

MICRO CADAM Helix
実践操作解説書

ACCESSプログラム開発ガイド
バッチプログラム編

2022年8月

株式会社CAD SOLUTIONS



CADS

このチュートリアルは主にMICRO CADAM Helixの操作経験者で
C言語のプログラミング知識はあるが、
ACCESSのプログラミングは初めてという方を対象としています。

第1章 プログラム開発

1. 作成プログラムの概要
2. プログラム作成用フォルダの準備

第2章 プログラムのコーディングと実行

1. 作成プログラムの全コード紹介
2. メインプログラム(初期化・終了処理)作成
3. コンパイル・リンク方法
4. 入力パラメータの取得
5. 入力パラメータのチェック
6. 図面のオープン・ファイル処理
7. 図面の準備と実行
8. ビュー・子図処理
9. 要素の検索
10. 不表示要素の検出
11. 要素の削除
12. オフセット・スプラインの削除
13. デバッグ方法

第1章 プログラム開発

1. 作成プログラムの概要

それではバッチ・モードのプログラムを実際に作成してみましょう。

作成するプログラムは指定した図面内の不表示要素を一括削除するものです。

実行プログラム名 : ERSNOSHO.EXE

入力パラメータ : 区画、グループ、ユーザー、図面名を空白で区切って順番に指定します。
 図面名はMC図面名形式のみ対応とし、区切り文字(カンマ)を使用することも省略して20文字の図面名を指定することも可能とします。

コマンド入力例 : >ersnosho c cad train practice1,test

>ersnosho c cad train "practice1 test" 空白を含む場合、二重引用符で囲みます

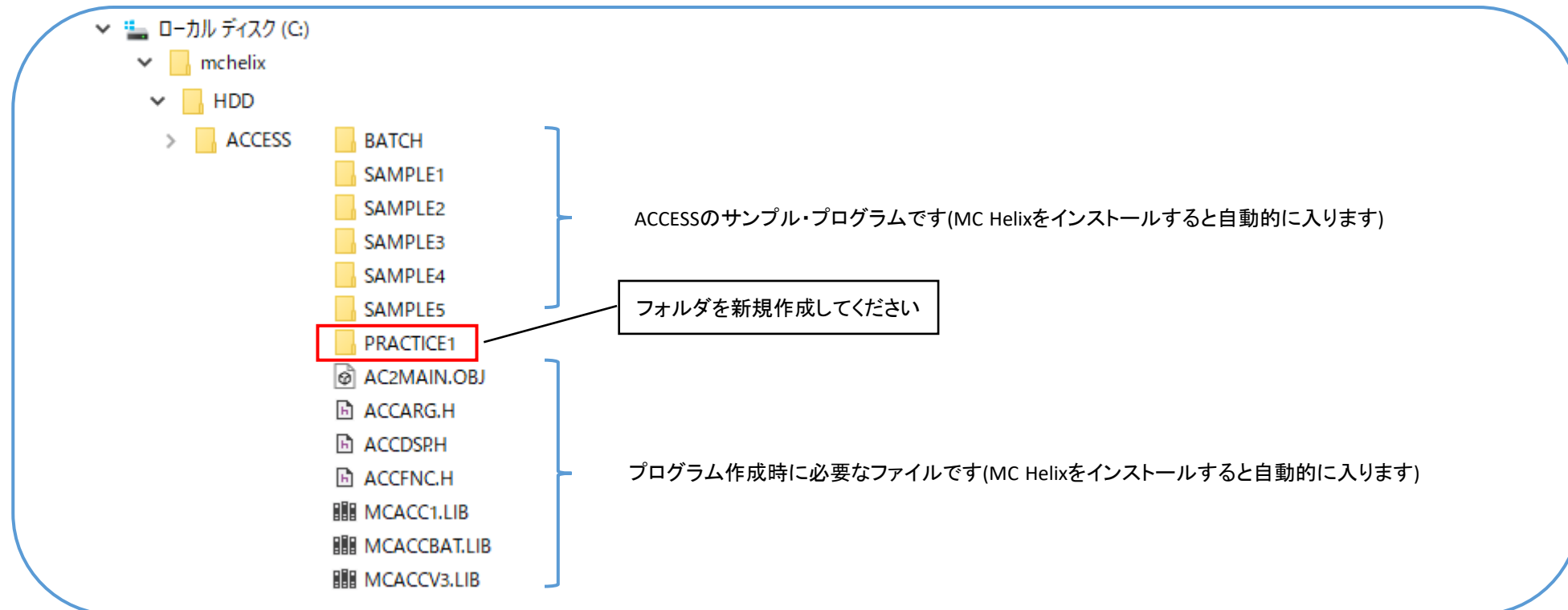
作成するプログラムは2022 R1からチュートリアルダウンロード用サンプル ③としてご提供している「不表示要素・点の一括削除(バッチ形式)」を元に簡略化したものです。
 サンプルの動作をご確認いただくことで作成するプログラムの動作イメージがわくと思います。
 一度ダウンロードしてお試してください。
 『チュートリアル お役立ち情報 アクセス用サンプルプログラム』

2. プログラム作成用フォルダの準備

作業に必要なフォルダを作成します。

ソースファイル、コンパイル・リンクに必要なバッチファイル、作成した実行モジュールなどを格納するためのフォルダが必要です。

特に作成場所の規定はありませんが、今回はACCESSの関数群を呼び出して使用するために必要なファイルなどがインストールされているc:\mchelix\HDD\ACCESSの下にPRACTICE1という名前で作成してください。



第2章 プログラムのコーディングと実行

1. 作成プログラムの全コード紹介

作成プログラムの全コードは[こちら](#)です。

処理の流れは次の通りです。

1. 初期化处理
2. 入力パラメータの取得とチェック
3. 図面のオープン
4. ビューの数の取得
5. ビュー内の不表示要素の検索・削除
ビューの数だけ繰り返す
6. 子図の数の取得
7. 子図内の不表示要素の検索・削除
子図の数だけ繰り返す
8. 図面の保存
9. 終了処理

ソースファイル内での対応箇所

1	
2	
3	B
4	
5	A
6	
7	A
8	B
9	

この後、コードを少しずつ組み込みながらコンパイル・リンクして、その都度動作を確認しながら進めます。

2. メインプログラム(初期化・終了処理)作成(1/3) **1** **9**

本頁より、各コードについて解説します。

1. プログラム作成のために準備したフォルダ(c:\mchelix\HDD\ACCESS\PRACTICE1)に“ERSNOSH0.C”という名前でテキスト・ファイルを作成してください。
2. エディタで“ERSNOSH0.C”を開いてください。ソースコードを記述していきます。
3. MC Helixとの連携を図るためメインに相当する関数の名前は“ac2userp”としてください。

バッチ・モードの場合“main()”で始めることもできますが、その詳細は『プログラマーズ・ガイド』を参照してください。

```
void    ac2userp(void)
{
    return;
}
```

3.メイン関数の名前

4. ACCESSの関数群を使用するためには初期化処理が必要です。まず最初に“MC_bubegn()”関数を呼び出してください。

関数群の引数の意味や仕様については『ACCESS 関数解説書』を参照しながらコーディングしてください。

オプションはバッチ・モードなので“0”を指定します。

バージョンはMC Helixのバージョンとして“52”を指定します。

```
iflgar[0] = 52L;    ...バージョン
iflgar[1] = 0L;    ...入力変数のチェック
iflgar[2] = 2L;    ...精度

ret = MC_bubegn(0L, iflgar, ioutar, 10L);
if (ret != 0) return;
```

