して下さい。

使用例

```
yfInitFileSystem();
```

yfFindFirst

最初に一致するファイルの検索

書式

int yfFindFirst(char *FileName, BYTE Atr, FindDTA *pDta, DIR_ENTRY *pDir)

引数

FileName	ファイルまたはディレクトリ名
Atr	ファイル属性
pDta	検索情報構造体へのポインタ
pDir	ディレクトリ情報構造体へのポインタ

戻り値

ファイルが見つかった場合は 0、それ以外の場合はエラーコード(0 以外)を返す。

説明

FileName に最初に一致するファイルまたはディレクトリを検索します。ファイル名にはワイルドカード(*、?）が使用できます。ファイルが見つかった場合は 0 を返します。見つからない、またはエラーの場合はエラーコードを返します。

Atr にはファイルの属性を設定します。特定のファイルを探したい場合は Atr の対応するビットに 1 をセットします。Atr が 0x1f の場合はすべてのファイルを探します。Atr が 0 の場合は通常ファイルのみ検索します。

Atr:

bit0	リードオンリー
bit1	隠ファイル
bit2	システムファイル
bit3	ボリュームラベル
bit4	サブディレクトリ

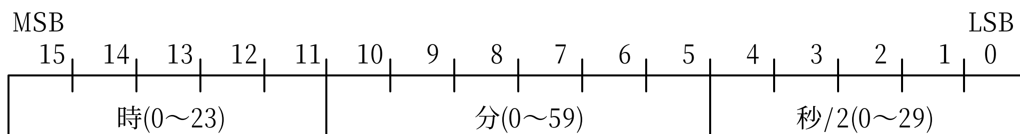
pDta は検索条件を格納する FindDTA 構造体へのポインタです。ユーザは構造体の領域を確保してそのポインタを渡すだけで、この構造体の中身について知る必要はありません。

ファイルまたはディレクトリが見つかった場合は、ファイル情報は DIR_ENTRY 構造体

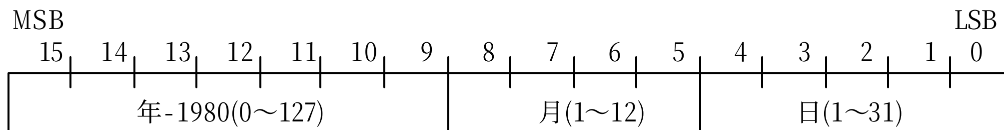
に格納されます。DIR_ENTRY 構造体は YSFILE.H の中で次のように定義されています。

```
//ディレクトリエントリ
typedef struct _DIR_ENTRY {
        BYTE    FileName[8];        //ファイル名
        BYTE    FileExt[3];         //拡張子
        BYTE    Atr;                //属性
        WORD    Time;               //時刻
        WORD    Date;               //日付
        DWORD   StartClus;          //開始クラスタ
        DWORD   Size;               //ファイルサイズ
        DWORD   Sect;               //このディレクトリエントリがあるセクタの番号
        WORD    SectOffset;         //このディレクトリエントリのセクタ内オフセット
} DIR_ENTRY;
```

時刻 Time は以下のように定義されます。秒は 2 秒単位であることに注意して下さい。



日付 Date は以下のように定義されます。年は 1980 年を基準とした値になります。



使用例

2番目以降に一致するファイルは yfFindNext システムコールで検索します。このシステムコールとセットで使われることが多いです。

```
FindDTA dta;
DIR_ENTRY dir;
if (yfFindFirst("*.*", 0x1f, &dta, &dir) == 0) {
	PrintDir(&dir);	//ファイル情報を表示する関数
	while (yfFindNext(&dta, &dir) == 0) {
		PrintDir(&dir);
	}
}
```

yfFindNext

次に一致するファイルの検索

書式

int yfFindNext(FindDTA *pDta, DIR_ENTRY *pDir)

引数

pDta	検索情報構造体へのポインタ
pDir	ディレクトリ情報構造体へのポインタ

戻り値

ファイルが見つかった場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

yfFindFirst によって検索した後に次に一致するファイルを検索します。事前に yfFindFirst が実行されていることが必要です。pDta は yfFindFirst で使用した同じ構造体です。さらにファイルを検索したい場合、続けてこのシステムコールを呼びます。

pDta は検索条件を格納する FindDTA 構造体へのポインタです。詳細は yfFindFirst の項を参照して下さい。

使用例

```
FindDTA dta;
DIR_ENTRY dir;
if (yfFindFirst("*.*", 0x1f, &dta, &dir) == 0) {
        PrintDir(&dir);    //ファイル情報を表示する関数
        while (yfFindNext(&dta, &dir) == 0) {
                PrintDir(&dir);
        }
}
```

yfMakeDir

ディレクトリの作成

書式

int yfMakeDir(const char *Path)

引数

Path　　　　　　ディレクトリ名

戻り値

成功した場合は 0、それ以外の場合はエラーコード (0 以外)を返す。

説明

ディレクトリを作成します。作成に成功した場合は 0、それ以外の場合はエラーコードを返します。

使用例

```
if (yfMakeDir("DATA")) {
        puts("ディレクトリ作成失敗");
}
```

yfMakeFile

ファイルの作成

書式

int yfMakeFile(const char *Path)

引数

Path　　　　　　ファイル名

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルを作成します。作成に成功した場合は 0、それ以外の場合はエラーコードを返します。作成されるファイルはサイズ 0 の空ファイルです。

使用例

```
if (yfMakeFile("TEST.DAT")) {
        puts("ファイル作成失敗");
}
```

yfDelete

ファイルまたはディレクトリの削除

書式

int yfDelete(const char *Path)

引数

Path　　　　　ファイルまたはディレクトリ名

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリを削除します。削除に成功した場合は 0、それ以外の場合はエラーコードを返します。ディレクトリを削除する場合はディレクトリの中身が空である必要があります。空でない場合は削除エラーになります。

使用例

```
if (yfDelete("TEST.DAT")) {
        puts("ファイル削除失敗");
}
```

yfRename

ファイルまたはディレクトリ名の変更

書式

int yfRename(const char *Path, const char *NewName)

引数

Path	ファイルまたはディレクトリの古い名前
NewName	新しい名前

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリの名前を変更します。成功した場合は 0、それ以外の場合はエラーコードを返します。新しい名前のファイルまたはディレクトリが既に存在する場合はエラーとなります。

使用例

```
if (yfRename("TEST.DAT", "NEWNAME.DAT")) {
        puts("ファイル名変更失敗");
}
```

yfGetAtr

ファイルまたはディレクトリの属性取得

書式

int yfGetAtr(const char *Path, BYTE *Atr)

引数

Path	ファイルまたはディレクトリ名
Atr	属性を格納するBYTE変数へのポインタ

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリの属性を取得します。成功した場合は 0、それ以外の場合はエラーコードを返します。属性は以下のように定義されます。通常ファイルの場合はAtr=0x20です。保存ビット(bit5)は常に1が読まれます。

Atr:

bit0	リードオンリー
bit1	隠ファイル
bit2	システムファイル
bit3	ボリュームラベル
bit4	サブディレクトリ
bit5	保存ビット(常に 1)

使用例

```
BYTE Atr;
if (yfGetAtr("TEST.DAT", &Atr)) {
	puts("属性取得失敗");
}
printf("Atr=%02X¥n", Atr&0xff);
```

yfSetAtr

ファイルまたはディレクトリの属性変更

書式

int yfGetAtr(const char *Path, BYTE Atr)

引数

Path	ファイルまたはディレクトリ名
Atr	変更する属性

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリの属性を変更します。成功した場合は 0、それ以外の場合はエラーコードを返します。属性は以下のように定義されます。通常ファイルの場合Atr=0x20です。ボリュームラベル、サブディレクトリの各属性は変更することができません。保存ビットは常に1に設定して下さい。

