

MICRO CADAM Helix 実践操作解説書

ACCESSプログラム開発ガイド 対話プログラム編 2

～ 既存プログラムを変更する ～

2022年8月

株式会社CAD SOLUTIONS

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 リソースファイルの作成

1. メッセージ構造
2. 作成プログラムのメッセージ仕様
3. メニュー・メッセージ作成支援ツールによるメッセージの作成
4. メニュー構造
5. 作成プログラムのメニュー仕様
6. メニュー・メッセージ作成支援ツールによるメニューの作成

第3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. コンパイル・リンク方法
3. メニューの設定
4. 対話操作の制御
5. ファンクションへの登録と実行
6. グループ化範囲の指定と矩形作成
7. グループ化処理
8. 処理の繰り返し
9. CSV出力の対象要素種別をメニューから取得
10. 追加対象要素の情報取得とCSV出力
11. グループ化範囲指定時の矩形ラバーバンド表示

第1章 プログラム開発

1. 作成プログラムの概要

プログラム作成2で作成したグループ化されている点の座標をCSVファイルに出力するプログラムを少し改修してみましょう。

改修部分は次の通りです。

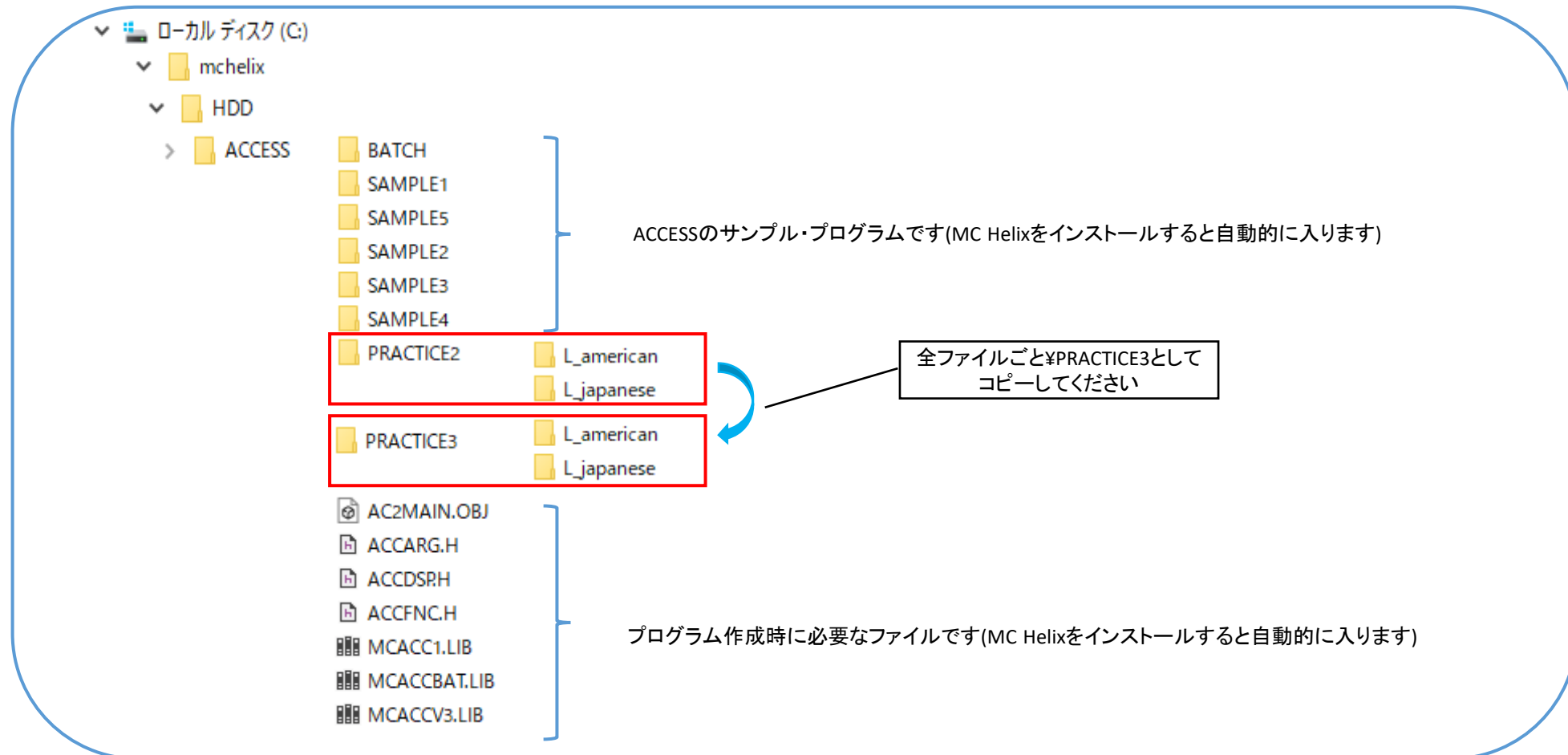
- ・ 2Dモジュールでグループ化した要素を対象としていたが、ACCESSプログラムで範囲を指定して要素をグループ化する
- ・ 処理を繰り返し行えるようにする
- ・ 出力対象に直線・円を追加し、メニューで指定できるようにする
- ・ グループ化の範囲指定時に矩形のラバーバンド表示を行う
- ・ 実行は<ACCESS>の【選択リスト】からではなく、ファンクションに登録して行う

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。

プログラム作成 2 で作成した¥PRACTICE2を¥PRACTICE3としてコピーしてください。

プログラム作成 3 ではプログラム作成 2 で作成したファイルを使用してそれに追加、修正する形で進めていきます。



第2章 リソースファイルの作成

1. メッセージ構造

作成プログラムでは画面上部のメッセージ表示領域に操作を指示するメッセージを表示します。

ACCESSのメッセージは3行分表示できます。

各行は書式の設定により、1行を何文字ずつのフィールドに分割するか指定できます。



- 書式1... 7つのフィールドで、6、10、28、16、16、12、12バイトずつ
- 書式2... 3つのフィールドで、18、63、19バイトずつ
- 書式3... 1つのフィールドで、100バイト
- 書式4... 3つのフィールドで、45、4、51バイトずつ
- 書式5... 2つのフィールドで、50、50バイトずつ
- 書式6... 2つのフィールドで、75、25バイトずつ
- 書式7... 3つのフィールドで、30、40、30バイトずつ
- 書式8... 7つのフィールドで、10、13、11、13、20、14、19バイトずつ
- 書式10... 4つのフィールドで、24、25、25、26バイトずつ
- 書式11... 3つのフィールドで、28、43、29バイトずつ
- 書式12... 3つのフィールドで、33、33、34バイトずつ

MC_msgcust関数を使用すると2Dモジュールの1行目の標準メッセージ(現在のウィンドウ・サイズ、選択要素の情報などが表示)を表示したまま、その下にACCESSのメッセージを表示することもできます。

メッセージ領域に表示するメッセージも「メニュー・メッセージ作成支援ツール」を使用して作成します。

2. 作成プログラムのメッセージ仕様

作成プログラムに追加するメッセージは次の通りです。

要素をグループ化するための範囲指定を行う時に操作指示のメッセージとして表示します。

メッセージ(日本語)	メッセージ(英語)	表示領域	種類	書式
「対角の第1点を指示」	"Indicate Corner"	2行目	操作指示	2-2(63)
「対角の第2点を指示」	"Indicate Opposite Corner"	2行目	操作指示	2-2(63)

既存のブロックに追加定義します。(今回は以下の名称で追加定義します)