4.オプション

4.エラー発生個所識別番号

入力変数のチェックは"0"を指定します。

精度は倍精度として"2"を指定します。

エラー発生個所識別番号は、「MC_bu」で始まる名前関数の多くにあります。同じ関数を何回も呼び出している場合などにどの関数呼び出しでエラーが発生したか特定するために利用できます。

今回は"10"を指定します。(値は任意の整数値ですが、各関数毎にユニークな番号を指定するとエラー箇所の特定が容易になります)

戻り値が0以外の場合、初期化に失敗したということですので処理を終了します。

5. プログラムを終了する時には最後に必ず終了処理を行う

必要があります。

"MC_buend()"関数を呼び出してください。

オプションは"1"を指定してください。

6. ACCESSの関数を使用していますので、その関数宣言が

必要です。

"accfnc.h"というインクルード・ファイルが提供されていますので、必ずインクルードしてください。

```
#include "accfnc.h" — 6.関数宣言

void ac2userp(void)
{
    .
    .
    ret = MC_bubegn(0L, iflgar, ioutar, 10L);
    if (ret != 0) return;
    .
    .
    MC_buend(1L, 22L);
    return; — 5.オプション
}
```

7. 初期化・終了処理はコーディングできましたが、もう1つプログラム実行中にエラーが発生した場合に役立つ関数を呼び出します。

“MC_bumode()”です。

発生したエラー情報や各関数の入出力パラメータ情報を
トレース情報として取得できるようになります。

オプションはモードを設定するので“1”を指定します。

トレース・モードは入出力パラメータとエラー情報を問い合わせられるよう“1”を指定します。

未使用パラメータには“0”を指定しておきます。

精度はMC_bubegnでの指定と同じ倍精度で“12”を指定します。

(MC_bubegnは倍精度“2”でしたので注意してください)

トレース情報は、c:¥MCADAMの下メッセージ・ファイル「ACCMSnnn.XXX」に出力されます。(nnnは000~999)
ただし、トレース・モードをオンにすると、メッセージ・ファイルはかなりの量が出力され、実行速度低下の原因となります。
デバッグのときだけオンにして、デバッグ終了後はオフにして使用することをお勧めします。

8. ソースファイル(ERSNOSHO.C)を保存します。

```
ret = MC_bubegn(0L, iflgar, ioutar, 10L);
```

```
modear[0] = 1L;  ...トレース・モード
```

```
modear[1] = 0L;  ...未使用
```

```
modear[2] = 12L; ...精度
```

```
ret = MC_bumode(1L, modear, 11L);
```

```
if (ret != 0) goto EXIT;
```

```
EXIT:
```

```
MC_buend(1L, 22L);
```

```
return;
```

```
}
```

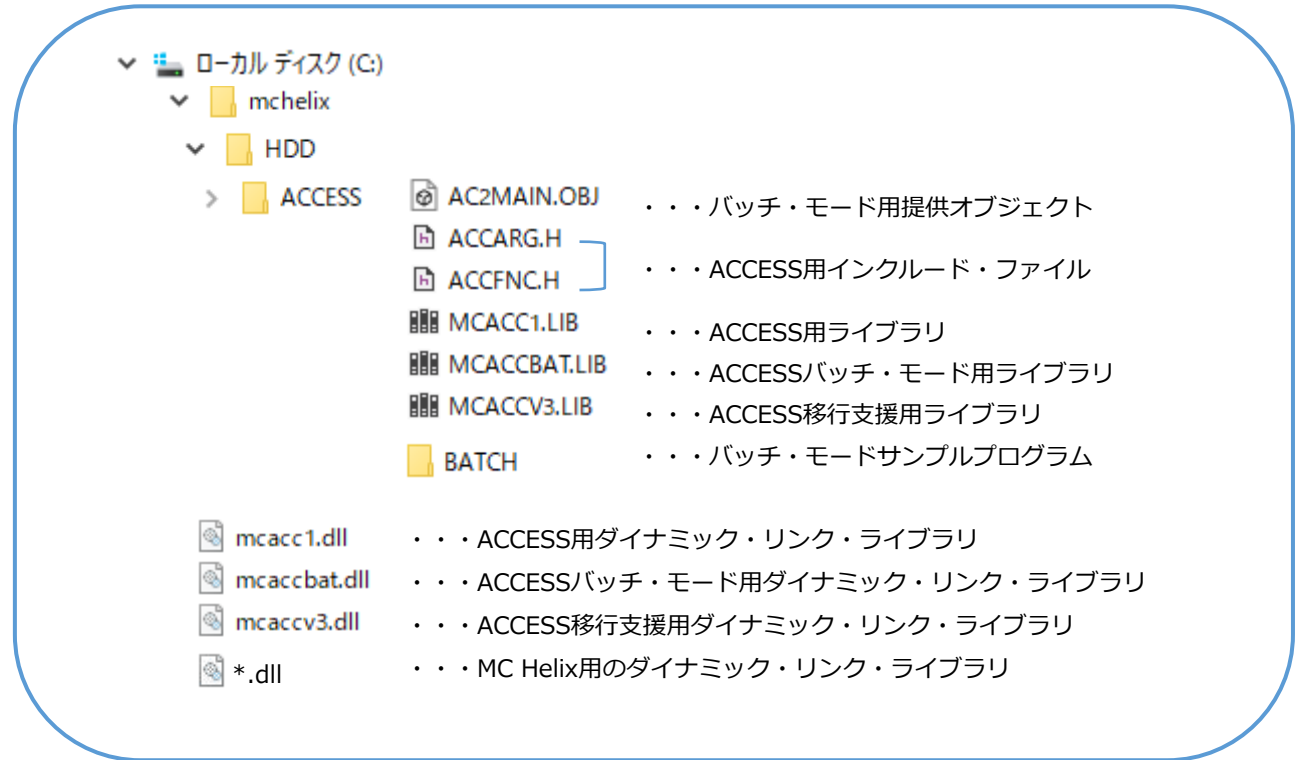
7.オプション

3. コンパイル・リンク方法(1/7)