Atr:

bit0	リードオンリー
bit1	隠ファイル
bit2	システムファイル
bit3	ボリュームラベル
bit4	サブディレクトリ
bit5	保存ビット(常に 1)

使用例

```
if (yfSetAtr("TEST.DAT", 0x21)) {     //リードオンリー属性にする
        puts("属性変更失敗");
}
```

yfGetTime

ファイルまたはディレクトリの時刻日付の取得

書式

int yfGetTime(const char *Path, struct tm *time)

引数

Path	ファイルまたはディレクトリ名
time	ANSI の time.h で定義される時間構造体へのポインタ

戻り値

成功した場合は 0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリの時刻日付を取得します。成功した場合は 0、それ以外の場合はエラーコードを返します。時刻と日付は ANSI 標準ヘッダファイル time.h で定義される struct tm 構造体に格納されます。この構造体については C 言語の書籍を参照して下さい。

使用例

```
#include <time.h>

struct tm DateTime;
if (yfGetTime("TEST.DAT", &DateTime)) {
        puts("時刻日付取得失敗");
}
printf("File Time = %s¥n", asctime(&DateTime));   //時刻日付を表示
```

yfSetTime

ファイルまたはディレクトリの時刻日付の変更

書式

int yfSetTime(const char *Path, struct tm *time)

引数

Path	ファイルまたはディレクトリ名
time	ANSI の time.h で定義される時間構造体へのポインタ

戻り値

成功した場合は 0、それ以外の場合はエラーコード(0 以外)を返す。

説明

ファイルまたはディレクトリの時刻日付を変更します。成功した場合は 0、それ以外の場合はエラーコードを返します。時刻と日付は ANSI 標準ヘッダファイル time.h で定義される struct tm 構造体に事前に格納しておきます。この構造体については C 言語の書籍を参照して下さい。

使用例

```
#include <time.h>


struct tm DateTime;
//2005 年 3 月 15 日に変更
DateTime.tm_sec = 0;
DateTime.tm_min = 0;
DateTime.tm_hour = 0;
DateTime.tm_mday = 15;
DateTime.tm_mon = 2;        //3 月
DateTime.tm_year = 105;    //1900+105 = 2005 年
if (yfSetTime("TEST.DAT", &DateTime)) {
        puts("時刻日付変更失敗");
}
```

yfGetCurDir

カレントディレクトリの取得

書式

int yfGetCurDir(char *Path)

引数

Path　　　　　　カレントディレクトリ名の格納先ポインタ

戻り値

常に 0 を返す。

説明

現在のカレントディレクトリ名を Path に格納します。

使用例

```
char        CurDir[256];
yfGetCurDir(CurDir);
puts(CurDir);
```

yfChangeCurDir

カレントディレクトリの変更

書式

int yfChangeCurDir(const char *Path)

引数

Path　　　　　変更するカレントディレクトリ名

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

現在のカレントディレクトリ名を Path に変更します。

使用例

```
if (yfChangeCurDir("A:¥¥DATA")) {
        puts("カレントディレクトリ変更失敗");
}
```

yfGetCurDrv

カレントドライブの取得

書式

BYTE yfGetCurDrv(void)

引数

なし

戻り値

カレントドライブ番号を返す。

説明

現在のカレントドライブ番号を返します。ドライブ番号は A: B: C: .....の順に 0、1、２.....です。

使用例

```
printf("DriveNo = %d¥n", yfGetCurDrv());
```

yfChangeCurDrv

カレントドライブの変更

書式

int yfChangeCurDrv(int DriveNo)

引数

Path　　　　　　変更するカレントドライブ番号

戻り値

成功した場合は0、それ以外の場合はエラーコード(0 以外)を返す。

説明

カレントドライブを DriveNoに変更します。ドライブ番号はA: B: C: .....の順に0、1、２.....です。成功した場合は0、それ以外の場合はエラーコードを返します。

使用例

```
if (yfChangeCurDrv(1)) {
        puts("カレントドライブ変更失敗");
}
```

yfGetOpenFileCount

現在オープンされているファイル数の取得

書式

WORD yfGetOpenFileCount(void)

引数

なし

戻り値

オープンされているファイル数を返す。

説明

現在オープンされているファイル数を返します。プログラム終了時に 0 であることをチェックするのに使用できます。

使用例

```
if (yfGetOpenFileCount()) {
        puts("クローズしていないファイルあり");
        yfCloseAll();        //すべてのファイルをクローズ
}
```

yfCreate

ファイルの作成とオープン

書式

int yfCreate(const char *Path, BYTE attr, WORD *Err)

引数

Path	ファイル名
attr	オープン属性(未サポート)
Err	エラーコード格納先ポインタ

戻り値

ファイルディスクリプタを返す。エラーの場合は-1 を返す。

説明

Path で示されるファイルを作成してオープンします。ファイルが既に存在する場合はそのファイルは一旦削除されます。attr でオープン属性を指定できますが現在は未サポートです。成功した場合はファイルディスクリプタを返します。エラーの場合は-1 を返し、Err が NULL でなければエラーコードを Err に格納します。

使用例

```
WORD Err;
int fd;
fd = yfCreate("TEST.DAT", 0, &Err);
if (fd == -1)) {
        puts("ファイル作成&オープン失敗");
}
```

yfOpen

ファイルのオープン

書式

int yfOpen(const char *Path, BYTE attr, WORD *Err)

引数

Path	ファイル名
attr	オープン属性(未サポート)
Err	エラーコード格納先ポインタ

戻り値

ファイルディスクリプタを返す。エラーの場合は-1 を返す。

説明

Path で示されるファイルをオープンします。ファイル存在しない場合はエラーとなります。attr でオープン属性を指定できますが現在は未サポートです。成功した場合はファイルディスクリプタを返します。エラーの場合は-1 を返し、Err が NULL でなければエラーコードを Err に格納します。

使用例

```
WORD Err;
int fd;
fd = yfOpen("TEST.DAT", 0, &Err);
if (fd == -1) {
        puts("ファイルオープン失敗");
}
```

yfClose

ファイルのクローズ

書式

int yfClose(int FileId)

引数

FileId	ファイルディスクリプタ

戻り値

成功した場合は0、それ以外の場合はエラーコードを返す。

説明

FileIdで示されるファイルをクローズします。成功した場合は0を返します。エラーの場合はエラーコードを返します。エラーが起きるのはオープンされていないファイルをクローズしようとした時だけです。そのため、エラーチェックを省略しても構いません。

使用例

```
WORD Err;
int fd;
fd = yfOpen("TEST.DAT", 0, &Err);
if ((fd == -1) {
        puts("ファイルオープン失敗");
}
....
yfClose(fd);
```

yfCloseAll

オープンされているファイルすべてのクローズ

書式

int yfCloseAll(void)

引数

なし

戻り値

常に 0 を返す。

説明

オープンされているファイルすべてをクローズします。戻り値は常に 0 です。

使用例

```
yfCloseAll();
```

yfWrite

ファイルのライト

書式

DWORD yfWrite(int FileId, const char *buf, DWORD size, WORD *Err)

引数

FileId	ファイルディスクリプタ
buf	書き込みデータの格納先ポインタ
size	書き込むサイズ(バイト単位)
Err	エラーコード格納先ポインタ

戻り値

書き込んだサイズを返す。エラーの場合は-1 を返す。

説明

FileId で示されるファイルにデータを書き込みます。データは buf に size バイト格納されているとします。成功した場合は実際に書き込んだサイズ返します。エラーの場合は-1 を返し、Err が NULL でなければエラーコードを Err に格納します。