ブロック名 : BLK_MSG_SP8 既存

ブロック識別子 : INCL_BLK_MSG_SP8 既存

メッセージ名 : MSG_IND1

MSG_IND2

2Dモジュールの各行には主に次のようなメッセージを表示しています。

ACCESSで作成するプログラムのメッセージ表示でも「操作指示」などの表示行を2Dモジュールと揃えることで統一感を持たせることができます。

また、1行目は2Dモジュールのメッセージをそのまま使用することもできます。

1行目・・・ファンクション名、ビュー名、スケール、ウィンドウ設定など

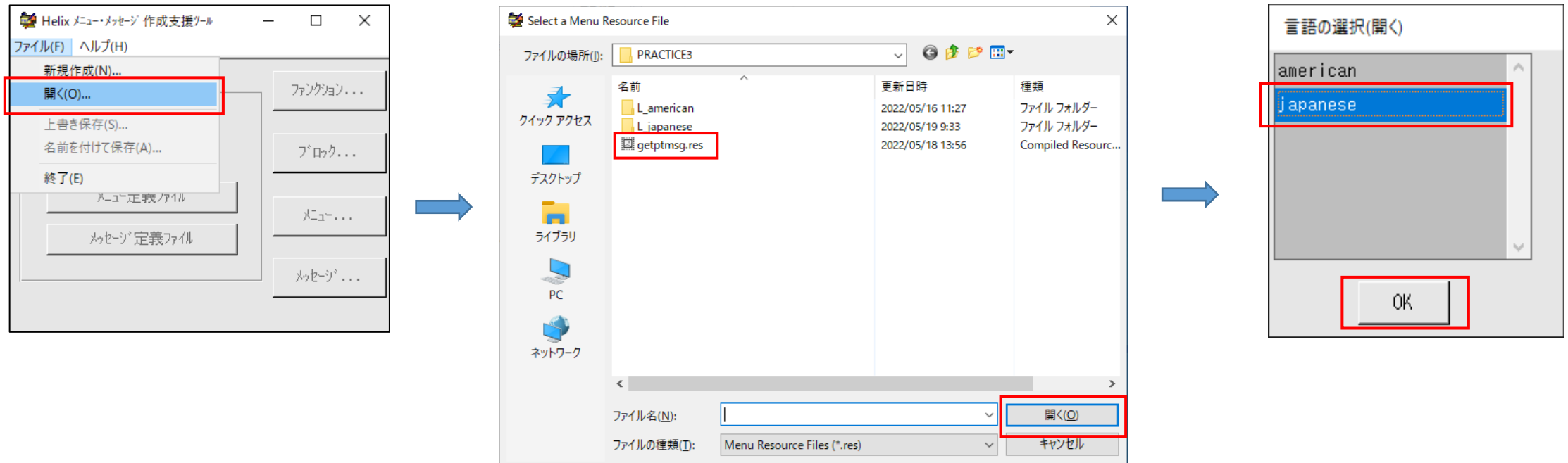
2行目・・・操作指示、モード情報など

3行目・・・エラー情報、各種の座標情報など

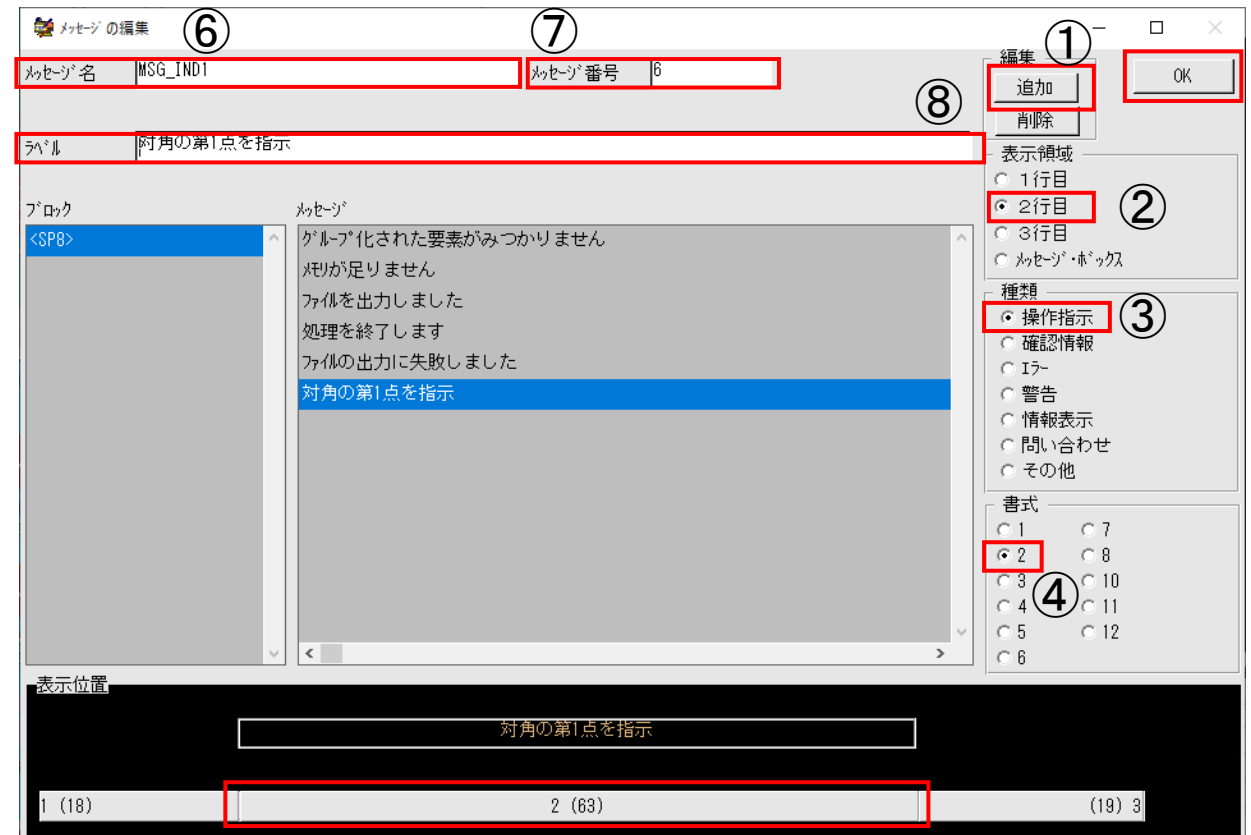
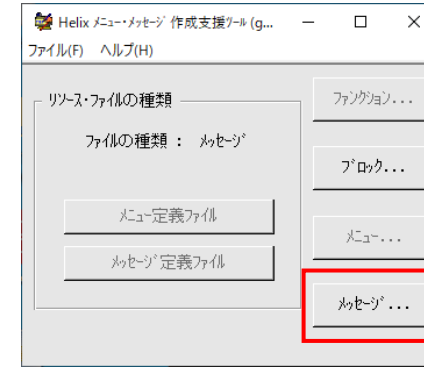
3. メニュー・メッセージ作成支援ツールによるメッセージの作成(1/3)

「メニュー・メッセージ作成支援ツール」を使用して、既存のメッセージ定義ファイル
¥PRACTICE3¥getptmsg.resに追加定義します。

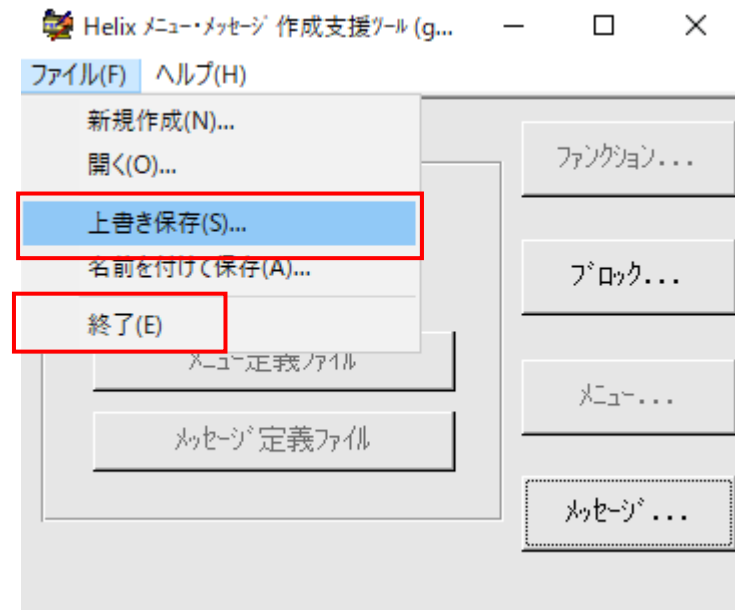
1. 「メニュー・メッセージ作成支援ツール」を起動します。
2. メニュー・バー[ファイル]から[開く]を選択します。
3. ¥PRACTICE3¥getptmsg.resを選択して[開く]を押します。
4. 「言語の選択」画面で[japanese]を選択して[OK]を押します。



5. [メッセージ]を押します。
6. 「メッセージの編集」画面でメッセージを追加していきます。
 - ①一番下のメッセージが選択された状態で[追加]を押す
 - ②表示領域から「2行目」を選択する
 - ③種類から「操作指示」を選択する
 - ④書式から「2」を選択する
 - ⑤表示位置として「2 (6 3)」の部分を選択する
 - ⑥メッセージ名「MSG_IND1」を入力して [Enter]キーを押す
 - ⑦メッセージ番号を指定する(入力値のまま)
 - ⑧ラベルとして「対角の第1点を指示」を入力して[Enter]キーを押す
7. 次のメッセージ(P.9参照)を定義するため ①から⑧を繰り返します。
8. [OK]を押します。



9. 起動直後の画面に戻りますので、メニュー・バー[ファイル]から[上書き保存]を選択します。
10. メニュー・バー[ファイル]から[終了]を選択して終了します。
11. 英語環境の文字情報ファイルにメッセージを追加します。
 - ① ¥L_japaneseのgetptmsg.txtをエディタで開いて登録したメッセージが追加されているか確認してください。
 - ② ¥L_americanのgetptmsg.txtをエディタで開いてください。
 - ③ ¥L_japaneseのgetptmsg.txtの追加されたメッセージ行を¥L_americanのgetptmsg.txtにコピーしてください。
 - ④ 日本語メッセージに対応するメッセージをP.9を参照し、英語メッセージに置き換えて保存してください。
 - ⑤ ¥L_japaneseのgetptmsg.txtを閉じてください。

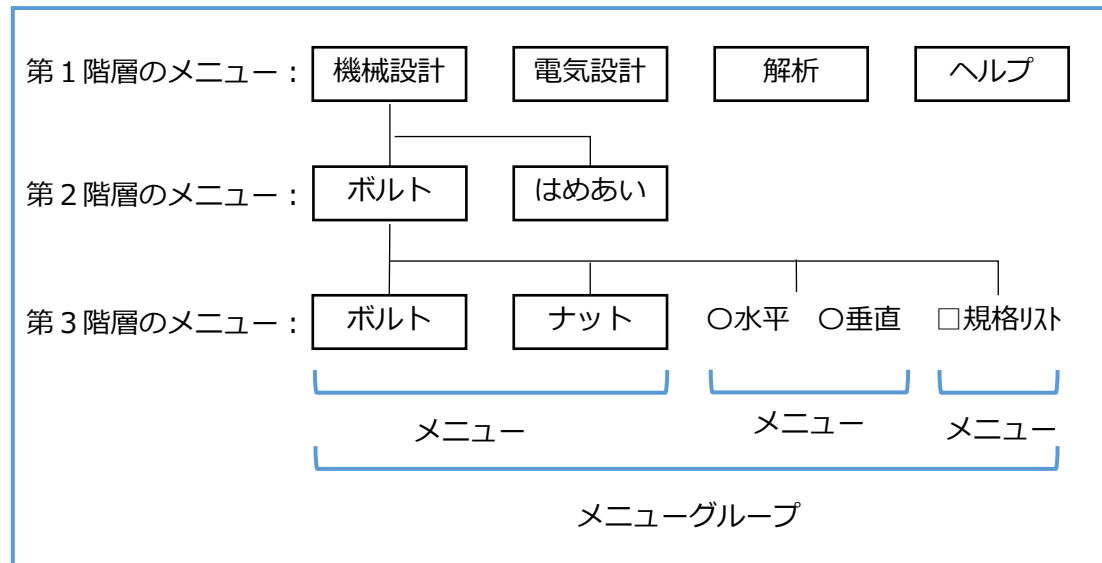


4. メニュー構造(1/4)

作成プログラムではメニューを表示します。

MC Helixのメニューは階層構造になっています。ACCESSでも同様です。

ファンクションのすぐ下に、いくつかのメニュー・ボタンが作成できます。このうち、どれかのメニュー・ボタンを押すたびに、さらに下のいくつかのメニュー・ボタンを表示することができます。



ファンクション<グループ>のメニューを例にとると次のような構造になっています。

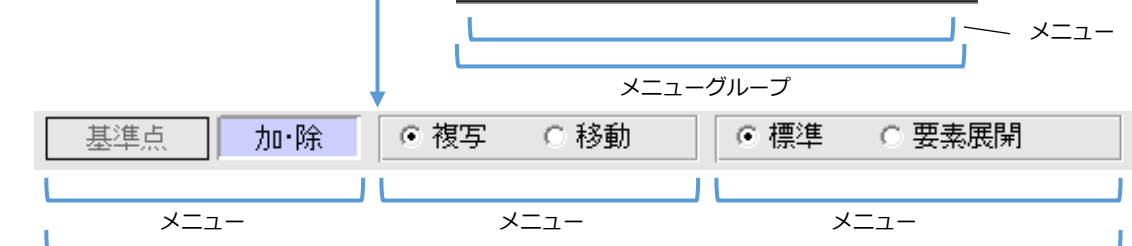
第1階層のメニュー:



第2階層のメニュー:



第3階層のメニュー:



メニューグループ

実際の表示は第1階層のメニューで【編集】を選択し、第2階層のメニューで【切取】を選択すると第3階層のメニューも一緒に表示されます。これは第3階層のメニューが1つ上位のメニュー・グループと同時に表示する指定になっているためです。また、第2階層のメニューを一度選択すると次回からは前回の選択状態で表示されます。



メニュー・グループ

それぞれの階層ごとのメニュー・ボタン群をメニュー・グループといいます。
ACCESSではこのメニュー・グループ単位でメニューを表示します。

メニュー

同じ種類のメニュー・ボタンからなる集まりをメニューといいます。このメニュー内のボタンは複数の中から1つだけしか選択できません。

メニュー・ボタンの種類

メニュー・ボタンには次の4種類があります。

ノーマル型 : 押されたボタンが選択（へこんだ）表示になります。

例) ボルト ナット

ラジオ型 : 押されたボタン（ラベルの前の記号）が選択表示になります。

例) 水平 垂直

複数項目の中から1つだけ選択できるようにする場合に使います。

チェック型 : ボタンを選択するたびにONとOFFが入れ替わる（フリップ）

例) 規格リスト

メニューです。二者択一の際に使います。

独立型 : ボタンが押されても選択（凹んだ）表示になりません。

例)

単独で押すたびに動作する機能の選択に利用します。

メニュー・グループの長さ

1つのメニュー・グループの長さは100 バイトまでです。

全角文字を2バイト、半角文字を1バイトとし、下記の基準で計算します。

ノーマル型 : ラベルの文字のバイト数 + 1バイト

ラジオ型 : ラベルの文字のバイト数 + 2バイト(○部分)

チェック型 : ラベルの文字のバイト数 + 2バイト(□部分)

独立型 : ラベルの文字のバイト数 + 1バイト

さらに、各メニュー（ボタンではない）の後に1バイト加算します。

例) ノーマル型 /ボルト/ナット/ ラジオ型 ○水平○垂直/ チェック型 □規格リスト/ (説明のため1バイト加算部分を"/"と"/"で表記しています)

<u>ノーマル型</u>	<u>ラジオ型</u>	<u>チェック型</u>
6+1バイト	4+2バイト	10+2バイト
7バイト	6バイト	12バイト

合計: 7×2(個)+6×2(個)+12×1(個)+1×3(個)=41バイト

学習機能

最後に選択されたメニューを覚えておく機能で、別のブロックのメニュー（ファンクションなど）に移動して再び前のメニューに戻ったとき、前に選択されたメニューとなります。この指定が無効なとき、常に初期状態で選択されているメニューになります。学習機能はメニュー単位で設定します。

メニュー選択の初期状態

メニュー内にひとつ選択された状態のメニュー・ボタンを定義できます。

上位メニュー・グループの表示

「上位メニュー・グループの表示」を有効にするとメニュー・グループを表示するとき、ひとつ上の階層のメニュー・グループを同時に表示します。この指定が無効なときメニューが選択されると、そのメニュー・グループは不表示になり、下の階層のメニュー・グループが表示されます。

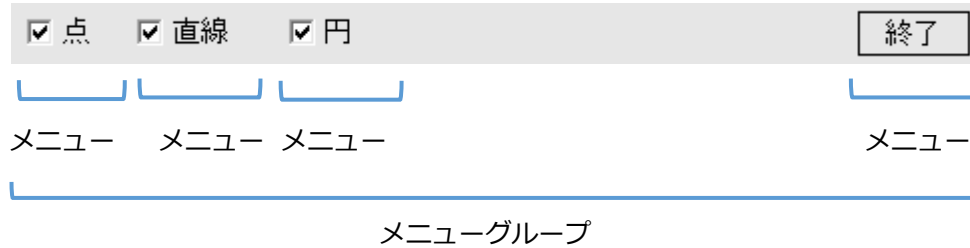
リターン・ボタン

第1階層のメニュー・グループより下のメニューを表示する場合で、その上位メニュー・グループが表示されていないとき、ひとつ上のメニューに戻すために、リターン・ボタンが自動的に付加されます。

5. 作成プログラムのメニュー仕様(1/2)

作成プログラムのメニューは次の通りです。

出力対象要素種別を指定するメニューと終了メニューです。



ブロック名 : BLK_MENU_SP8
コメント : <SP8>
ブロック識別子 : INCL_BLK_MENU_SP8

メニュー(日本語)	メニュー(英語)	種類	初期状態
点	POINT	チェック型	✓
点OFF	POINTOFF	チェック型	
直線	LINE	チェック型	✓
直線OFF	LINEOFF	チェック型	
円	CIRCLE	チェック型	✓
円OFF	CIRCLEOFF	チェック型	
終了	END	独立型	

チェック型の場合、ONのボタンとOFFのボタンを定義することによりON/OFFの切り替えを行います。
画面に表示されるメニュー・ボタンとしては1つですが、実際には2つのメニュー・ボタンの定義が必要となります。

メニューグループ : GRP_MENU_SP8_ROOT
コメント : GRP_MENU_SP8_ROOT

メニュー コメント 番号	: MENU_SP8_ROOT_PT MENU_SP8_ROOT_PT 1	: MENU_SP8_ROOT_LINE MENU_SP8_ROOT_LINE 2	: MENU_SP8_ROOT_CIRC MENU_SP8_ROOT_CIRC 3	: MENU_SP8_ROOT_END MENU_SP8_ROOT_END 4
--------------------	---------------------------------------------	-------------------------------------------------	-------------------------------------------------	-----------------------------------------------

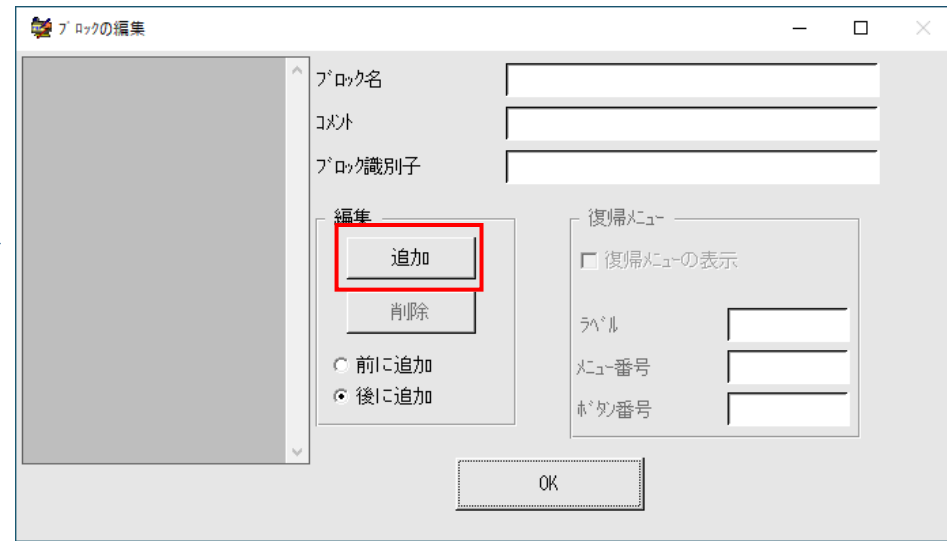
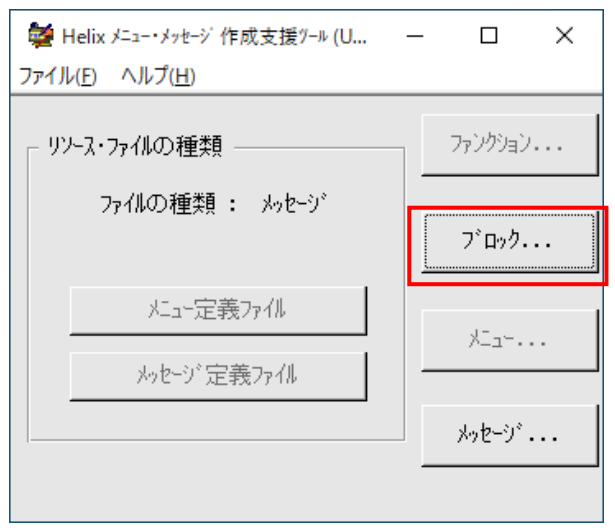
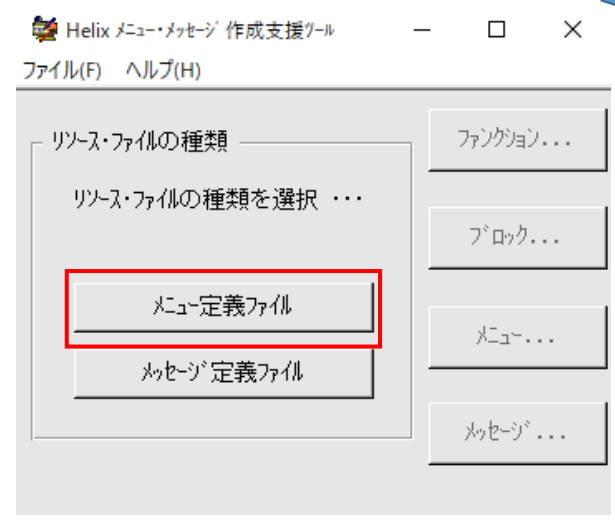
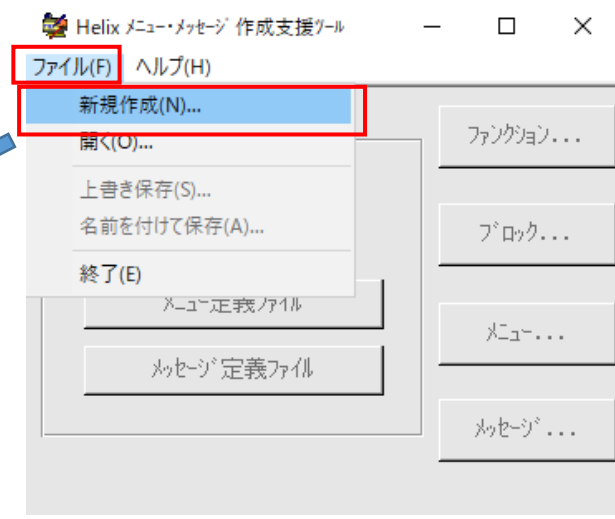
メニュー・ボタン ラベル 番号	: MI_PT_ON 点 1	: MI_LINE_ON 直線 1	: MI_CIRC_ON 円 1	: MI_END 終了 1
-----------------------	----------------------	-------------------------	------------------------	---------------------

メニュー・ボタン ラベル 番号	: MI_PT_OFF 点OFF 2	: MI_LINE_OFF 直線OFF 2	: MI_CIRC_OFF 円OFF 2	
-----------------------	--------------------------	-----------------------------	----------------------------	--