開始・終了処理がコーディングできたところで一度コンパイル・リンクしてみましょう。

ACCESSプログラムのリンク・コンパイルに必要なファイルはc:\\$mhelix\\$HDDの下に導入されています。

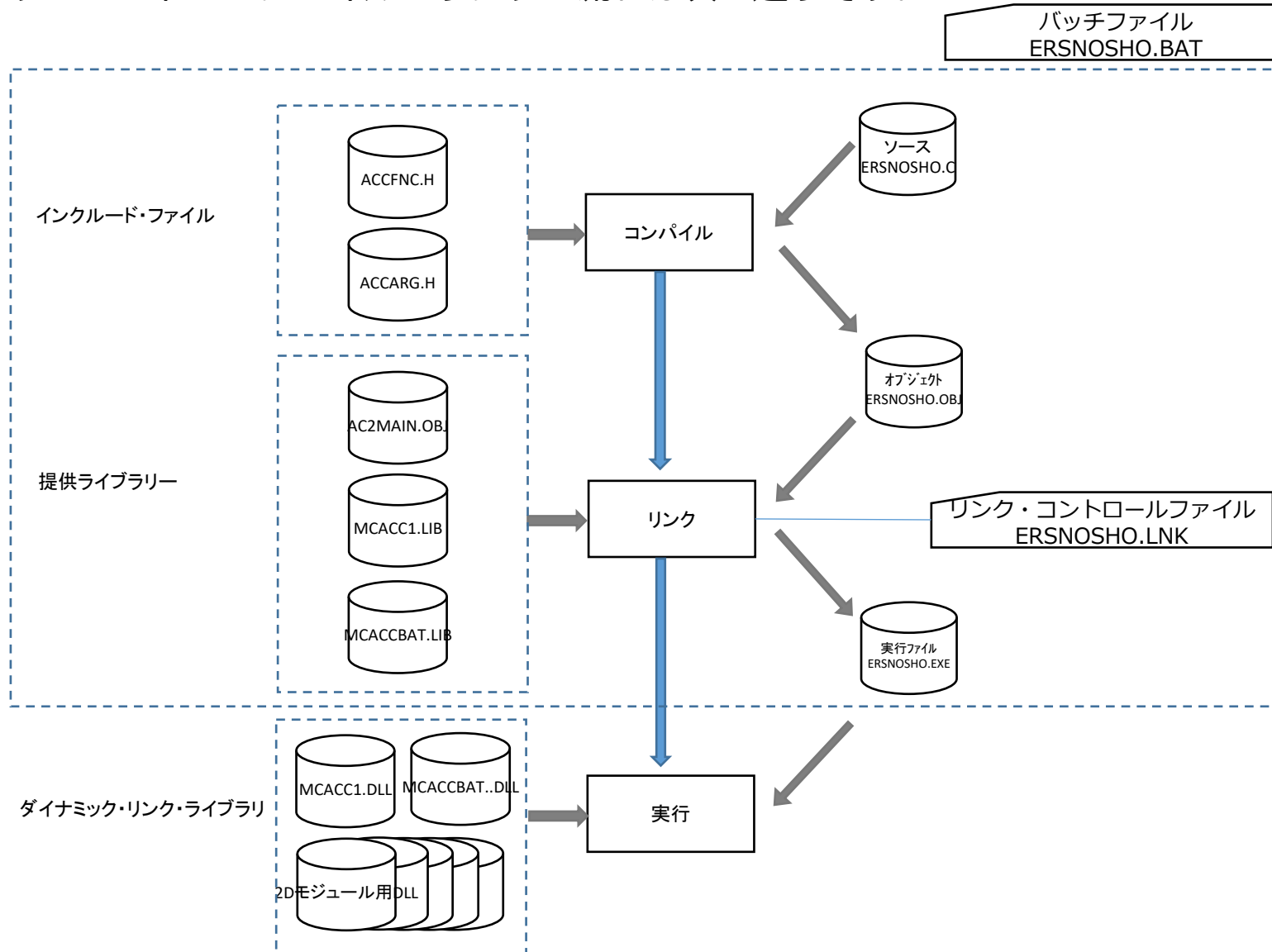
これらのうち必要なファイルを使用してコンパイル・リンクします。



ファイル名	説明
AC2MAIN.OBJ	バッチ・モード用提供オブジェクト
ACCARG.H	ACCESS用インクルード・ファイル
ACCFNC.H	
MCACC1.LIB	ACCESS用ライブラリ
MCACCBAT.LIB	ACCESSバッチ・モード用ライブラリ
MCACCV3.LIB	ACCESS移行支援用ライブラリ
BATCH	バッチ・モードサンプルプログラム
mcacc1.dll	ACCESS用ダイナミック・リンク・ライブラリ
mcaccbat.dll	ACCESSバッチ・モード用ダイナミック・リンク・ライブラリ
mcaccv3.dll	ACCESS移行支援用ダイナミック・リンク・ライブラリ
*.dll	MC Helix用のダイナミック・リンク・ライブラリ

コンパイル・リンク方法(2/7)

バッチ・モードのコンパイル・リンクの流れは次の通りです。



コンパイル・リンクはコマンドプロンプトからバッチファイルを実行することで行います。
まず、バッチファイルとリンク・コントロールファイルを準備します。

これ以降、64bit版モジュールを作成する方法を解説します。

1. サンプルとして提供されているバッチ・モードプログラム作成用のバッチファイルとリンク・コントロールファイルをソースファイルと同じフォルダにコピーします。

c:¥mhelix¥HDD¥ACCESS¥BATCHの下の

PROG.BAT	——	バッチファイル
PROG.LNK	——	リンク・コントロールファイル

をc:¥mhelix¥HDD¥ACCESS¥PRACTICE1にコピーします。

ファイル名をプログラム名に合わせてERSNOSHO.BATとERSNOSHO.LNKにリネームします。

拡張子「.lnk」のファイルはショートカットファイルとみなされ、エクスプローラーでは拡張子が表示されません。
編集のためファイルを開く場合、ファイルをエディタにドラッグ&ドロップしてください。
リネームする場合、拡張子以外の部分(表示されている部分)のみを変更してください。

2. バッチファイル内のプログラム名(PROG)を作成プログラム名に合わせて“ERSNOSHO”に変更します。

Visual Studio のビルド環境を準備するバッチファイルの呼び出しを追加します。

“strcpy”関数など低いセキュリティレベルの古い関数に対する警告を表示しないため

“/D_CRT_SECURE_NO_WARNINGS”を付加します。(任意です)

ERSNOSHO.BAT

```
@ECHO OFF
```

```
CALL "C:¥Program Files (x86)¥Microsoft Visual Studio 14.0¥VC¥vcvarsall.bat" amd64
```

Visual Studio のビルド環境を準備するバッチファイル呼び出しバージョンやインストール時の指定により場所は異なります。64ビットモジュールを作成するためのオプションを指定して呼び出します。

```
SET ORG_INCLUDE=%INCLUDE%  
SET ORG_LIB=%LIB%
```

```
SET ACCESS_PATH=C:¥MCHelix¥HDD¥ACCESS
```

ACCESS用ファイルの導入されている場所を設定

```
SET LIB=%ACCESS_PATH%;%LIB%;  
SET INCLUDE=%ACCESS_PATH%;%INCLUDE%;
```

ライブラリとインクルード・ファイルの参照場所にACCESS用ファイルの場所を追加

```
cl /c /W3 /J /MD /DWIN32 /D_NTSDK /D_CRT_SECURE_NO_WARNINGS ERSNOSHO.C
```

コンパイル

```
LINK @ERSNOSHO.LNK
```

コントロールファイルを使用するリンク

```
MT -MANIFEST ERSNOSHO.EXE.MANIFEST -OUTPUTRESOURCE:ERSNOSHO.EXE;1
```