使用例

```
//ファイルのコピー
int fd1,fd2,size;
char buf[512];
fd1 = yfOpen("TEST1.DAT", 0, NULL); if (fd1==-1) return;
fd2 = yfCreate("TEST2.DAT", 0, NULL); if (fd2==-1) { yfClose(fd1); return; }
while ((size = yfRead(fd1, buf, 512, NULL)) != -1) {
        if (yfWrite(fd2, buf, size, NULL) < size) {
                puts("ライトエラー");
        }
}
yfClose(fd1);
yfClose(fd2);
```

yfRead

ファイルのリード

書式

DWORD yfRead(int FileId, char *buf, DWORD size, WORD *Err)

引数

FileId	ファイルディスクリプタ
buf	読み込んだデータの格納先ポインタ
size	読み込んだサイズ(バイト単位)
Err	エラーコード格納先ポインタ

戻り値

読み込んだサイズを返す。エラーの場合は-1 を返す。

説明

FileId で示されるファイルからデータを読み込みます。読み込んだデータは buf に格納します。成功した場合は実際に読み込んだサイズ返します。エラーの場合は-1 を返し、Err が NULL でなければエラーコードを Err に格納します。

使用例

```
//ファイルのコピー
int fd1,fd2,size;
char buf[512];
fd1 = yfOpen("TEST1.DAT", 0, NULL); if (fd1==-1) return;
fd2 = yfCreate("TEST2.DAT", 0, NULL); if (fd2==-1) { yfClose(fd1); return; }
while ((size = yfRead(fd1, buf, 512, NULL)) != -1) {
        if (yfWrite(fd2, buf, size, NULL) < size) {
                puts("ライトエラー");
        }
}
yfClose(fd1);
yfClose(fd2);
```

yfSeek

ファイルのシーク

書式

DWORD yfSeek(int FileId, SDWORD offset, int Mode, WORD *Err)

引数

FileId	ファイルディスクリプタ
offset	移動サイズ
Mode	モード（基準点)
Err	エラーコード格納先ポインタ

戻り値

移動後のアクセスポイントを返す。エラーの場合は-1 を返す。

説明

FileId で示されるファイルのアクセスポイントを移動します。移動の基準点は Mode で指定します。以下の通りです。

0:ファイルの最初から

1:現在位置から

2:ファイルの最後から

移動サイズは offset で指定します。成功した場合は移動後のファイル先頭を基準としたアクセスポイントを返します。エラーの場合は-1 を返し、Err が NULL でなければエラーコードを Err に格納します。移動がファイルのサイズを超える場合はファイルサイズが大きくなります。大きくなった分のデータは不定です。またファイル先頭より前に移動することはできません。

使用例

```
//先頭から 100 バイト目に移動
if (yfSeek(fd1, 100, 0, NULL) == -1) {
        puts("移動失敗");
}
```

yfGetBpbAdrs

BPB アドレスの取得

書式

BPB_INFO *yfGetBpbAdrs(int DriveNo, WORD *Err)

引数

DriveNo	ドライブ番号
Err	エラーコード格納先ポインタ

戻り値

BPB のアドレスを返す。エラーの場合は NULL を返す。

説明

ファイルシステムはディスクや FAT の情報を BPB_INFO という構造体に格納しています。この構造体のアドレスを得ることでディスクや FAT の情報を得ることができます。

成功した場合は BPB_INFO 構造体のアドレスを返します。失敗した場合は NULL を返し、Err が NULL でなければエラー情報を Err に格納します。

BPB_INFO 構造体の詳細は以下の通りです。

```
typedef struct _BPB_INFO {
        WORD    BytsPerSec;       // 1 セクタあたりのバイト数
        BYTE    SecPerClus;       // 1 クラスタあたりのセクタ数
        WORD    RsvdSecCnt;       // 予約セクタの数
        BYTE    NumFATs;          // FAT の数
        WORD    RootEntCnt;       // ルートディレクトリエントリの数 FAT32 の場合
=0
        WORD    TotSec16;         // トータルセクタ数 FAT32 の場合=0
        BYTE    Media;            // メディアタイプ
        WORD    FATSz16;          // FAT のセクタ数 FAT32 の場合=0
        WORD    SecPerTrk;        //トラックあたりのセクタ数
        WORD    NumHeads;         // ヘッド数
        DWORD   HiddSec;          // パーテーション先頭セクタの LBA
        DWORD   TotSec32;         // トータルセクタ数
        DWORD   FATSz32;          // FAT のセクタ数
        WORD    ExtFlags;         // 拡張フラグ
        WORD    FSVer;            // ファイルシステムのバージョン　現在は 0
        DWORD   RootClus;         // ルートディレクトリの先頭クラスタ番号
        WORD    FSInfo;           // 通常は 1
        WORD    BkBootSec;        // ブートセクタのバックアップのセクタ番号
        //追加情報
        DWORD   FatMask;          //FAT マスク
        DWORD   VolID;            //ボリューム ID
        BYTE    VolLab[11];       //ボリュームラベル
        BYTE    FatType;          // 0-FAT12 1-FAT16 2-FAT32
        DWORD   FirstSecNo;       //データ領域最初のセクタ番号=クラスタ 2
        DWORD   CountOfClust;     //クラスタ数
} BPB_INFO;
```

使用例

```
//ディスクの空き サイズを出力する
DWORD EmpCls;            //空きクラスタ
BPB_INFO *pBpb;
if (yfGetEmptyClust(0, &EmpCls)) {
        return;
}
if ((pBpb = yfGetBpbAdrs(0, NULL)) != NULL) {
        DWORD EmpSize;
        EmpSize = EmpCls * pBpb->SecPerClus * SECTOR_SIZE;
        printf("EmptySize = %ld¥n", EmpCls);
}
return;

//SECTOR_SIZE は 512 で YSFILE.H で定義されている
```

yfGetEmptyClust

空きクラスタ数の取得

書式

int yfGetEmptyClust(int DriveNo, DWORD *cnt)

引数

DriveNo	ドライブ番号
cnt	空きクラスタ数の格納先ポインタ

戻り値

成功した場合は0、それ以外の場合はエラーコードを返す。

説明

ディスクの空きクラスタ数を cnt に格納します。成功した場合は 0 を返します。それ以外の場合はエラーコードを返します。

使用例

```
//ディスクの空きサイズを出力する
DWORD EmpCls;           //空きクラスタ
BPB_INFO *pBpb;
if (yfGetEmptyClust(0, &EmpCls)) {
        return;
}
if ((pBpb = yfGetBpbAdrs(0, NULL)) != NULL) {
        DWORD EmpSize;
        EmpSize = EmpCls * pBpb->SecPerClus * SECTOR_SIZE;
        printf("EmptySize = %ld¥n", EmpCls);
}
return;

//SECTOR_SIZEは 512 で YSFILE.H で定義されている
```

yfGetLabel

ボリュームラベルの取得

書式

int yfGetLabel(int DriveNo, char *Vol)

引数

DriveNo	ドライブ番号
Vol	ボリュームラベルの格納先ポインタ

戻り値

成功した場合は0、それ以外の場合はエラーコードを返す。

説明

ボリュームラベルをVolで示される領域に格納します。ボリュームラベルは半角で11文字までですのでVolは12バイト確保して下さい。またボリュームラベルがない場合はVolに"NONAME"が格納されます。

使用例

```
#include <ysfile.h>

char Label[12];
int ret;
if ((ret = yfGetLabel(0, Label)) == 0) {
        if (!strcmp(Label, "NONAME"))
                printf("ボリュームラベルはありません¥n");
        else
                printf("ボリュームラベルは %s  です¥n", Label);
}
else {
        printf("システムコールエラー %s¥n", yfErrorMsg(ret));
}
```

yfSetLabel

ボリュームラベルの設定・変更

書式

int yfSetLabel(int DriveNo, char *Vol)