6. メニュー・メッセージ作成支援ツールによるメニューの作成(1/9)

「メニュー・メッセージ作成支援ツール」を使用して、メニュー定義ファイルを作成します。

1. 「メニュー・メッセージ作成支援ツール」を起動します。
2. メニュー・バー[ファイル]から[新規作成]を選択します。
3. [メニュー定義ファイル]を押します。
4. [ブロック]を押します。
「ブロックの編集」画面が表示されます。
5. [追加]を押します。



6. ブロックを定義します。

ブロックには次の項目があります。

- ・ブロック番号(「ブロック名」の項目で定義)
- ・コメント
- ・ブロック識別子

ブロック名	: BLK_MENU_SP8
コメント	: <SP8>
ブロック識別子	: INCL_BLK_MENU_SP8

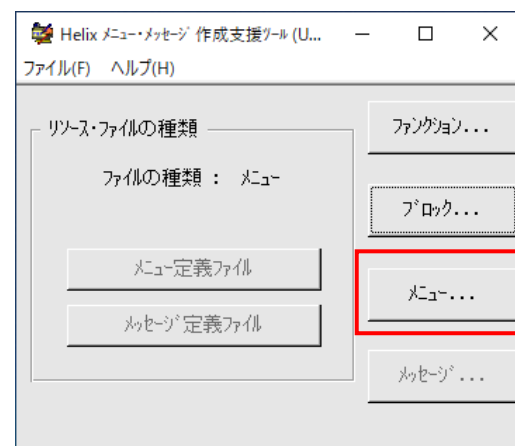
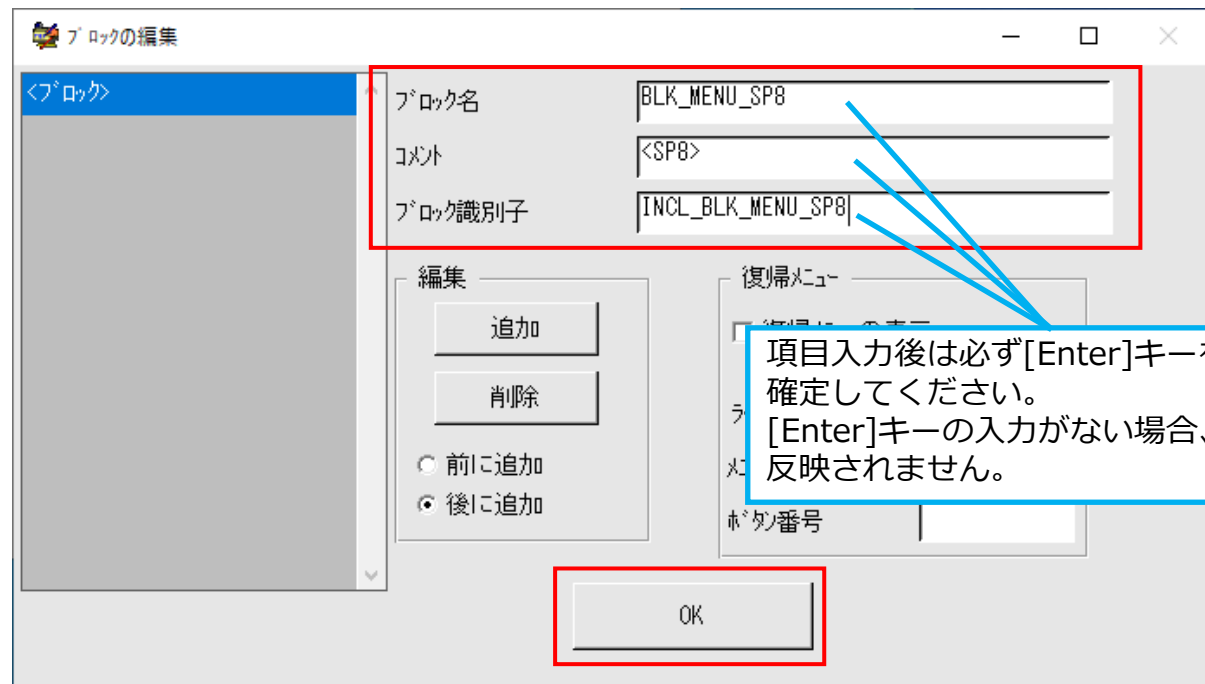
各項目入力後は確定のため[Enter]

ACCESS関数は、「ブロック番号」を使って制御します。
また、「ブロック識別子」は、コンパイルのとき使用します。

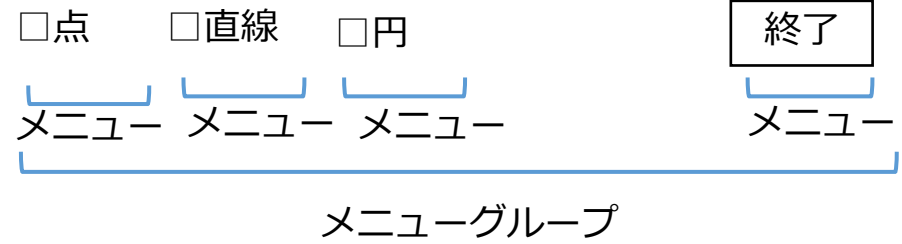
7. [OK]を押します。

8. [メニュー]を押します。

「ブロックの選択」画面が表示されます。



- 9. 6. で定義したブロック<SP8>を選択して[OK]を押します。
- 10. 「メニュー・グループの編集」画面が表示されます
- 11. メニュー・グループを定義します。



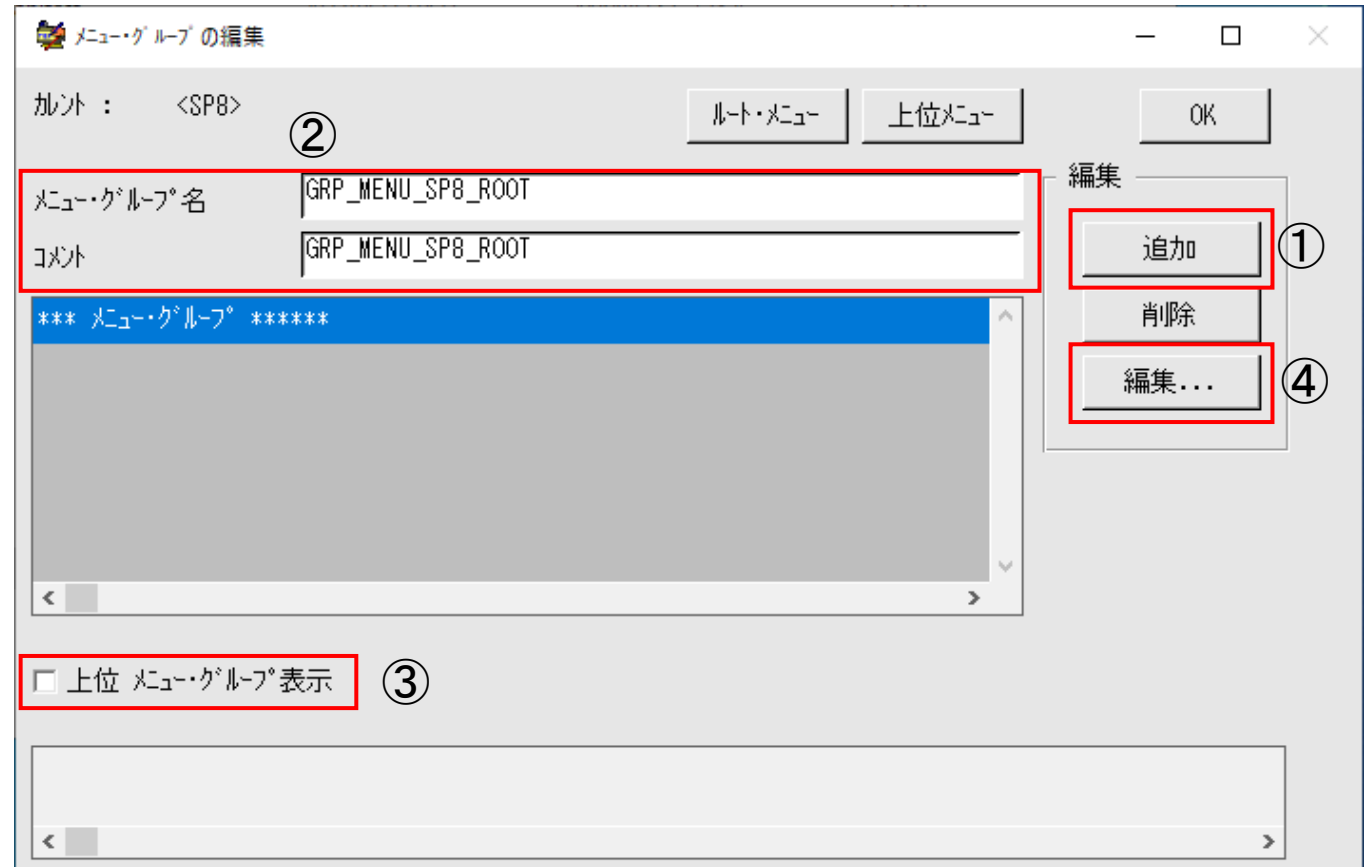
- ①[追加]を押す
- ②メニューグループ名、コメントを指定

メニューグループ : GRP_MENU_SP8_ROOT
 コメント : GRP_MENU_SP8_ROOT

各項目入力後は確定のため[Enter]