マニフェスト埋め込み

```
SET INCLUDE=%ORG_INCLUDE%  
SET LIB=%ORG_LIB%  
SET ORG_INCLUDE=  
SET ORG_LIB=  
SET ACCESS_PATH=
```

3. リンク・コントロールファイル内のプログラム名(PROG)を作成プログラムに合わせて“ERSNOSHO”に変更します。

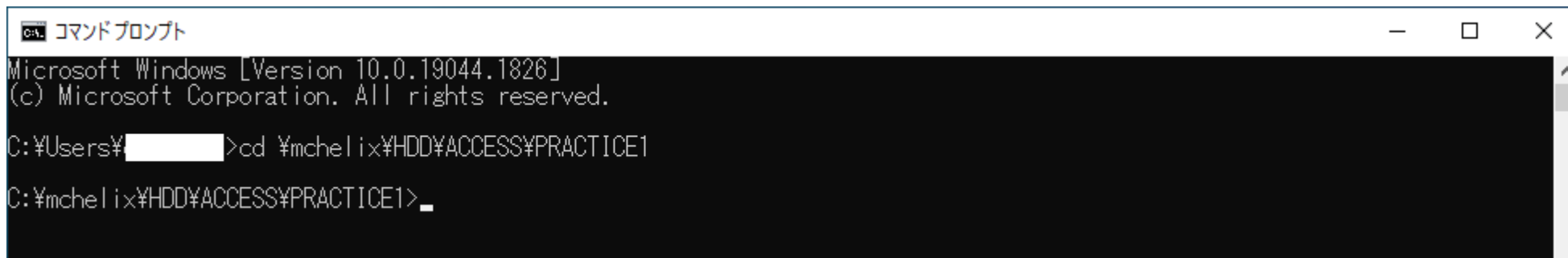
ERSNOSHO.LNK

```
/SUBSYSTEM:console /ENTRY:mainCRTStartup  
/manifest  
/OUT:ERSNOSHO.EXE ...実行ファイル名  
/MAP:ERSNOSHO.MAP ...リスト・ファイル名  
AC2MAIN.OBJ ...提供オブジェクト・ファイル  
ERSNOSHO.OBJ ...オブジェクト・ファイル  
MCACC1.LIB ...ライブラリ名  
MCACCBAT.LIB ...バッチ用ライブラリ名  
MSVCRT.LIB OLDNAMES.LIB KERNEL32.LIB GDI32.LIB USER32.LIB  
COMDLG32.LIB WSOCK32.LIB
```


4. コマンド・プロンプトを起動します。

5. ソースファイルがあるフォルダに移動します。

コマンドプロンプトで"cd %mchelix%HDD%ACCESS%PRACTICE1"と入力して[Enter]



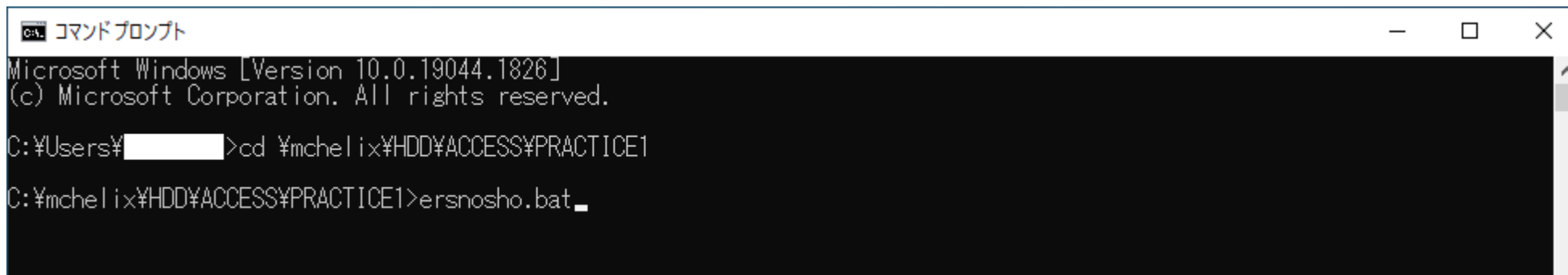
```
コマンドプロンプト
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\%>cd %mchelix%HDD%ACCESS%PRACTICE1

C:\mchelix%HDD%ACCESS%PRACTICE1>_
```

6. バッチファイルを実行します。

コマンドプロンプトで"ersnosho.bat"と入力して[Enter]



```
コマンドプロンプト
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\%>cd %mchelix%HDD%ACCESS%PRACTICE1

C:\mchelix%HDD%ACCESS%PRACTICE1>ersnosho.bat_
```

7. コンパイル・リンクが実行されます。

エラーが発生した場合はコーディングを修正し、再度コンパイル・リンクを行います。

ソースファイルと同じフォルダに“ERSNOSHO.EXE”が作成されていることを確認してください。

(ここでは正常にコンパイル・リンクできることを確認するだけでモジュールの実行は行いません。)

The screenshot shows a Windows Command Prompt window on the left and a File Explorer window on the right. The Command Prompt displays the following text:

```
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\%[redacted]>cd %mhelix%HDD%ACCESS%PRACTICE1

C:\mhelix%HDD%ACCESS%PRACTICE1>ersnosho.bat
Microsoft(R) C/C++ Optimizing Compiler Version 19.00.24215.1 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

ERSNOSHO.C
Microsoft (R) Incremental Linker Version 14.00.24215.1
Copyright (C) Microsoft Corporation. All rights reserved.

/SUBSYSTEM:console /ENTRY:mainCRTStartup
/manifest
/OUT:ERSNOSHO.EXE
/MAP:ERSNOSHO.MAP
AC2MAIN.OBJ
ERSNOSHO.OBJ
MCACC1.LIB
MCACCBAT.LIB
MSVCRT.LIB OLDNAMES.LIB KERNEL32.LIB GDI32.LIB USER32.LIB
COMDLG32.LIB WSOCK32.LIB
Microsoft (R) Manifest Tool version 10.0.10011.16384
Copyright (c) Microsoft Corporation 2012.
All rights reserved.

C:\mhelix%HDD%ACCESS%PRACTICE1>
```

The File Explorer window shows the directory structure: mhelix > HDD > ACCESS > PRACTICE1. The file list is as follows:

名前	更新日時	種類	サイズ
ERSNOSHO.EXE	2022/06/14 16:45	アプリケーション	17 KB
ERSNOSHO.EXE.manifest	2022/06/14 16:45	MANIFEST ファイル	1 KB
ERSNOSHO.MAP	2022/06/14 16:45	Linker Address Map	39 KB
ERSNOSHO.obj	2022/06/14 16:45	3D Object	11 KB
ERSNOSHO.BAT	2022/06/14 16:45	Windows バッチファ...	1 KB
ERSNOSHO	2022/06/14 16:44	ショートカット	1 KB
ERSNOSHO.pdb	2022/05/12 9:18	Program Debug D...	484 KB
ERSNOSHO.C	2022/04/27 13:42	C Source	15 KB

4. 入力パラメータの取得 2

インクルード・ファイル「ACCARG.H」を利用することでプログラム起動時に入力パラメータとして渡された複数のデータを受け取ることができます。データ参照に必要な変数がファイル内で構造体として定義されています。