引数

DriveNo	ドライブ番号
Vol	ボリュームラベルの格納先ポインタ

戻り値

成功した場合は0、それ以外の場合はエラーコードを返す。

説明

ボリュームラベルを Vol で示される文字列に変更します。ボリュームラベルがない場合は新しく作られます。ボリュームラベルは半角で 11 文字までですので 11 文字を超えないようにして下さい。全角の場合は 5 文字までです。

使用例

```
#include <ysfile.h>

int ret;
if ((ret = yfSetLabel(0, "NEWNAME")) != 0) {
        printf("システムコールエラー %s¥n", yfErrorMsg(ret));
}
```

yfDelLabel

ボリュームラベルの削除

書式

int yfDelLabel(int DriveNo)

引数

DriveNo　　ドライブ番号

戻り値

成功した場合は0、それ以外の場合はエラーコードを返す。

説明

ボリュームラベルを削除します。

使用例

```
#include <ysfile.h>

int ret;
if ((ret=yfDelLabel(0)) != 0) {
	printf("システムコールエラー %s¥n", yfErrorMsg(ret));
}
```

yfErrorMsg

エラーメッセージ文字列の取得

書式

char *yfErrorMsg(int err_no)

引数

err_no　　　エラー番号

戻り値

エラーメッセージ文字列のポインタを返します。エラー番号が不正な場合はNULLを返します。

説明

エラーメッセージ文字列のポインタを返します。エラー番号が不正な場合はNULLを返します。

使用例

```
#include <ysfile.h>

int ret;
if ((ret=yfDelLabel(0)) != 0) {
        printf("システムコールエラー %s¥n", yfErrorMsg(ret));
}
```

yfCheckFat

FAT のチェック

書式

int yfCheckFat(int DriveNo)

引数

DriveNo　　　　ドライブ番号

戻り値

0　正常

1　FAT に異常あり

2　読み込みエラー

3　ディスクがない

説明

2 つある FAT 領域を比較して異常を調べます。2 つの FAT が完全に一致した場合に 0 を返します。何らかのエラーがある場合は 0 以外の値を返します。

ディスク容量が大きい場合、この関数の実行には時間がかかります。

使用例

```
#include <ysfile.h>

int ret;
if ((ret=yfCheckFat(0)) != 0) {
        printf("FAT 異常あり");
}
```

yfRecoveryFileSystem

ファイルシステムの修復

書式

int yfRecoveryFileSystem(int DriveNo)

引数

DriveNo　　　ドライブ番号

戻り値

0　修復成功

1　読み込みエラー

2　修復失敗

説明

YS-FILE は書き込み途中で電源断が起こった場合にファイルシステムを修復する機能があります。電源断が起こりファイルシステムに矛盾が生じた場合、次回起動時にデバイスドライバファイル中の yfFileSystemError 関数が呼ばれます。この関数の中に yfRecoveryFileSystem を記述しておけば修復することができます。

修復には時間がかかりますので、メッセージを表示するか LED を点灯させるなど、シグナルを記述しておいた方が良いでしょう。

使用例

```
//デバイスドライバファイル中
//の yfFileSystemError の中に記述する

void yfFileSystemError(int DriveNo)
{
        puts("ファイルシステム修復します");
        yfRecoveryFileSystem(DriveNo);
        puts("修復完了");
}
```

第５章　ANSI 関数詳細

ystdio.h で定義されるシンボル

●マクロ

シンボル	初期値	意味
YBUFSIZ	512	YFILE構造体中のシステムバッファのサイズ
YBACKSIZ	2	yungetcでの押し戻せる文字の個数
YFILENAME_MAX	63	ファイル名の最大長
YFOPEN_MAX	2	最大同時オープンファイル数
YL_tmpnam	13	一時ファイル名の最大長
YTMP_MAX	32767	一時ファイル名生成回数の最大
YSEEK_SET	0	yfseek で、ファイルの先頭が基準
YSEEK_CUR	1	yfseek で、ファイルの現在位置が基準
YSEEK_END	2	yfseek で、ファイルの最後が基準
YEOF	(-1)	ファイルの終わり
_YIOFBF	0	ysetvbufで、バッファ制御あり
_YIOLBF	1	ysetvbufで、行バッファ
_YIONBF	2	ysetvbufで、バッファなし

●型

シンボル	意味
yfpos_t	ファイルの位置を格納する型。unsigned long と等価
YFILE	オープンされたファイルの情報を格納する構造体

yfopen

ファイルのオープン

書式

YFILE *yfopen(const char *s, const char *f)

引数

s	ファイル名文字列
f	オープンモード文字列

戻り値

オープンしたファイルの YFILE 構造体へのポインタを返す。失敗した場合は NULL を返す。

説明

ファイル名 s のファイルをオープンモード f に従ってオープンします。成功した場合はオープンした YFILE 構造体のポインタを返します。失敗した場合は NULL を返します。以後のファイルへのアクセスはYFILE構造体を通して行います。

オープンモード文字列に含まれる文字によってオープンモードが決まります。

r　読み込みモードでオープンする。ファイルが存在しない場合はエラー

w　書き込みモードでオープンする。ファイルが存在する場合はファイルサイズが0になる。ファイルが存在しない場合は新たに作られる。

a　追加モードでオープンする。既存のファイルの最後から書き込むためにオープンする。ファイルが存在しない場合は新たに作られる。

b　バイナリモードでオープンする。r/w/aと組み合わせて使う。

＋　更新モードでオープンする。ファイルに対して読み書き両方が可能。ただしバイナリモードの時のみ有効。r/w/aと組み合わせて使う。

許されるオープンモードは以下のいずれかである。

"r"	"rb"	"r+b"
"w"	"wb"	"w+b"
"a"	"ab"	"a+b"

　バイナリモードでない時は自動的にテキストモードになります。バイナリモードではファイル内のデータはそのまま処理されます。テキストモードではデータは以下のように処理されます。

'¥r'の読み込みは無視される。
'¥n'の書き込みは'¥r'、'¥n'の 2 文字の書き込みとなる。

使用例

```
#include <ystdio.h>

YFILE *fp;
if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
        puts("オープン失敗");
        return;
}
```

yfreopen

ファイルの再オープン

書式

YFILE *yfreopen(const char *s, const char *f, YFILE *fp)

引数

s	代替ファイル名文字列
f	オープンモード文字列
fp	既存 YFILE 構造体へのポインタ

戻り値

YFILE構造体をへのポインタを返す。失敗した場合はNULL を返す。

説明

既にオープンしているファイルの YFILE 構造体を別のファイルのオープンに割り当てます。既存ファイルは一旦クローズされ、ファイル名s、オープンモード f で再オープンされます。成功した場合は YFILE 構造体へのポインタを、失敗した場合は NULL を返します。オープンモードについてはyfopen の項を参照して下さい。

補足

この関数は stdout、stdin、stderr のファイルへのリダイレクトを可能にするために存在しますが､YS-FILE の場合 stdout 等存在しませんのであまり使用頻度は多くありません。

yfclose

ファイルのクローズ

書式

int yfclose(YFILE *fp)

引数

fp　　　オープン済み YFILE構造体へのポインタ

戻り値

成功した場合は0、失敗した場合は YEOF(=-1)を返す。

説明

既にオープンしているファイルをクローズします。成功した場合は 0 を、失敗した場合は YEOF(=-1)を返します。クローズに先立ちバッファリングしてある文字がすべて書き出されます。ytmpfile でオープンされた一時ファイルはクローズ後、削除されます。