- ③「上位メニューグループ表示」のチェックをはずす
- ④[編集] ボタンを押す

- 12. 「メニューの編集」画面が表示されます。



1 3. メニューを定義します。

①種類を選択する 種類：チェック型

②[追加]を押す

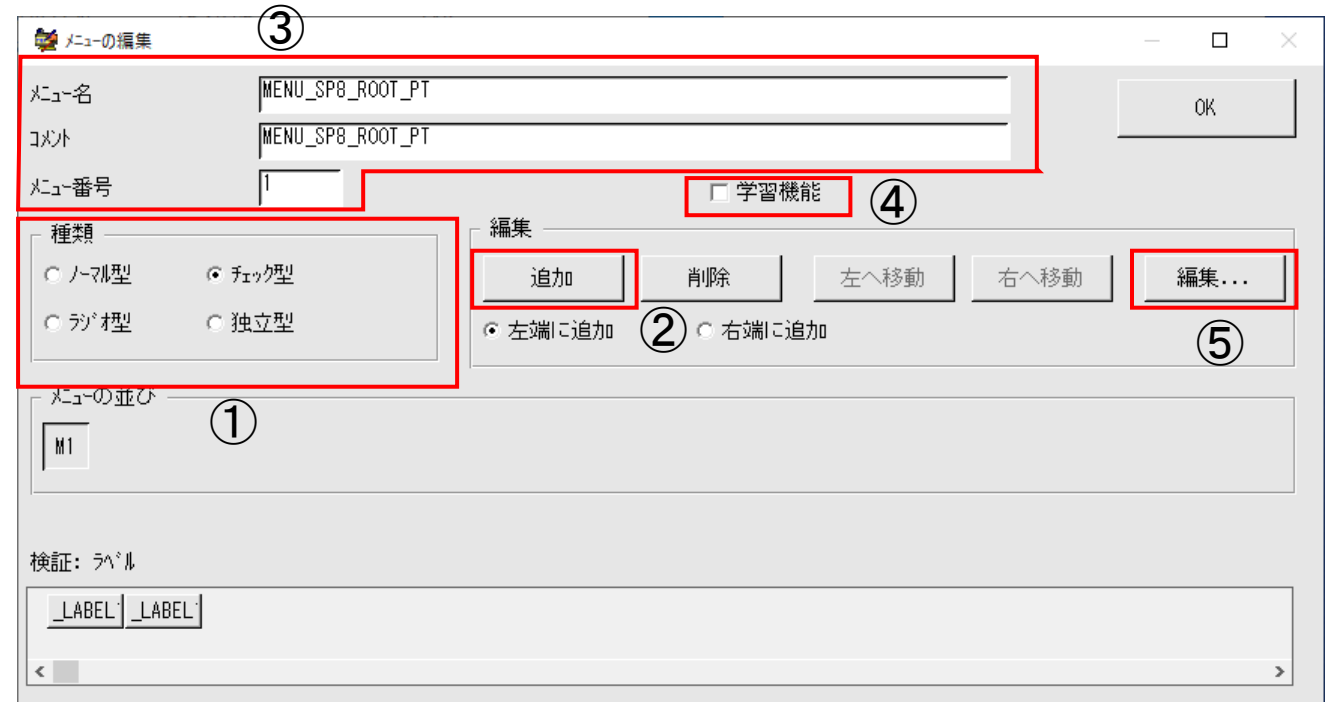
③メニュー名、コメント、
メニュー番号を入力する

メニュー名 : MENU_SP8_ROOT_PT
コメント : MENU_SP8_ROOT_PT
メニュー番号 : 1 (表示値のまま)

各項目入力後は確定のため[Enter]

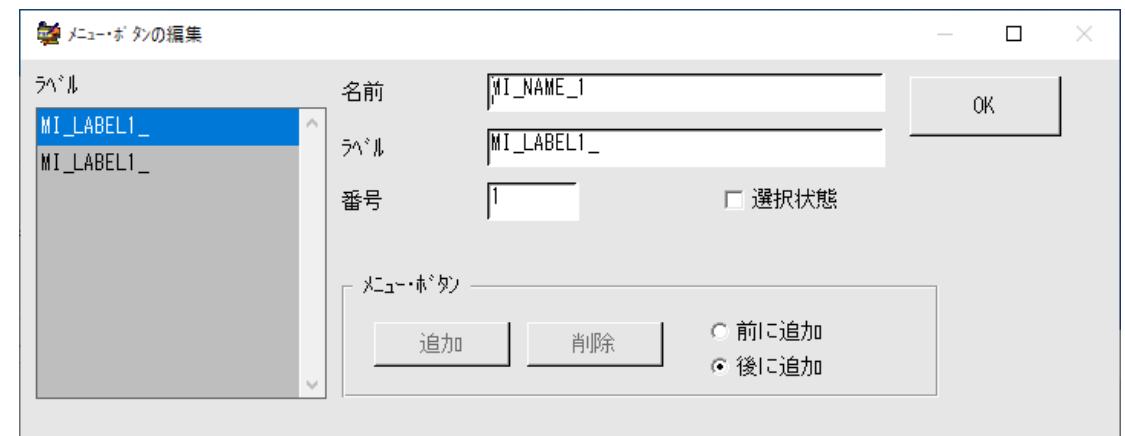
④学習機能のチェックをはずす

⑤[編集]を押す



1 4. 「メニュー・ボタンの編集」画面が表示されます。

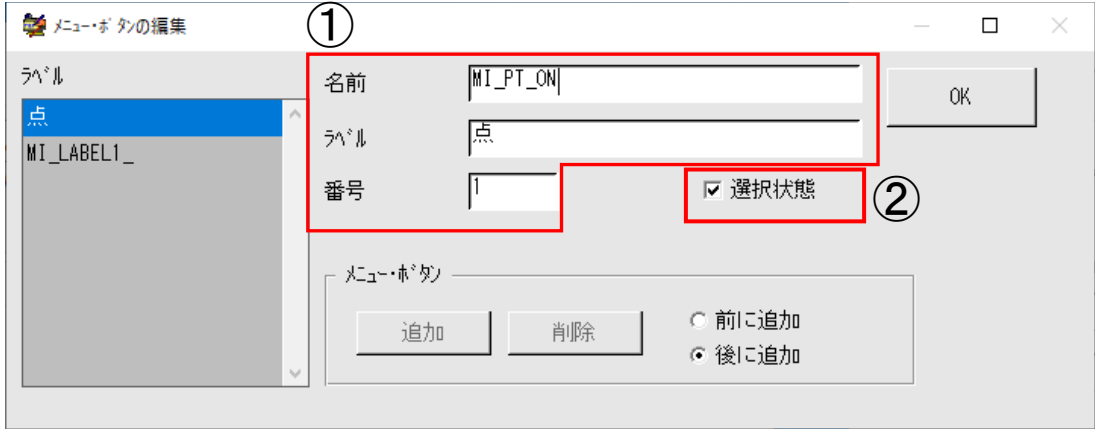
チェック型の場合、ONのボタンとOFFのボタンを定義することによりON/OFFの切り替えを行います。ON/OFF用2個と決まっているため、最初からリストに追加されています。この2個に対してラベル等を設定します。他の種類のボタンの場合、[追加]により必要なボタンを定義していきます。



- ①左側のラベルリストの1行目が選択されている状態で名前、ラベル、番号を入力する

名前 : MI_PT_ON
ラベル : 点
番号 : 1 (表示値のまま)

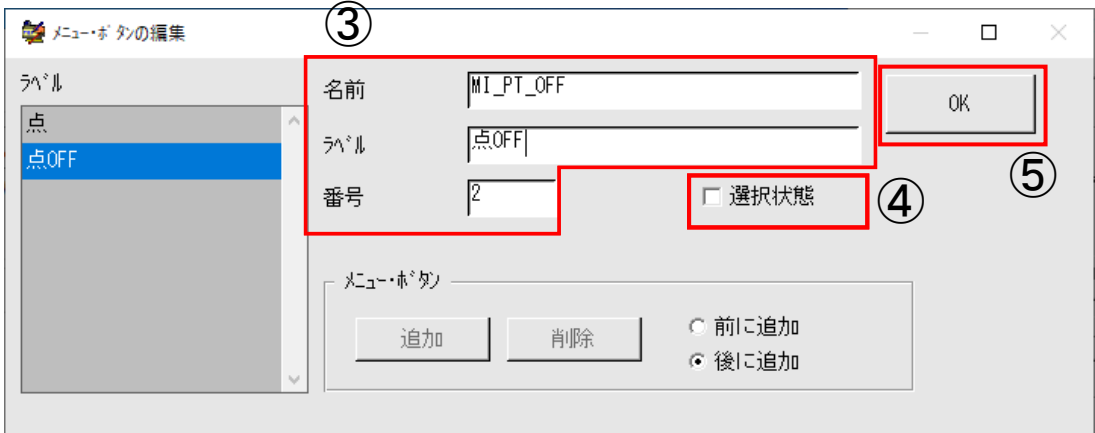
ラベルの文字がメニューとして表示されます。



- ②選択状態をチェックする
- ③左側のラベルリストの2行目を選択して名前、ラベル、番号を入力する

名前 : MI_PT_OFF
ラベル : 点OFF
番号 : 2 (表示値のまま)

ラベルの文字はメニューとしては表示されません。



- ④選択状態のチェックをはずす
- ⑤[OK]を押す

初期状態でチェックありにする場合、ONのボタンの選択状態にチェックを入れます。
初期状態でチェックなしにする場合、OFFのボタンの選択状態にチェックを入れます。

15. 「メニューの編集」画面に戻るので直線、円のメニューを定義するため13. 14. を繰り返します。

メニュー名 : MENU_SP8_ROOT_LINE
 コメント : MENU_SP8_ROOT_LINE
 メニュー番号 : 2(表示値のまま)

名前 : MI_LINE_ON
 ラベル : 直線
 番号 : 1(表示値のまま)
 選択状態 : チェックあり

名前 : MI_LINE_OFF
 ラベル : 直線OFF
 番号 : 2(表示値のまま)
 選択状態 : チェックなし

メニュー名 : MENU_SP8_ROOT_CIRC
 コメント : MENU_SP8_ROOT_CIRC
 メニュー番号 : 3(表示値のまま)

名前 : MI_CIRC_ON
 ラベル : 円
 番号 : 1(表示値のまま)
 選択状態 : チェックあり

名前 : MI_CIRC_OFF
 ラベル : 円OFF
 番号 : 2(表示値のまま)
 選択状態 : チェックなし



直線、円を初期状態でチェックありにするためONのボタンの選択状態にチェックを入れます。

16. 「メニューの編集」画面に戻るので「終了」メニューを定義します。

①種類を選択する 種類：独立型

②[右端に追加]を選択する

③[追加]を押す

④メニュー名、コメント、
メニュー番号を入力する

メニュー名 : MENU_SP8_ROOT_END
 コメント : MENU_SP8_ROOT_END
 メニュー番号 : 4(表示値のまま)

各項目入力後は確定のため[Enter]