- ・ ACCARG argc . . . 入力したデータの個数
 - ・ ACCARG argv . . . 各データの文字列のアドレス
1. ソースファイルをエディタで開いて"accarg.h"をインクルードします。
 2. パラメータを確認するためACCARG argcの個数分のACCARG argvをプリント文で出力します。
 3. ソースファイルを保存して、コマンドプロンプトでバッチファイルを実行してコンパイル・リンクします。
 4. 実行用のコマンドプロンプトを起動します。
 5. 実行モジュールのあるフォルダに移動してコマンドを入力します。>ersnosho c cad train practice1,test
入力パラメータが正しく出力されていることを確認してください。

```
#include "accfnc.h"
#include "accarg.h"

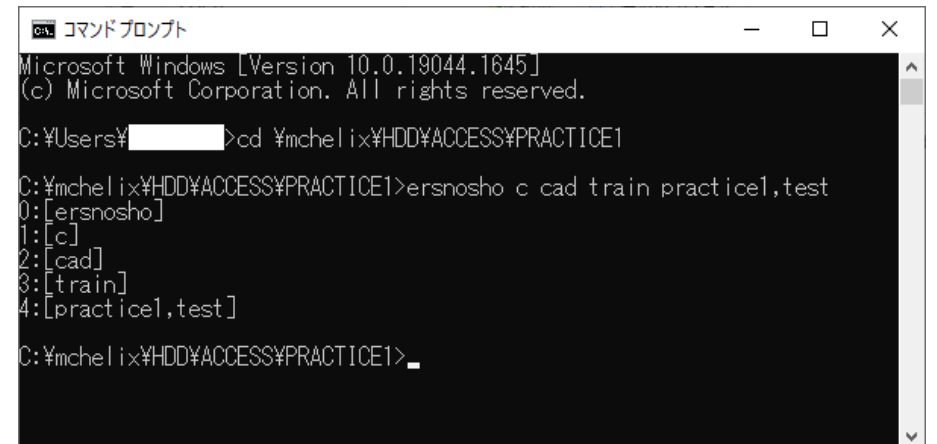
:
:
ret = MC_bubegn(0L, iflgar, ioutar, 10L);

:
:
ret = MC_bumode(1L, modear, 11L);

for (ii = 0; ii < ACCARG argc; ii++) {
    printf( "%ld:[%s]¥n", ii, ACCARG argv[ii] );
}
```

1.インクルードする

2.個数分のデータが格納されている



```
コマンドプロンプト
Microsoft Windows [Version 10.0.19044.1645]
(c) Microsoft Corporation. All rights reserved.

C:¥Users¥>cd ¥mhelix¥HDD¥ACCESS¥PRACTICE1
C:¥mhelix¥HDD¥ACCESS¥PRACTICE1>ersnosho c cad train practice1,test
0:[ersnosho]
1:[c]
2:[cad]
3:[train]
4:[practice1,test]

C:¥mhelix¥HDD¥ACCESS¥PRACTICE1>
```

5. 入力パラメータのチェック 2

入力パラメータのチェックを行います。

図面名がカンマで区切られた分割形式で入力されている場合
カンマを取り除き20桁の連続形式に変換します。

1. 区画名、グループ名、ユーザー名、図面名を格納する変数を宣言するため配列の長さを"#define"します。
2. 区画名、グループ名、ユーザー名、図面名を格納する変数を宣言して初期化します。
3. 入力パラメータの個数をチェックし、少ない場合はパラメータの入力例を表示して終了します。
4. 区画名、グループ名、ユーザー名、図面名を順番に変数に格納します。
5. 図面名にカンマが含まれるかチェックし、含まれる場合は"MC_budwgn"により連続形式に変換します。
ACCESS関数の引数では図面名は連続形式で扱うことが多いからです。
6. コンパイル・リンクしてエラーがないことを確認します。

```

for (ii = 1; ii < ACCARG. argc; ii++) {
    len = (int)strlen(ACCARG. argv[ii]);

    switch( ii ) {
        :
        case 4:
            pos = strchr(ACCARG. argv[ii], ',');
            len = (int)strlen( ACCARG. argv[ii] );
            if (pos == NULL) {
                if( len < 1 || len > DWGID_LEN ) {
                    printf( "Error Invalid Drawing Name\n" );
                    goto EXIT;
                }
                memcpy( dwgid, ACCARG. argv[ii], len);
            }
            else {
                if( len < 1 || len > DWG_LEN ) {
                    printf( "Error Invalid Drawing Name\n" );
                    goto EXIT;
                }
                ret = MC_budwgn( 1L, group, user, dwgid, &len,
                               ACCARG. argv[ii], 12L );

                if (ret != 0) {
                    printf( "Error Invalid Drawing Name\n" );
                    goto EXIT;
                }
            }
            break;
        }
    }
}

```

4. パラメータを順番に変数に格納

4. 図面名を格納する処理

5. 図面名のカンマチェック

5. カンマなしなのでそのまま格納

5. 20桁の連続形式に変換して格納

6. 図面のオープン・ファイル処理 3 8 B

入力パラメータで指定された図面をオープン・ファイルします。

- ACCESSでは図面ごとに固有の番号(モデル参照番号)を付け、この番号で制御します。ACCESSでは1～10のモデル参照番号を使って図面を開きます。
今回は"1"を使います。
- モデル参照番号1として開く図面の区画を指定します。
"MC_budriv"を使用します。
- グループ、ユーザー、図面名を指定してモデル参照番号1として図面を開きます。"MC_buopen"を使用します。
既存図面を呼び出すのでオプションは"2"です。
- モデル参照番号1として開いている図面を現在のグループ、ユーザー、図面名のまま上書き保存します。
"MC_bufile"を使用します。
- "MC_buopen", "MC_bufile"はエラーの種類が多いのでエラー表示部分をプログラム内に別定義した関数(以後内部関数とします) "disp_msg"とします。(P.8の「作成プログラムの全コード紹介」を参照ください。)
- コンパイル・リンクしてエラーがないことを確認してください。

```
#define MODEL_NO 1
:
:
ret =MC_budriv(1L, MODEL_NO, vol, 13L);
if (ret != 0) {
    printf( "Error Set Volume¥n" );
    goto EXIT;
}
ret = MC_buopen(2L, MODEL_NO, group, user, dwgid, 14L);
if (ret != 0) {
    disp_msg( ret );
    goto EXIT;
}
:
:
ret = MC_bufile(5L, MODEL_NO, group, user, dwgid, 20L);
if (ret != 0) {
    disp_msg( ret );
    goto EXIT1;
}
```

1.モデル参照番号を宣言

2.区画を設定

3.グループ、ユーザー、図面名(連続形式)を指定して既存図面を開く

4.上書き保存する

5.エラー表示を内部関数で行う

7. 図面の準備と実行

作成プログラムの動作を確認するためにテスト用の図面を作成して、今までコーディングしたプログラムを実行してみます。

1. 任意の区画、グループ、ユーザーに新規図面を作成してください。

ここでは、区画：c、グループ：cad、ユーザー：train、図面名：practice1,test で作成します。

2. そのまま図面を保存します。(図面の保存時刻を確認しておいてください。)
3. 保存時刻から1分以上経ったことを確認し、実行用のコマンドプロンプトから作成したプログラムを実行します。
>ersnosho c cad train practice1,test (1. で作成した図面を指定してください)
4. 2. で保存した図面が内容は変更されず、保存時刻が更新されていることを確認してください。
5. トレース情報のメッセージ・ファイル「ACCMSnnn.XXX」がc:¥MCADAMの下に作成されていることを確認してください。「ACCMSnnn.XXX」のうちnnnの数字が一番大きく、最新のファイルをエディタで開いてください。作成プログラムから呼び出したACCESS関数のパラメータやリターンコードが表示されています。
4. で保存時刻が更新されていない場合はエラーが発生している可能性があります。
ACCESS関数のリターンコードを確認してください。