使用例

```
#include <ystdio.h>

YFILE *fp;
if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
        puts("オープン失敗");
        return;
}
yfclose(fp);
```

ysetvbuf

YFILEへのバッファ割り当て

書式

int ysetvbuf(YFILE *fp, char *buf, int mode, size_t size)

引数

fp	オープン済み YFILE構造体へのポインタ
buf	ユーザ定義バッファの先頭アドレス
mode	モード
size	バッファサイズ

戻り値

成功した場合は0、失敗した場合は YEOF(=-1)を返す。

説明

既にオープンされているファイルの YFILE 構造体にユーザ定義のバッファを割り当てます。ファイルごとにバッファサイズを変えたい場合、一時的に大量のバッファを使用したい場合に使います。また mode で行バッファやバッファなしの制御もできます。mode に許される値は以下の通りです。

_YIOFBF	入出力を完全にバッファリング（デフォルト）
_YIOLBF	入出力を行バッファリング
_YIONBF	入出力のバッファリングなし

bufがNULLかsizeが0の場合、最初から用意されたシステムバッファが利用されます。ユーザ定義のバッファが利用されるのは buf が NULL でなくかつ size が 1 以上の場合です。

buf が NULL で size が 1 以上の場合はシステムバッファのサイズが size に切りつめられます。ただし、size がシステムバッファのサイズを超える場合は上限がシステムバッファサイズになります。

システムバッファサイズはYS-FILEのコンフィグレーションで変更できます。またシステムバッファのサイズはシンボル YBUFSIZ で参照できます。buf と size の仕様は ANSI

の規格と多少異なります。

使用例

```
#include <ystdio.h>

//ユーザ定義のバッファを割り当てる
YFILE *fp;
char      UserBuf[1000];    //ユーザバッファ

if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
          puts("オープン失敗");
          return;
}
if (ysetvbuf(fp, buf, _YIOFBF, 1000)) {
          puts("バッファ割り当て失敗");
          yfclose(fp);
          return;
}
```

ysetbuf

YFILEへのバッファ割り当て

書式

void ysetbuf(YFILE *fp, char *buf)

引数

fp	オープン済み YFILE構造体へのポインタ
buf	ユーザ定義バッファの先頭アドレス

戻り値

なし

説明

既にオープンされているファイルの YFILE 構造体にユーザ定義のバッファを割り当てます。ysetvbufの簡略版です。バッファのサイズはシステムバッファのサイズと同じです。システムバッファのサイズはYS-FILEのコンフィグレーションで変更できます。またシステムバッファのサイズはYBUFSIZ で参照できます。

使用例

```
#include <ystdio.h>
//ユーザ定義のバッファを割り当てる
YFILE *fp;
char    UserBuf[YBUFSIZ];        //ユーザバッファ

if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
        puts("オープン失敗");
        return;
}
if (ysetbuf(fp, buf)) {
        puts("バッファ割り当て失敗");
        yfclose(fp);
        return;
}
```

yfflush

バッファのフラッシュ

書式

int yfflush(YFILE *fp)

引数

fp　　オープン済み YFILE構造体へのポインタ

戻り値

成功した場合は0、失敗した場合は YEOF(=-1)を返す。

説明

バッファをフラッシュしてすべてのデータをファイルに書き出します。成功した場合は 0 を、失敗した場合は YEOF(=-1)を返します。fp が NULL の場合はオープンされているファイルすべてをフラッシュします。

使用例

```
#include <ystdio.h>
YFILE *fp;
if ((fp = yfopen("TEST.DAT", "w")) == NULL) {
        puts("オープン失敗");
        return;
}
yfputc('A', fp);
yfflush(fp);
```

yfgetc　ygetc

ファイルからの一文字読み込み

書式

int yfgetc(YFILE *fp)

引数

fp　　　　　　　　オープン済み YFILE構造体へのポインタ

戻り値

　読み込んだ一文字(値　0〜255)、失敗した場合あるいはファイルの終わりの場合はYEOF(=-1)を返す。

説明

　オープン済みのファイルから一文字を読み込みます。エラーまたはファイルの終わりの場合は YEOF(=-1）を返します。

　ygetc は yfgetc をマクロで再定義したもので機能は同じです。

使用例

```
#include <ystdio.h>
//一文字単位のファイルのコピー
YFILE *fp1,*fp2;
int c;
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while ((c = yfgetc(fp1)) != YEOF)
        yfputc(c, fp2);
yfclose(fp1);
yfclose(fp2);
```

yfgets

ファイルからの一行読み込み

書式

char *yfgets(char *s, int n, YFILE *fp)

引数

s	読み込んだ一行の格納先ポインタ
n	読み込む文字の最大数
fp	オープン済み YFILE構造体へのポインタ

戻り値

成功した場合は引数 s をそのまま返す。失敗またはファイルの終わりに達し一文字も読み込まれなかった場合ははは NULL を返す。

説明

オープン済みのファイルから最大 n 文字の一行を読み込みます。一行の終わりは'¥n'か YEOF で区別されます。一行が n を超える場合は n 文字読み込んだ時点で読み込みがストップします。一行の終わりには'¥0'が自動的に付加されます。したがって格納先は n+1 バイトの領域が必要です。

エラーがない場合は引数 s がそのまま返されます。エラーがあるかまたはファイルの終わりに達し、一文字も読み込まれなかった場合には NULL が返されます。

使用例

```
#include <ystdio.h>
//一行単位のファイルのコピー
YFILE *fp1,*fp2;
int c;
char buf[1000];
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while (yfgets(buf, 999, fp1) != NULL)
        yfputs(buf, fp2);
yfclose(fp1);
yfclose(fp2);
```

yfputc　yputc

ファイルへの一文字書き込み

書式

int yfputc(int c, YFILE *fp)

引数

c	出力する一文字
fp	オープン済み YFILE構造体へのポインタ

戻り値

書き込んだ文字を返す、失敗した場合は YEOF(=-1)を返す。

説明

オープン済みのファイルへ一文字を書き込みます。書き込みが成功した場合は書き込んだ文字を、エラーの場合は YEOF(=-1）を返します。

yputc は yfputc をマクロで再定義したもので、機能は同じです。

使用例

```
#include <ystdio.h>
//一文字単位のファイルのコピー
YFILE *fp1,*fp2;
int c;
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while ((c = yfgetc(fp1)) != YEOF)
        yfputc(c, fp2);
yfclose(fp1);
yfclose(fp2);
```

yfputs

ファイルへの文字列書き込み

書式

int yfputs(const char *s, YFILE *fp)

引数

s　　書き込む文字列

fp　　オープン済み YFILE構造体へのポインタ

戻り値

成功した場合は 1 を返す。失敗した場合は YEOF(=-1)を返す。

説明

オープン済みのファイルへ文字列を書き込みます。成功した場合は 1 を、エラーがあるあった場合は YEOF(=-1)が返されます。改行('¥n')は最後に付加されません。

使用例

```
#include <ystdio.h>
//一行単位のファイルのコピー
YFILE *fp1,*fp2;
int c;
char buf[1000];
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while (yfgets(buf, 999, fp1) != NULL)
        yfputs(buf, fp2);
yfclose(fp1);
yfclose(fp2);
```

yungetc

ファイル入力バッファへの文字押し戻し

書式

int yungetc(int c, YFILE *fp)

引数

c	押し戻す文字
fp	オープン済み YFILE構造体へのポインタ

戻り値

成功した場合は押し戻した文字 c を返す。失敗した場合は YEOF(=-1)を返す。

説明

オープン済みファイルの入力バッファへ文字を押し戻します。押し戻すのは入力バッファであり、ファイルそのものではありません。書き込みモードでオープンされたファイルへは押し戻しできません。また YEOF(=-1)の押し戻しはできません。押し戻せる文字の個数は YS-FILE のコンフィグレーションで変更できます。初期値は 2 です。