⑤学習機能のチェックをはずす

⑥[編集]を押す



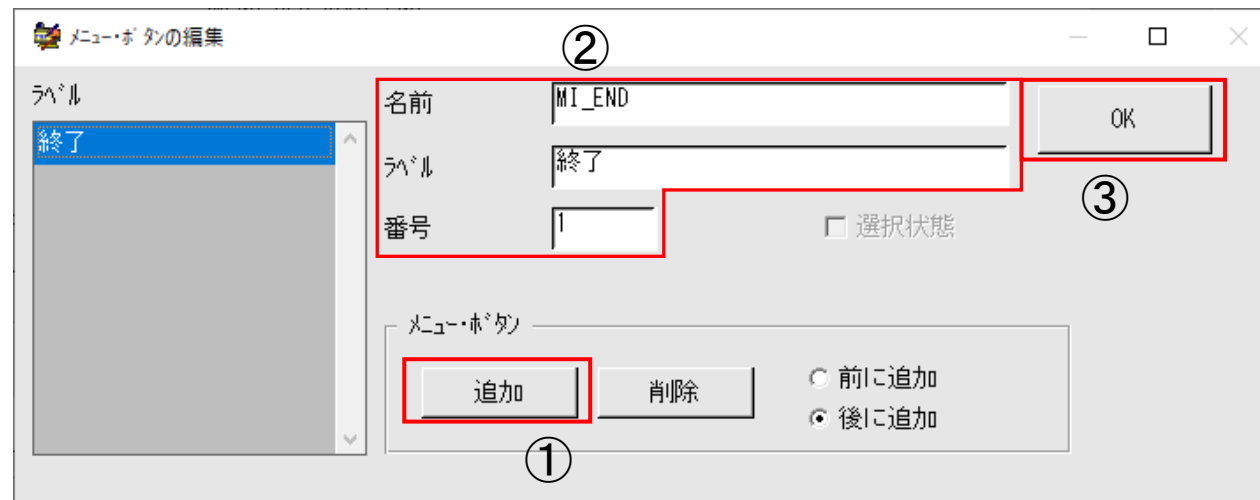
17. 「メニュー・ボタンの編集」画面が表示されます。

①[追加]を押す

②名前、ラベル、番号を入力する

名前	: MI_END
ラベル	: 終了
番号	: 1(表示値のまま)

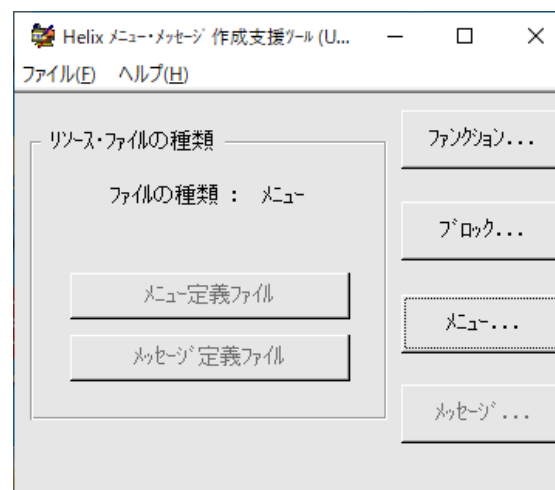
③ [OK]を押す



18. 「メニューの編集」画面に戻りますので [OK]を押します。

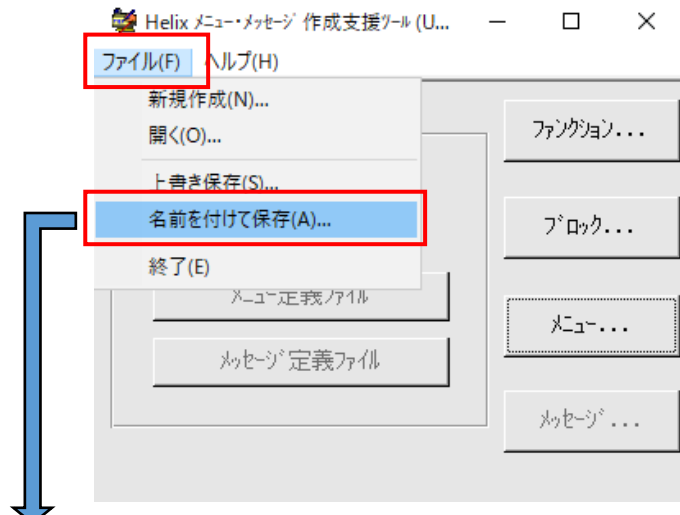
19. 「メニュー・グループの編集」画面に戻りますので [OK] を押します。

20. 起動直後の画面まで戻ります。

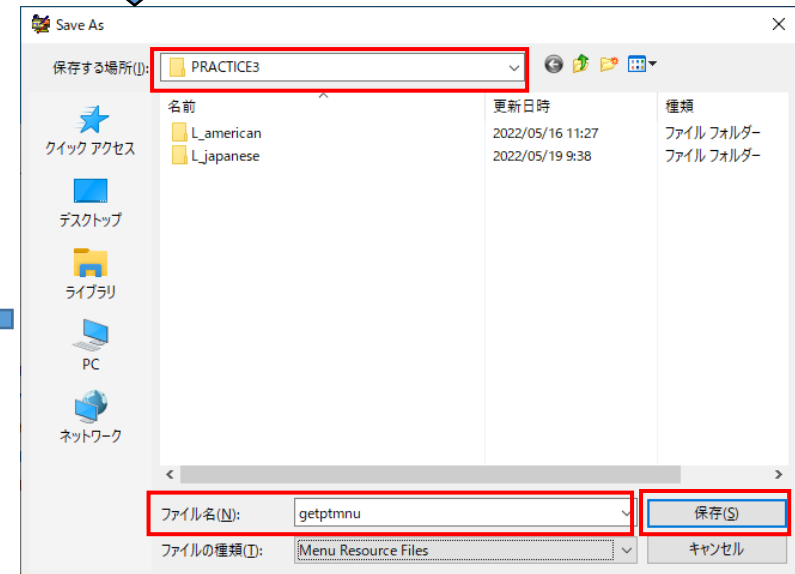


- 2 1. メニュー・バー[ファイル]から[名前を付けて保存]を選択します。
- 2 2. 「Save As」画面が表示されるので保存場所とファイル名を指定して[保存]を押します。。

保存場所 : c:\mchelix\HDD\ACCESS\PRACTICE3
 ファイル名 : getptmnu



- 2 3. 「言語の選択(別名保存)」画面が表示されるので[japanese]を選択して[OK]を押します。
- 2 4. c:\mchelix\HDD\ACCESS\PRACTICE3の下に getptmnu.resと getptmnu.dfnが¥L_japaneseの下に getptmnu.txtが作成されていることを確認してください。



- 2 5. 英語環境の文字情報ファイルを作成します。
 - ① ¥L_japaneseのgetptmnu.txtを ¥L_americanにコピーします。
 - ②コピーしたgetptmnu.txtをエディタで開いて、 P.17を参照して日本語メニューに対応する英語メニューに置き換えてファイルします。

第3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

リソースファイルの準備はできましたので、次はコードの記述です。
作成プログラムの全コードは[こちら](#)です。

プログラム作成 2 からの主な改修箇所は次の通りです。

1. メニューの設定
2. 対話操作の制御
3. グループ化範囲の指定と矩形作成
4. グループ化処理
5. 処理の繰り返し
6. CSV出力の対象要素種別をメニューから取得
7. 追加対象要素の情報取得とCSV出力
8. グループ化範囲指定時の矩形ラバーバンド表示

ソースファイル内での対応箇所

1	
2	
3	A
4	B
5	
6	
7	
8	C

プログラム 2 で作成し、¥PRACTICE3 にコピーしたソースファイル getpoint.c を修正したり、コードを追加したりしながら進めます。

2. コンパイル・リンク方法

コンパイル・リンクは¥PRACTICE2からコピーしたバッチ・ファイル、リンクコントロールファイルなどをそのまま使用します。

1. コマンド・プロンプトを起動します。
2. 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¥PRACTICE3”と入力して[Enter]

3. バッチファイルを実行します。

コマンドプロンプトで“getpoint.bat”と入力して[Enter]

3. メニューの設定 **1**

「メニュー・メッセージ作成支援ツール」により作成したメニュー定義ファイルを利用してメニューを表示する準備を行います。

インクルード・ファイル： getptmnu.dfn

ブロック識別子： INCL_BLK_MENU_SP8

ブロック名： BLK_MENU_SP8

1. ブロック識別子を指定してインクルードファイルをインクルードします。
2. メニュー定義ファイルを、名前と名前のバイト数を指定してMC_menuinitで読み込みます。
3. 使用するメニューが登録されているブロックをMC_menuinitで指定してメニュー・ブロックの設定を開始します。
4. 終了処理部分でメニュー・ブロックの設定を終了します。
5. 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。（まだ実行は行いません。）

```

#define INCL_BLK_MENU_SP8
#include "getptmnu.dfn"
:
:
retc = MC_menuinit ( 8L, "getptmnu" );
if ( retc ) goto exit;

retc = MC_menuinitbk ( BLK_MENU_SP8 );
if ( retc ) goto exit;
menublk = 1L;
:
:
exit:
if ( menublk ) MC_menuendbk ( BLK_MENU_SP8 );

```

1.ブロック識別子を指定する
1.インクルードする
2.名前とバイト数を指定して読み込む
3.ブロックの設定を開始する
3.ブロックの設定を終了する時のため正常に開始されたことを保持するフラグを立てる
4.ブロックの設定を終了する

メニュー関係の関数を使用する前に必ずMC_menuinitを実行するよう記述します。この関数が2回以上実行された場合、新しくメニュー定義ファイルを読み込み、前回読み込まれていたファイルの内容と置き換えます。MC_menuinitbkでブロックを指定するとこれ以降MC_menuendbkを実行するまでの間にメニュー関係の関数で設定を行うメニューはこのブロックのものが使用されます。

4. 対話操作の制御(1/2) 2

ACCESSでは2Dモジュールと同様の対話操作ができます。

まず、メニューを表示し、「終了」メニューかファンクションが選択された場合に作成プログラムを終了できるようにしてみましょう。

1. MC_menusetでメニュー・グループ番号を指定して、表示するメニューを指定します。
2. ACCESSでは次の対話機能が利用できますが、操作待ちの前にそれらの機能を受け付けるか否かを設定しておく必要があります。
 - ・ 要素の選択
 - ・ 位置の指示
 - ・ キー入力
 - ・ YNキー
 - ・ メニューの選択
 - ・ ファンクションの選択
 - ・ ダイアログ・ボックスの操作

今は「メニューの選択」と「ファンクションの選択」のみを受け付ける設定にします。

```

#define   IND    2L
#define   MENU   5L
#define   FK     6L
    
```

機能を示す番号を分かりやすい文字で定義しておく

1.表示するメニューグループ(□点 □直線 □円 /終了/)を指定する
 画面上にメニューが実際に表示されるのは操作待ちになった時です


```

MC_menuset ( GRP_MENU_SP8_ROOT );
    
```