図面内のビュー・子図の情報を取得します。

1. 現行のビュー情報を取得します。

ACCESSではビュー・子図は番号で制御します。

ビュー番号：正の整数

PV:0 その他のビュー:1~ 全ビュー:1000

子図番号：負の整数

現行ビューの情報を保持するのは要素を検索している

途中などでビュー・子図が変更される可能性があるためです。

処理終了時に元のビューに戻します。

2. ビュー・子図の数情報を取得し、プリント文で出力します。

3. 処理対象としたいビュー・子図を現行に切り替えて情報を取得します。回数分繰り返します。

ビュー名・子図名を取得して、プリント文で出力します。

4. コンパイル・リンクして実行してください。

PVのみの図面で実行してPV名のみが出力されることを

確認してください。図面内にビュー(V1)・子図(#1,#2,#3)を

作成して再度実行し、複数のビュー・子図名が出力される

ことを確認してください。

```

/* Get Number of View */
iopt = 1L;
ret = MC_gvdnum(iopt, MODEL_NO, &ivdtIn);
.
.
printf("View Number:%ld¥n", ivdtIn);

if(ivdtIn == 0L) {
    goto EXIT1;
}

/* Erase Elements in View */
for(ii=0; ii<ivdtIn; ii++)
{
    /* Change Current View */
    iopt = 1L;
    ivudet = ii;
    ret = MC_bucds(iopt, MODEL_NO, numopt, &ivudet, &number, 16L);
    if(ret != 0) {
        disp_msg( 999 );
        goto EXIT2;
    }

    /* Get View Name */
    ret = MC_buview( 1L, MODEL_NO, &ivudet, &VIEW.ivuid, &iparvu,
        (float*)vaxar, vmrxar, 17L );

    printf("View Name :%.2s¥n", VIEW.vuid);
}
    
```

2.ビューの個数を取得

2.ビューの個数を出力

3.ビューの個数分繰り返す

3.処理対象としたいビューに切り替える

3.ビュー情報を取得

3.ビュー名を出力

9. 要素の検索 5 7 A

ビュー・子図内の要素を検索します。

1. 要素の検索はビュー・子図共通で使用できるよう内部関数として作成します。
2. 要素検索のための関数は“MC_bufnd1”、“MC_bufnd2”、“MC_bunext”がありますが、ここでは“MC_bunext”を使用します。指定した要素の次の要素を問い合わせることができるため、繰り返すことでビュー・子図内の全要素を検索できます。要素はポインタという値で制御します。
3. 検索した要素の要素種別番号(要素の種別を区別する固有番号)を取得し、プリント文で出力します。
4. この内部関数を各ビュー・子図名を取得・出力した後に呼び出します。(P.8「作成プログラムの全コード紹介」参照)
5. コンパイル・リンクします。
6. テスト用図面の各ビュー・子図内に点、直線、円などの要素を作成して保存し、プログラムを実行します。
各ビュー・子図内の要素のポインタと要素種別番号が出力されることを確認してください。

```

static short erase_elements()
{
    :
    iopt = 1L; /* Search Next Element */
    mptr = 1L;
    ret = MC_bunext(iopt, MODEL_NO, mptr, &nxtptr, 33L);
    mptr = nxtptr;
    if(ret == 0)
    {
        for(;;)
        {
            iopt = 1L; /* Search Next Element */
            ret = MC_bunext(iopt, MODEL_NO, mptr, &nxtptr, 34L);

            /* Check Element Type */
            itype = MC_biityp( MODEL_NO, mptr );
            if( itype < 0L ) goto NEXT_ELM;

            printf( "mptr = %ld, itype = %ld", mptr, itype );
            printf( "\n" );

            if(ret != 0) break; /* No Rest */
            mptr = nxtptr;
        }
    }
    return(s_ret);
}

```

2. 最初は要素ポインタ“1”で検索

2. 最初の要素ポインタを保持

2. 次の要素を先に検索しておく

3. 要素種別番号を取得

2. 要素の終わりで繰り返しを抜ける

2. 次の検索のためポインタを入れ替える

1 0. 不表示要素の検出 **A**

ビュー・子図内の不表示要素を検出します。

1. 要素検索の内部関数内で検索された要素が表示対象要素かどうか判定します。“MC_bisho”を使用します。
2. 表示処理対象外(不表示)要素の場合、プリント文で“NO-SHO”を付加して出力します。
3. コンパイル・リンクします。
4. テスト用図面の各ビュー・子図内の要素の内、いくつかの要素を不表示にして保存し、プログラムを実行します。
各ビュー・子図内の要素の内、不表示にした要素の要素種別番号の後ろに“NO-SHO”が出力されることを確認してください。

```

static short erase_elements()
{
    .
    .
    for(;;)
    {
        iopt = 1L;      /* Search Next Element */
        ret = MC_bunext(iopt, MODEL_NO, mptr, &nxtptr, 34L);

        /* Check Element Type */
        itype = MC_biityp( MODEL_NO, mptr );
        if( itype < 0L ) goto NEXT_ELM;

        /* Check Sho/No-Sho */
        isho = MC_bisho( MODEL_NO, mptr );
        if( isho != 0L && isho != 1L ) goto NEXT_ELM;

        printf( "mptr = %ld, itype = %ld", mptr, itype );
        if( isho == 0L ) printf( " NO-SHO" );
        printf( "\n" );
    }

    NEXT_ELM:
        if(ret != 0) break; /* No Rest */
        mptr = nxtptr;
    }
    .
    .
}

```

1.要素が表示対象か取得

2.不表示要素の場合、“NO-SHO”と出力

1 1. 要素の削除 **A**

不表示要素を削除します。

1. 表示要素の場合、次の要素を検索する処理に進みます。
2. 不表示要素の場合、“MC_budel”を使用して削除します。
削除した要素のポインターは“MC_bunext”で次の要素を検索する時に指定してもエラーとなります。従って、削除する前に次の要素を検索しておく必要があります。特に、検索しながらの要素削除ではACCESS関数を呼び出す順番に注意してください。
3. コンパイル・リンクします。
4. プログラムを実行して、表示要素のポインターが出力されないことと図面内の不表示要素が削除されていることを確認してください。図面のPVに曲線とそのオフセット・スプラインを作成し、両方を不表示にして保存してください。プログラムを実行し、曲線が削除されていないことを確認してください。
“MC_budel”では削除できない要素がいくつかありますが、オフセット・スプラインを持つ親曲線もその1つです。
次はオフセット・スプライン、親曲線両方を削除できるようにしてみましょう。

```
for(;;)
{
    iopt = 1L; /* Search Next Element */
    ret = MC_bunext(iopt, MODEL_NO, mptr, &nxtptr, 34L);

    /* Check Element Type */
    itype = MC_biityp( MODEL_NO, mptr );
    if( itype < 0L ) goto NEXT_ELM;

    /* Check Sho/No-Sho */
    isho = MC_bisho( MODEL_NO, mptr );
    if( isho != 0L && isho != 1L ) goto NEXT_ELM;
    if( isho == 1L ) goto NEXT_ELM;

    printf( "mptr = %ld, itype = %ld", mptr, itype );
    if( isho == 0L ) printf( " NO-SHO" );
    printf( "\n" );

    iopt = 1L; /* Erase Element */
    mptrar[0] = mptr;
    MC_budel(iopt, MODEL_NO, mptrar, 35L);

    NEXT_ELM:
    if( ret != 0 ) break; /* No Rest */
    mptr = nxtptr;
}


```

2.削除される前に次の要素を検索しておく

1.表示要素の場合、次の要素検索に進む

2.不表示要素の場合、削除する

2.削除した要素のポインターと既に検索しておいた次の要素のポインターを入れ替える

1 3. デバッグ方法(1/4)