読み込んだ文字と違う文字を押し戻すことも可能です。また読み込みがまったくない状態で押し戻すことも可能です。

使用例

```
#include <ystdio.h>
YFILE *fp
int c;
if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
        return;
}
yungetc('A', fp);
yungetc('B', fp);
c = ygetc(fp);      //'B'が読み込まれる
c = ygetc(fp);      //'A'が読み込まれる
```

yfread

ファイルからのデータ配列読み込み

書式

size_t yfread(void *ptr, size_t size, size_t nobj, YFILE *fp)

引数

ptr	データ配列の先頭ポインタ
size	配列要素 1 個の大きさ
nobj	要素の数
fp	オープン済み YFILE構造体へのポインタ

戻り値

入力できた要素数を返す。

説明

オープン済みファイルから大きさsizeバイト要素をnobj個読み込み､ptrに格納します。実際に読み込んだ要素数を返します。途中でファイルの終わりに達した場合はnobjより小さい値を返す場合があります。読み込むバイト数はsize*nobjバイトになります。

使用例

```
#include <ystdio.h>
//fread と fwrite を使ったファイルのコピー
YFILE *fp1,*fp2;
char buf[1000];
int n;
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while ((n = yfread(buf, 1, 1000, fp1) != 0)
        yfwrite(buf, 1, n, fp2);
yfclose(fp1);
yfclose(fp2);
```

yfwrite

ファイルへのデータ配列書き込み

書式

size_t yfwrite(const void *ptr, size_t size, size_t nobj, YFILE *fp)

引数

ptr	データ配列の先頭ポインタ
size	配列要素 1 個の大きさ
nobj	要素の数
fp	オープン済み YFILE構造体へのポインタ

戻り値

出力できた要素数を返す。

説明

オープン済みファイルへ大きさ size バイトの要素を nobj 個、ptr からファイルに書き込みます。実際に書き込んだ要素数を返します。書き込みエラーの発生により nobj より小さい値を返す場合があります。書き込むバイト数は size*nobj バイトになります。

使用例

```
#include <ystdio.h>
//fread と fwrite を使ったファイルのコピー
YFILE *fp1,*fp2;
char buf[1000];
int n;
if ((fp1 = yfopen("TEST1.DAT", "r")) == NULL) return;
if ((fp2 = yfopen("TEST2.DAT", "w")) == NULL) {yfclose(fp1); return;}
while ((n = yfread(buf, 1, 1000, fp1) != 0)
        yfwrite(buf, 1, n, fp2);
yfclose(fp1);
yfclose(fp2);
```

yfseek

ファイルのシーク

書式

int yfseek(YFILE *fp, long offset, int org)

引数

fp	オープン済み YFILE構造体へのポインタ
offset	移動量
org	移動の原点

戻り値

成功した場合は 0 を返す。失敗した場合は YEOF(=-1)を返す。

説明

オープン済みファイルのアクセス位置を移動します。移動量は offset で移動の原点は org で示します。移動の原点には以下のものがあります。

YSEEK_SET	ファイルの先頭
YSEEK_CUR	ファイルの現在位置
YSEEK_END	ファイルの終端

"a"、"ab"でオープンされた場合はこの関数は機能しません。また"a+", "ab+"でオープンされた場合は読み込みのみ機能します。

バイナリモードとテキストモードでは使用できる機能に差があります。

バイナリモード

原点には YSEEK_SET、YSEEK_CUR、YSEEK_END のすべて使える。

テキストモード

次のいずれかの場合だけが有効です。他の場合は動作不定です。

1 offset=0 かつ org が YSEEK_SET または YSEEK_END

2 offset に yftell 関数の返却値をセットし、かつ org が YSEEK_SET の場合

yfseek を実行すると yungetc で押し戻された文字はすべて破棄されます。

使用例

```
#include <ystdio.h>
YFILE *fp;
long cur;
int c;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL) {
	return;
}
c = yfgetc(fp);		//1
c = yfgetc(fp);		//2
cur = yftell(fp);	//現在位置保存
c = yfgetc(fp);		//3
c = yfgetc(fp);		//4
if (yfseek(fp, cur, YSEEK_SET) == YEOF) {
	puts("シークエラー");
}
c = yfgetc(fp);		//3 番と同じ文字が読まれる
```

yrewind

ファイルの巻き戻し

書式

void yrewind(YFILE *fp)

引数

fp　　　　オープン済み YFILE構造体へのポインタ

戻り値　　なし

説明

オープン済みファイルのアクセス位置をファイルの先頭に戻します。

これは yfseek(fp, 0L, SEEK_SET)と等価です。

使用例

```
#include <ystdio.h>
YFILE *fp;
int c;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL) {
        return;
}
c = yfgetc(fp);     //1
c = yfgetc(fp);     //2
yrewind(fp);        //先頭に戻る
```

yftell

ファイルの現在位置の取得

書式

long yftell(YFILE *fp)

引数

fp　　　　　オープン済み YFILE構造体へのポインタ

戻り値

成功した場合はファイルの現在位置を返す。失敗した場合はYEOF(=-1)を返す。

説明

オープン済みファイルの現在のアクセス位置を返します。失敗した場合は YEOF(=-1)を返します。

使用例

```
#include <ystdio.h>
YFILE *fp;
long cur;
int c;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL) {
        return;
}
c = yfgetc(fp);         //1
c = yfgetc(fp);         //2
cur = yftell(fp);       //現在位置保存
c = yfgetc(fp);         //3
c = yfgetc(fp);         //4
if (yfseek(fp, cur, YSEEK_SET) == YEOF) {
        puts("シークエラー");
}
c = yfgetc(fp);         //3 番と同じ文字が読まれる
```

yfgetpos

ファイルの現在位置の取得

書式

int yfgetpos(YFILE *fp, yfpos_t *ptr)

引数

fp	オープン済み YFILE構造体へのポインタ
ptr	現在位置の格納先ポインタ

戻り値

成功した場合は0 を返す。失敗した場合はYEOF(=-1)を返す。

説明

オープン済みファイルの現在のアクセス位置を ptr に格納します。成功した場合は0 を失敗した場合は YEOF(=-1)を返します。yftell と同じ機能ですが、yftell と yfseek の組み合わせより、yfgetpos と yfsetpos の組み合わせの方が分かりやすいです。

使用例

```
#include <ystdio.h>
YFILE *fp;
yfpos_t cur;
int c;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL) {
        return;
}
c = yfgetc(fp);     //1
c = yfgetc(fp);     //2
if (ygetpos(fp, &cur)) puts("エラー");          //現在位置保存
c = yfgetc(fp);     //3
c = yfgetc(fp);     //4
if (ysetpos(fp, &cur)) puts("エラー");          //元に戻す
c = yfgetc(fp);     //3 番と同じ文字が読まれる
```

yfsetpos

ファイルの現在位置の移動

書式

int yfsetpos(YFILE *fp, const yfpos_t *ptr)

引数

fp	オープン済み YFILE構造体へのポインタ
ptr	移動する位置を保存し場所へのポインタ

戻り値

成功した場合は0 を返す。失敗した場合はYEOF(=-1)を返す。

説明

オープン済みファイルの現在のアクセス位置を ptr に格納されている値に移動します。成功した場合は 0 を失敗した場合は YEOF(=-1)を返します。ptr に格納されている値は fgetpos で得た値である必要があります。したがって yfsetpos は必ず yfgetpos とのセットで使用します。