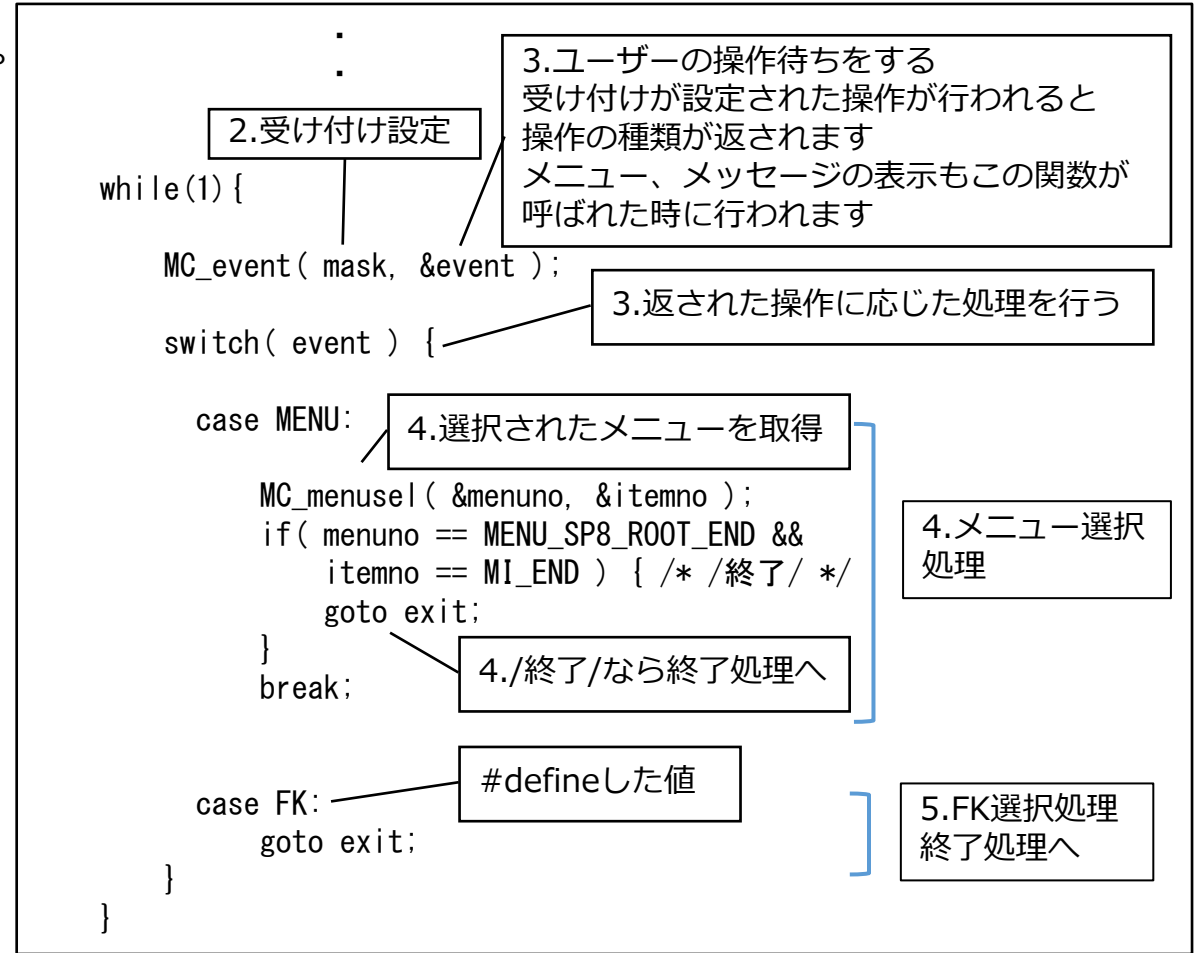


```

mask[0] = 0L; /* SELECT */
mask[1] = 0L; /* IND   */
mask[2] = 0L; /* KEYIN */
mask[3] = 0L; /* YN    */
mask[4] = 1L; /* MENU  */
mask[5] = 1L; /* FK    */
mask[6] = 0L; /* POPUP */
    
```

2.操作待ちの前にそれぞれの機能を受け付けるか否かの設定を行う
 メニュー選択とファンクション選択のみ行うのでその機能の部分に"1"を設定する

3. ユーザーの操作待ちをする関数MC_eventを呼びます。
この関数を実行すると2. で受け付ける設定をした対話操作が行われるまでプログラムは待ち状態になります。受け付ける設定の対話操作が行われるとどの操作が行われたかがeventに返されます。それぞれの操作に応じて処理を行います。
4. メニューが選択された場合の処理を行います。
MC_menuselで選択されたメニュー番号とボタン番号を取得します。それが「終了」の場合、処理を終了し、それ以外の場合は何もせず、再度操作待ちを行います。
5. ファンクションが選択された時は処理を終了します。
6. 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。



プログラム作成2と同様にACCESS.LSTに追記して実行することもできますが、今回はファンクション・バーに登録して実行してみましょう。その方法を次のページから説明します。

5. ファンクションへの登録と実行(1/4)

ファンクションに作成プログラムを割り付けて実行します。

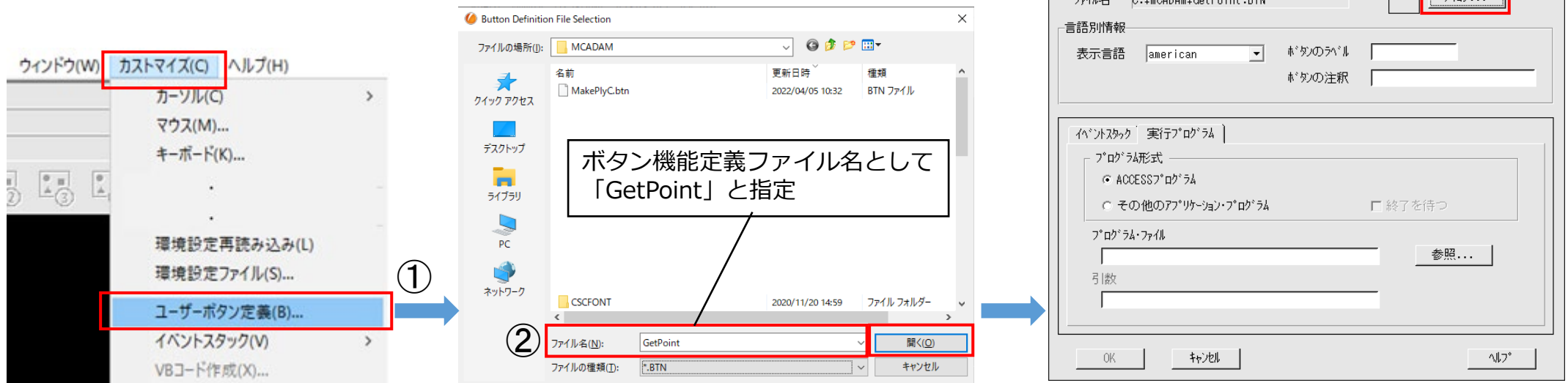
定義する機能の情報を記載した「ボタン機能定義ファイル(xxxxxxx.btn)」を作成し、このファイルをファンクションのカスタマイズ機能でボタンに割り付けることでファンクションボタンからプログラムが実行できるようになります。

ファンクション名<点CSV出力>



1. 「ボタン機能定義ファイル」を作成します。

- ①MC Helixを起動し、メニュー・バーから[カスタマイズ]-[ユーザーボタン定義]を選択する
- ②ファイル名(GetPoint)を指定して[開く]ボタンを押す(C:¥MCADAMの下に作成するよう指定)
- ③「ユーザーボタン定義の設定」画面が開くので[アイコン...]を押す



- ④ 「アイコンの選択」画面が開くので、機能を割り当てたファンクションがアイコンで表示されたときに使用するアイコンを選択して、[OK]を押す
- ⑤ 「ユーザーボタン定義の設定」画面に戻るのでボタンのラベルを指定する
表示言語を切り替えて、日本語、英語両方を指定する

日本語：「点CSV出力」

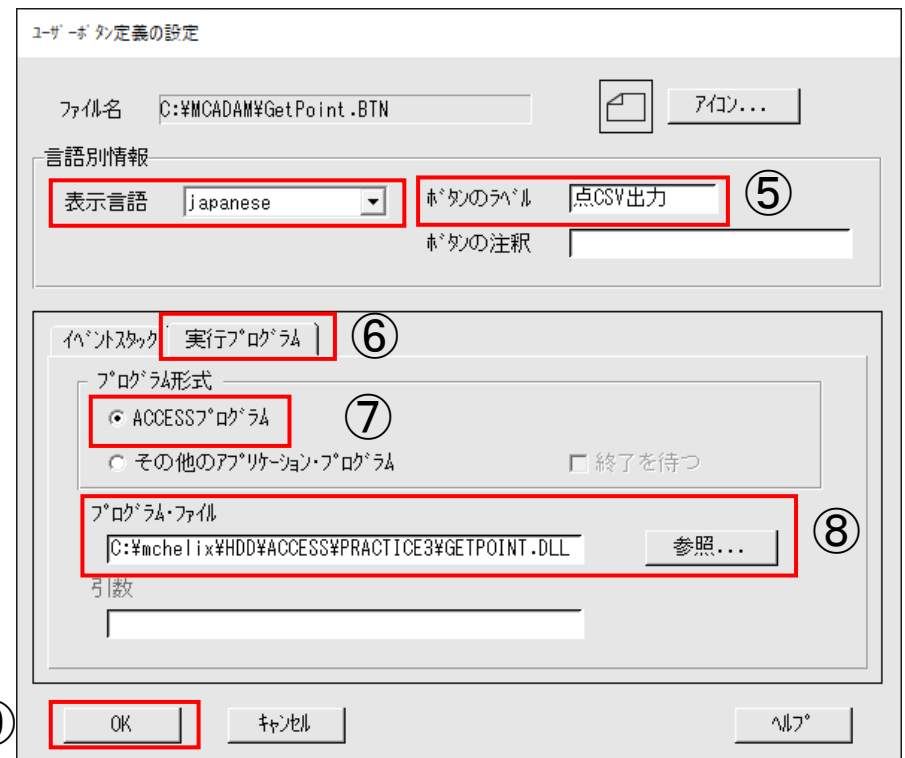
英語：「GetPoint」

全角文字を2バイトとして計算し、10バイトまでで指定してください。
入力しなかった場合「Untitled」と表示されます。

任意のアイコンを選択する



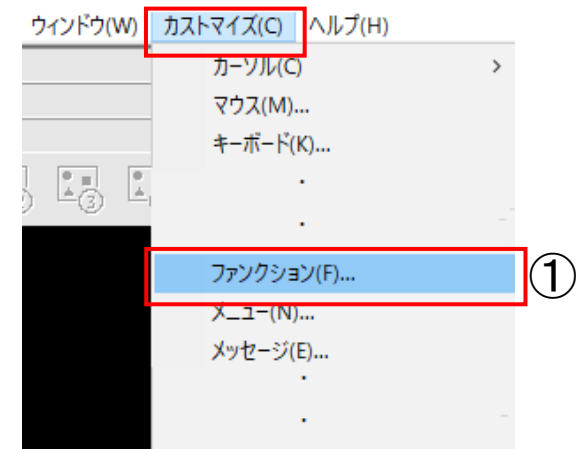
- ⑥ 「実行プログラム」シートを選択する
- ⑦ 「プログラム形式」の「ACCESSプログラム」を選択する
- ⑧ 「プログラム・ファイル」に割り付けるプログラムを指定する
キー入力するか[参照...]ボタンから作成プログラム
「C:\mchelix\HDD\ACCESS\PRACTICE3\GETPOINT.DLL」
を指定する
- ⑨ [OK]を押す(ファイルが保存されます)