プログラムのデバッグには「Microsoft Visual Studio」を使用します。

デバッグするためにはデバッグ用のオプションを指定して、コンパイル・リンクする必要があります。

1. コンパイル時のオプションに「 /Od /Zi」を追加します。

プログラム作成用バッチファイルのコンパイル行に追加します。

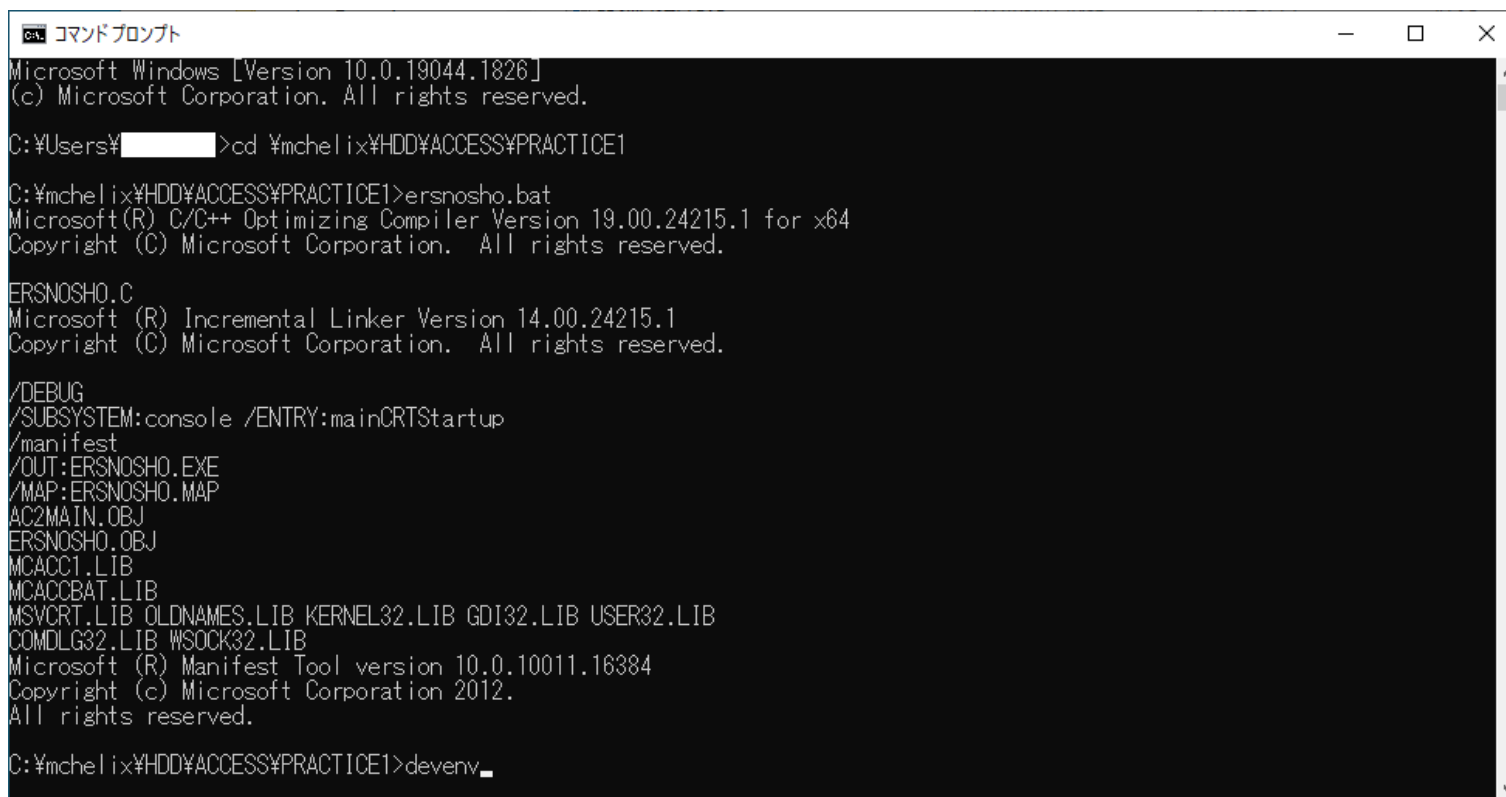
```
·  
·  
SET LIB=%ACCESS_PATH%;%LIB%;  
SET INCLUDE=%ACCESS_PATH%;%INCLUDE%;  
  
cl /c /W3 /J /MD /Od /Zi /DWIN32 /D_NTSDK /D_CRT_SECURE_NO_WARNINGS ERSNOSHO.C  
  
LINK @ERSNOSHO.LNK  
·  
·
```

2. リンク・コントロールファイルの先頭に「/DEBUG」を追加します。

```
/DEBUG  
/SUBSYSTEM:console /ENTRY:mainCRTStartup  
/manifest  
·  
·
```

3. コンパイル・リンク用のコマンドプロンプトからオプションを追加したバッチファイルを実行します。
4. 同じコマンドプロンプトから「Microsoft Visual Studio」を起動します。

コマンドプロンプトで"devenv"と入力して[Enter]



```
コマンドプロンプト
Microsoft Windows [Version 10.0.19044.1826]
(c) Microsoft Corporation. All rights reserved.

C:¥Users¥[redacted] >cd ¥mchelix¥HDD¥ACCESS¥PRACTICE1

C:¥mchelix¥HDD¥ACCESS¥PRACTICE1>ersnosho.bat
Microsoft(R) C/C++ Optimizing Compiler Version 19.00.24215.1 for x64
Copyright (C) Microsoft Corporation. All rights reserved.

ERSNOSHO.C
Microsoft (R) Incremental Linker Version 14.00.24215.1
Copyright (C) Microsoft Corporation. All rights reserved.

/DEBUG
/SUBSYSTEM:console /ENTRY:mainCRTStartup
/manifest
/OUT:ERSNOSHO.EXE
/MAP:ERSNOSHO.MAP
AC2MAIN.OBJ
ERSNOSHO.OBJ
MCACCT.LIB
MCACCBAT.LIB
MSVCRT.LIB OLDNAMES.LIB KERNEL32.LIB GDI32.LIB USER32.LIB
COMDLG32.LIB WSOCK32.LIB
Microsoft (R) Manifest Tool version 10.0.10011.16384
Copyright (c) Microsoft Corporation 2012.
All rights reserved.

C:¥mchelix¥HDD¥ACCESS¥PRACTICE1>devenv_
```

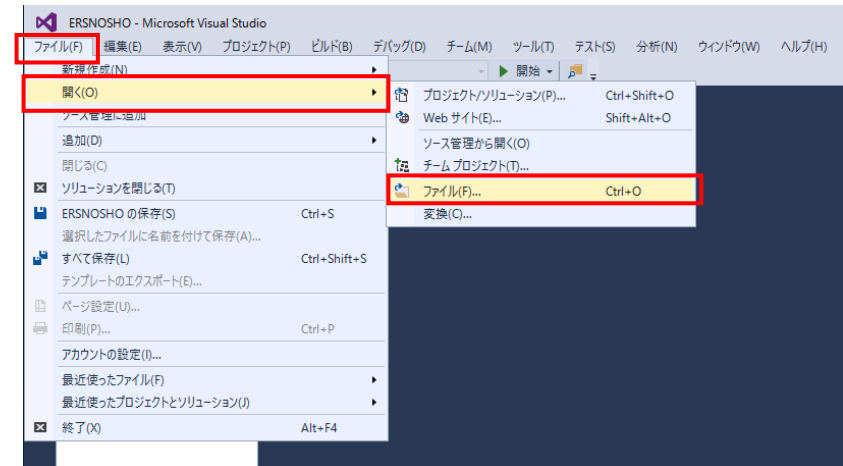
5. 「Microsoft Visual Studio」 が起動します。

メニュー[ファイル]の[開く]-[プロジェクト/ソリューション]を
選択します。



6. 「プロジェクトを開く」ダイアログが表示されるので、
「c:¥mhelix¥HDD¥ACCESS¥PRACTICE1」から
実行プログラム「ERSNOSHO.EXE」を選択します。