使用例

```
#include <ystdio.h>
YFILE *fp;
yfpos_t cur;
int c;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL)
        return;
c = yfgetc(fp);     //1
c = yfgetc(fp);     //2
if (ygetpos(fp, &cur)) puts("エラー");        //現在位置保存
c = yfgetc(fp);     //3
c = yfgetc(fp);     //4
if (ysetpos(fp, &cur)) puts("エラー");        //元に戻す
c = yfgetc(fp);     //3 番と同じ文字が読まれる
```

yremove

ファイルの削除

書式

int yremove(const char *fname)

引数

fname　　　　　ファイル名

戻り値

成功した場合は 0 を返す。失敗した場合はファイルシステムが返すエラーコードを返します。

説明

fname で示されるファイルを削除します。成功した場合は 0 を返します。失敗した場合はファイルシステムが返すエラーコードを返します。

使用例

```
#include <ystdio.h>

if (yremove("TEST.DAT")) {
        puts("ファイル削除失敗");
}
```

yrename

ファイル名の変更

書式

int yrename(const char *old, const char *new)

引数

old	元のファイル名
new	新しいファイル名

戻り値

成功した場合は 0 を返す。失敗した場合はファイルシステムが返すエラーコードを返します。

説明

old で示されるファイルの名前を新しいファイル名 new に変えます。既に new というファイルが同じディレクトリに存在する場合はエラーとなります。成功した場合は 0 を返します。失敗した場合はファイルシステムが返すエラーコードを返します。

使用例

```
#include <ystdio.h>

if (yrename("TEST.DAT", "NEWNAME.DAT")) {
        puts("ファイル名変更失敗");
}
```

yfeof

ファイル終端の検出

書式

int yfeof(YFILE *fp)

引数

fp	オープン済み YFILE構造体へのポインタ

戻り値

ファイルの終わりに達している場合は 0 以外の値を返す。ファイルの終わりに達していない場合は 0 を返す。

説明

ファイルの終わりに達しているかどうか調べます。ファイル入力関数の呼び出し直後に使用してファイルの終わりかどうか調べます。ファイル入力関数の呼び出し前に、次の入力がファイルの終わりかどうか事前に調べることはできません。あくまでも入力した結果ファイルの終わりかどうかを判断します。

ファイルの終わりに達している場合は 0 以外の値を返します。ファイルの終わりに達していない場合は0 を返します。

ファイルが一度終わりに達した場合でも、次の操作をした直後はyfeof は 0 を返します。

1 yungetc()で文字を押し戻したとき

2 yfseek()を実行したとき

3 ycrearerr()を実行したとき

使用例

```
#include <ystdio.h>
while (1) {
        yfgets(buf, 100, fp);
        if (yfeof(fp)) {
                break;
        }
}
```

yferror

エラーの検出

書式

int yferror(YFILE *fp)

引数

fp	オープン済み YFILE構造体へのポインタ

戻り値

エラーが発生している場合は 0 以外の値を返す。エラーが発生していない場合は 0 を返す。

説明

ファイル入出力関連の関数を実行した結果、エラーが起こったかどうか調べます。エラーが発生している場合は 0 以外の値を返します。エラーが発生していない場合は 0 を返します。

一度エラーが発生すると、その後エラーが発生しない場合でもエラー状態は保持されます。ycrearerr()によってのみエラー状態は解除されます。

使用例

```
#include <ystdio.h>
while (1) {
        yfgets(buf, 100, fp);
        if (yfeof(fp) || yferror(fp)) {
                break;
        }
}
```

yclearerr

エラー状態、ファイル終端状態のクリア

書式

void yclearerr(YFILE *fp)

引数

fp　　　　オープン済み YFILE構造体へのポインタ

戻り値　　なし

説明

　エラー状態、ファイル終端状態をクリアします。この関数を実行した後、yfeof()、yferror()を実行しても 0 が返されます。

使用例

```
#include <ystdio.h>
YFILE *fp;
if ((fp = yfopen("TEST1.DAT", "r")) == NULL)
        return;

ycrearerr(fp);
```

ytmpnam

一時ファイル名の生成

書式

char *ytmpnam(char *s)

引数

fp　　　　　　　**オープン済み YFILE構造体へのポインタ**

戻り値

作成した一時ファイル名を格納した領域へのポインタ

説明

カレントディレクトリにあるファイル名と重複しないファイル名（一時ファイル名）を作成します。s が NULL でなければ一時ファイル名を s に格納します。s が NULL の場合はこの関数の static 領域に格納し、そのポインタを返します。一時ファイル名の作成に失敗した場合は NULL を返します。

一時ファイル名の形式は以下の通りです。

TMPxxxxx.$$$

xxxxx **は** 00000 **から** 32767 **までの数値です。一時ファイル名の作成に失敗するのは** TMP00000.$$$〜TMP32767.$$$**のすべてのファイルがカレントディレクトリ内にある場合です。**

使用例

```
#include <ystdio.h>

char *s;
if ((s = ytmpnam(NULL)) != NULL) {
        puts(s);
}
```

ytmpfile

一時ファイルの生成

書式

YFILE *ytmpfile(void)

引数

なし

戻り値

作成した一時ファイルのYFILE構造体へのポインタ。失敗した場合はNULLを返す。

説明

カレントディレクトリにあるファイル名と重複しないファイル（一時ファイル）を作成します。一時ファイルの名前の作成方法は ytmpnam の方法に従います。一時ファイルは"wb+"モードでオープンされクローズすると自動的に削除されます。

一時ファイルの作成とオープンに成功すると、そのYFILE構造体へのポインタを返します。失敗した場合はNULLを返します。

使用例

```
#include <ystdio.h>

YFILE *fp;
if ((fp = ytmpfile()) == NULL) {
        puts("一時ファイルの作成に失敗しました");
}

yfclose(fp);        //ファイルは自動的に削除される
```

yfscanf

ファイルからの書式付き入力

書式

int　　yfscanf(YFILE *fp, const char *fmt, ...)

引数

fp	オープン済み YFILE構造体へのポインタ
fmt	入力書式文字列

戻り値

代入された入力項目数を返す、入力失敗の場合はYEOF(=-1)を返す。

説明

ファイルから入力書式に従って読み込みます。この関数の仕様は ANSI 規格に準拠していますので、市販の C 言語の書籍等をご覧下さい。

使用例

```
#include <ystdio.h>

YFILE *fp;
int d1,d2,d3;
if ((fp = yfopen("TEST.DAT", "r")) == NULL) {
        puts("ファイルのオープンに失敗しました");
}
yfscanf(fp, "%d %d %d", &d1, &d2, &d3);
```

yfprintf

ファイルへの書式付き出力

書式

int yfprintf (YFILE *fp, const char *fmt, ...)