7. メニュー[ファイル]の[開く]-[ファイル]を選択します。



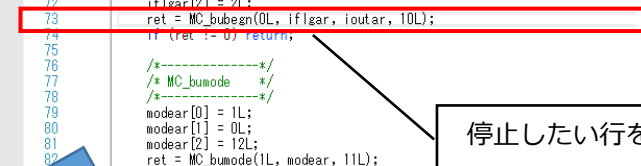
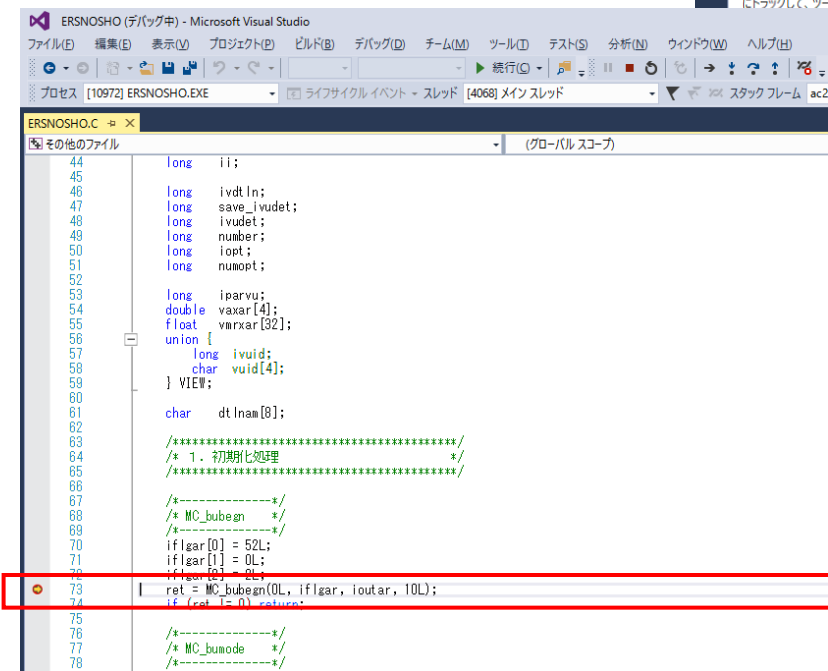
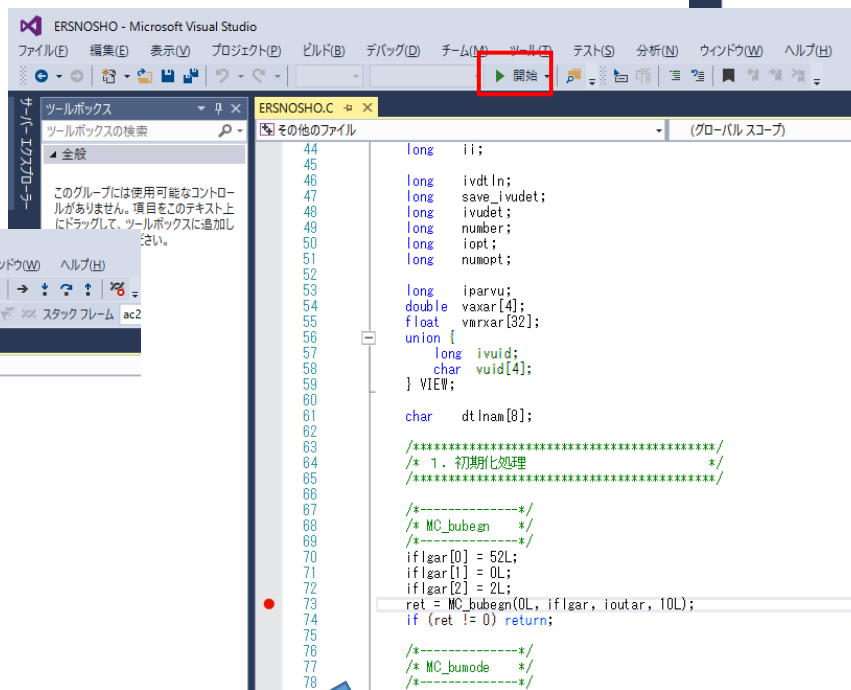
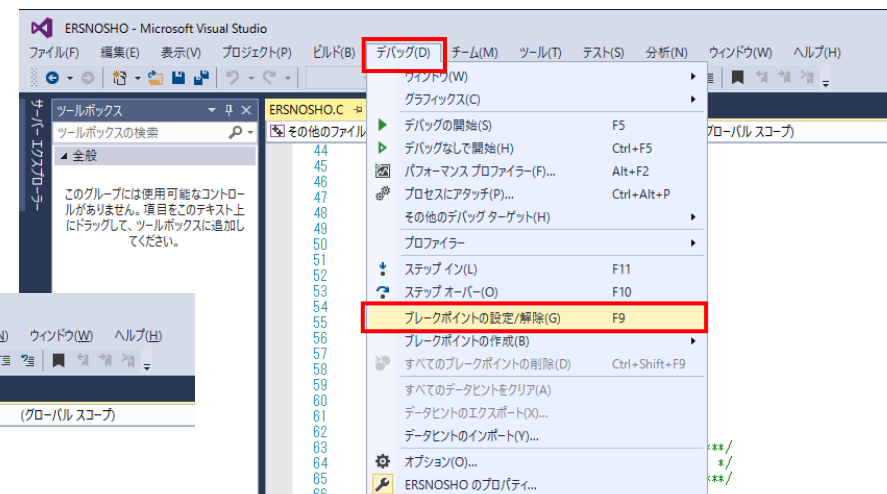
8. 「ファイルを開く」ダイアログが表示されるので、
「c:¥mhelix¥HDD¥ACCESS¥PRACTICE1」から
ソースファイル「ERSNOSHO.C」を選択します。

ソースコードが表示されます。

コマンドライン引数はプロジェクトのプロパティウィンドウの[引数]で指定してください。
(プロパティウィンドウはソリューションエクスプローラーでプロジェクト選択後、右クリックで表示される
ポップアップメニューで[プロパティ]をクリックすることで表示されます。)

9. デバッグで停止したい行を選択して、メニュー[デバッグ]の [ブレークポイントの設定/解除]または[F9]でブレークポイントを設定します。

10. メニュー[デバッグ]の[デバッグ開始]またはツール・バーの [開始]で実行します。



ブレークポイントで停止しますのでデバッグしてください。



※当資料内の文章・画像・商標等（以下、「データ」）に関する著作権とその他の権利は、弊社または原作者、その他の権利者のものです。企業等が非営利目的で使用する場合、個人的な使用を目的とする場合、その他著作権法により認められている場合を除き、データは弊社、原作者、その他の権利者の許諾なく使用することはできません。

※データ等のご利用またはご利用できなかったことによって生じた損害については、弊社は一切の責任を負わないものとし、いかなる損害も補償をいたしません。

※掲載されている内容は2022年8月時点のものです。内容は、事前の予告なしに変更することがあります。

MICRO CADAM、MICRO CADAM Helix は、株式会社CAD SOLUTIONSの商標です。
他の会社名、製品名およびサービス名等はそれぞれ各社の商標です。