引数

fp	オープン済み YFILE構造体へのポインタ
fmt	出力書式文字列

戻り値

変換され転送されたバイト数を返す。

説明

ファイルへ出力書式に従って書き込みます。この関数の仕様は ANSI 規格に準拠していますので、市販の C 言語の書籍等をご覧下さい。

使用例

```
#include <ystdio.h>

YFILE *fp;
int d1,d2,d3;
if ((fp = yfopen("TEST.DAT", "w")) == NULL) {
        puts("ファイルのオープンに失敗しました");
}
d1 = 1;
d2 = 2;
d3 = 3;
yfprintf(fp, "%d %d %d¥n", d1, d2, d3);
```

yvfprintf

ファイルへの書式付き出力(可変個引数集付き)

書式

int yvfprintf(YFILE *fp, const char *fmt, va_list arg)

引数

fp	オープン済み YFILE構造体へのポインタ
fmt	出力書式文字列
arg	可変個引数集

戻り値

変換され転送されたバイト数を返す。

説明

ファイルへ出力書式に従って書き込みます。yprintf と同様ですが、可変個引数集を指定できることが違います。yfprintf 関数の動作を少し変えることに利用できます。この関数の仕様はANSI 規格に準拠していますので、市販の C 言語の書籍等をご覧下さい。

使用例

```
//エラー関数の作成
#include <ystdio.h>
#include <stdarg.h>
void error_out (YFILE *fp, const char *fmt, ...) {
        va_list arg;
        va_start(arg, fmt);
        yprintf(fp, "!!ERROR: ");
        yvfprintf(fp, fmt, arg);
        va_end(arg);
}
呼び出し例
error_out(fp, "i = %d¥n", i);          // !!ERROR i = 10
```

第６章　miniDOS

６．１　miniDOS とは

miniDOS は YS-FILE のサンプルプログラムとして提供されるディスクオペレーティングシステムです。機能は高くありませんが、YS-FILE のサンプルとして ANSI 関数、システムコール関数の使い方の例題になっています。

また、ファイル、ディレクトリ管理ツールとしても使用できます。

６．２　使用できるコンパイラ

miniDOS が使用できるコンパイラは以下の通りです。

１　イエローソフト社製 YC シリーズ C コンパイラ　YCH8 または YCSH

２　printf や fgets など、キーボード、コンソール画面に対して入出力ができるコンパイラ、およびライブラリであること。

３　コンパイラの他にターミナルソフトが必要です。(YellowIDE でも可)

６．３　miniDOS のコンパイル

miniDOS のコンパイル方法、実行方法に関しては「SD カード/USB メモリ開発セット 導入の手引き」の「第４章　YS-FILE の構築とテストプログラムの実行」をご覧下さい。

６．４　miniDOS コマンド一覧

コマンド	意味
ATTRIB	ファイル属性の表示と設定・解除
CD	カレントディレクトリの変更と表示
CHKDSK	ディスクのチェック
COPY	ファイルのコピー
DEL	ファイルの削除
DIR	ファイル名の一覧表示
DUMP	ファイル内容のバイナリ表示
EXIT	miniDOS の終了
LABEL	ボリュームラベルの設定・変更・削除
MD	サブディレクトリの作成
RD	サブディレクトリの削除
REN	ファイル名の変更
RENDIR	サブディレクトリ名の変更
TYPE	ファイル内容のテキスト表示
USER	ユーザ定義コマンド
VOL	ボリュームラベルの表示
SUS	省電力モードへの移行(SD カードのみ）
POW	省電力モードの解除(SD カードのみ)

SUS と POW はイエローソフトの SD カード IO モジュール専用のコマンドです。他のデバイスには使用できません。

ATTRIB

ファイル属性の表示と設定・解除

書式

ATTRIB [+Rまたは-R]　ファイル名

オプション

+R　　読み込み専用属性の設定

-R　　読み込み専用属性の解除

説明

　ファイルの読み込み属性の設定・解除を行います。オプションを省略した場合はファイル属性の表示のみ行います。ファイル名にワイルドカードを使用できます。

CD

カレントディレクトリの変更と表示

書式

CD [パス名]

説明

　カレントディレクトリを変更します。パス名を省略した場合は、現在のカレントディレクトリ名を表示します。

CHKDSK

ディスクのチェック

書式

CD ドライブ名

説明

　ディスクのチェックを行います。実際には詳細なチェックを行っているわけではなく、システムコールの yfCheckFat を実行しています。2 つの FAT 領域を比較して結果を報告します。

COPY

ファイルのコピー

書式

COPY [オプション] ファイル名またはパス名　[ファイル名またはパス名]

オプション

/V　　　コピー後ベリファイを行います。

説明

　ファイル単位、あるいはディレクトリ単位でコピーします。1 番目のファイル名にワイルドカードを使用することができます。許される組み合わせは以下の通りです。

1 番目のファイル名にワイルドカードを使用した場合、 2 番目はディレクトリでなくてはいけない。

●COPY ファイル名 [パス名]

例　COPY　*.C　A:¥SRC

1 番目のファイル名またはパス名にワイルドカードを使用しない場合、許される組み合わせは以下の通りです。

●COPY パス名　[パス名]

　ディレクトリ単位でファイルをコピーします。2 番目のパス名を省略した場合カレントディレクトリにコピーされる

●COPY ファイル名 [パス名]

●COPY ファイル名 [ファイル名]

　１番目のファイルを 2 番目のディレクトリあるいはファイルにコピーします。2 番目のパス名またはファイル名を省略した場合、カレントディレクトリにコピーされる。

DEL

ファイルの削除

書式

DEL　ファイル名

説明

　ファイルを削除します。ファイル名にワイルドカードを使用できます。

DIR

ファイル名の一覧表示

書式

DIR [オプション] [ファイル名またはパス名]

オプション

/P　　一画面分(20 行)表示すると、いったん表示を停止する。

/W　　ワイド表示（1 行に 5 個づつ表示）にする。

/E　　ディスクの空き容量を表示する。

説明

　ファイル名に一致するファイル、あるいはパス名のディレクトリの中にあるファイルすべてを表示する。ファイル名にワイルドカードが使用できます。ファイル名またはパス名を省略した場合はカレントディレクトリ内のファイルを表示します。

DUMP

ファイル内容のバイナリ表示

書式

DUMP [オプション] ファイル名

オプション

/P　　　一画面分(20 行)表示すると、いったん表示を停止する。

説明

　ファイル名内容をバイナリで表示します。（バイナリダンプ）

EXIT

miniDOS の終了

書式

EXIT

説明

　miniDOS を終了します。

LABEL

ボリュームラベルの設定、変更、削除

書式

LABEL [ドライブ名] [ボリュームラベル名]

説明

　ボリュームラベル名を省略した場合、現在のボリュームラベル名を表示します。その後、変更するかどうか問い合わせがあります。問い合わせに対して、リターンキーのみで応答すると、ボリュームラベルを削除するかどうか問い合わせがあります。

　ドライブ名を省略した場合はカレントドライブが選択されます。

　ボリュームラベル名を省略しなかった場合、このボリュームラベル名に変更されます。

MD

サブディレクトリの作成

書式

MD パス名

説明

　サブディレクトリを作成します。

RD

サブディレクトリの削除

書式

RD パス名

説明

　サブディレクトリを削除します。削除するディレクトリの中身は空でなければなりません。

REN

ファイル名の変更

書式

REM 旧ファイル名　新ファイル名

説明

　ファイル名を変更します。

RENDIR

サブディレクトリ名の変更

書式

REM 旧パス名　新パス名

説明

　サブディレクトリの名前を変更します。

TYPE

ファイル内容のテキスト表示

書式

TYPE [オプション] ファイル名

オプション

/P　　　一画面分(20 行)表示すると、いったん表示を停止する。

説明

　ファイル名内容をテキストで表示します。

USER

ユーザコマンドの実行

書式

USER

説明

　ユーザコマンドを実行します。ユーザコマンドは、ユーザが自由に作成するコマンドです。miniDOS のソースファイル中の void ComUser(void)関数に実行させたいプログラムを記述します。

VOL

ボリュームラベルの表示

書式

VOL [ドライブ名]

説明

　ディスクのボリュームラベルを表示します。ドライブ名を省略した場合、カレントドライブが選択されます。

SUS

省電力モードへの移行

書式

SUS

説明

　省電力モードへ移行します。このコマンドはイエローソフトの SD カード IO モジュール専用のコマンドです。他のデバイスには使用できません。

POW

省電力モードの解除

書式

POW

説明

　省電力モードを解除します。このコマンドはイエローソフトの SD カード IO モジュール専用のコマンドです。他のデバイスには使用できません。