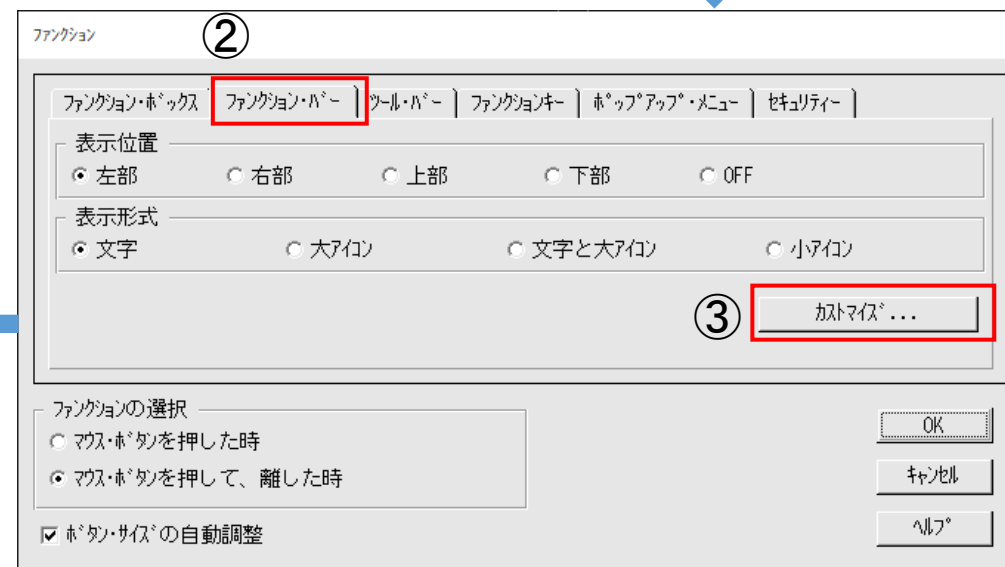
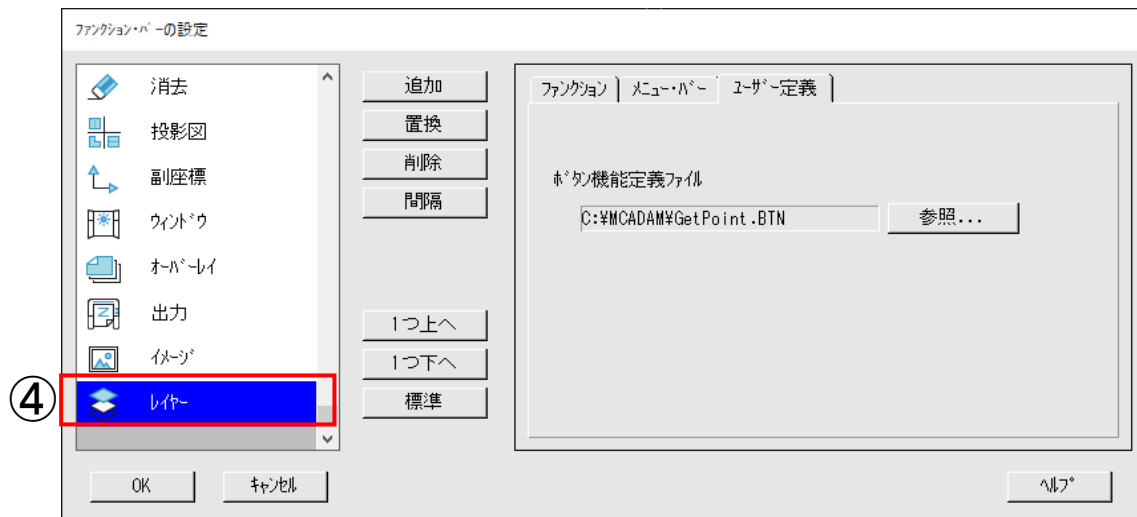
2. 作成した「ボタン機能定義ファイル(GetPoint.btn)」をファンクションカスタマイズで割り付けます。

- ① MC Helixのメニュー・バーから[カスタマイズ]-[ファンクション]を選択する
- ② 「ファンクション」画面が表示されるので「ファンクション・バー」シートを選択する

他のシートを選択して同様の設定をすることでファンクション・ボックスやツール・バーに割り付けることもできます。



- ③ [カスタマイズ...]ボタンを押す
- ④ 「ファンクション・バーの設定」画面が表示されるので左側のリストで一番下の項目を選択する



⑤ 「ユーザー定義」シートを選択し、[参照]ボタンを押して、ファイル選択画面で1. で作成した「ボタン機能定義ファイル」を指定する C:¥MCADAM¥GetPoint.btn

⑥ [追加]ボタンを押す

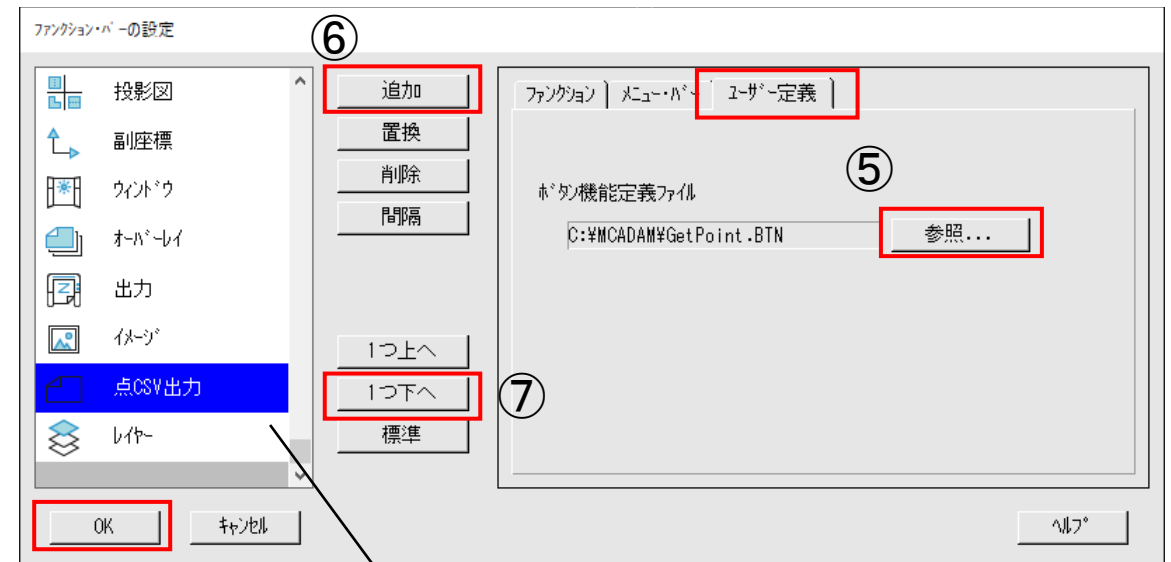
⑦ ④で選択した項目の上側に追加されるので[1つ下へ]を押してリストの一番下に移動する(任意)

⑧ [OK]を押す

⑨ 「ファンクション」画面に戻るので[OK]を押す

3. ファンクション<点CSV出力>が追加されていることを確認してください。

4. ファンクションを選択するとメニューが表示され、他のファンクションを選択するかメニュー/終了/を選択するとプログラムが終了することを確認してください。



表示したい位置へ[1つ上へ]
[1つ下へ]で移動できます

6. グループ化範囲の指定と矩形作成(1/4) 3 A

グループ化するための矩形範囲をIND,INDにより指定します。

1. 位置のIND操作を受け付けるよう設定します。
2. 同様の操作処理を共通化できるよう処理の進行状況を示す変数を設定します。
3. 進行状況に応じて操作指示用のメッセージを表示します。
4. 操作待ちします。
5. INDされた位置を取得します。進行状況に応じた配列変数にセットします。

ACCESSにはビューごとに定義できるビュー座標系と図面に対して一義的に決められ、ビューとは関係なく、図面全体に有効なペーパー座標系があります。MC_positionでは両方の座標系で位置が取得できます。

2回目のINDの場合、操作待ちを抜けて次の処理に移ります。1回目のINDの場合、進行状況を進めて再度操作待ちします。

```
mask[1] = 1L; /* IND   */
status = 1L;
while(1) {
  switch( status ) {
    case 1: MC_msgset( MSG_IND1 ); /* "対角の第1点を選択" */
           break;
    case 2: MC_msgset( MSG_IND2 ); /* "対角の第2点を選択" */
           break;
  }
  MC_event( mask, &event );
  switch( event ) {
    case IND:
      MC_position( &vxy[status-1][0], &vxy[status-1][1],
                  &pxy[status-1][0], &pxy[status-1][1] );
      if( status == 2L ) {
        goto group;
      }
      status++;
      break;
    case MENU:
      .
      .
  }
}
```

1.INDを受け付けるよう設定

2.処理の進行状況を示す変数を設定 (スタート時を1とする)

3.処理の進行状況に合わせてメッセージを設定する

4.操作待ちとメニュー・メッセージ表示

5.INDされた位置を取得

5.2回目のINDの場合、グループ化処理に進む

5.1回目のINDの場合、矩形のもう一方を指示するよう再度操作待ちへ

グループ化範囲を示す矩形を作成、表示します。
また、プログラムを終了する時は矩形が残らないよう削除する処理を入れます。

1. 矩形のポインターを保持する変数を宣言します。
削除のため、すでに作成済みか判定できるように初期値は0を設定し、作成されたら矩形のポインターが保持されるようにします。
2. 矩形を作成する部分は内部関数とします。
INDされた位置のペーパー座標を渡して矩形を作成し、そのポインターを返します。

範囲指定の矩形はビュー座標には関係なく画面に対して平行に作成し、図面全体に有効とするためペーパー座標系で作成します。

3. 矩形が作成されたら、通常表示します。
4. プログラムを終了する時は矩形を削除します。

先にMC_budelで要素を削除するとそのポインターは無効となり、後で表示を消去しようとMC_budspeを呼んでもエラーになります。関数を呼び出す順番に注意してください。

```

long   rectptr=0L;
      .
      .
while(1){
      .
      .
      MC_event( mask, &event );
      .
}

group:
  iret = CreateRect ( 1L, pxy[0], pxy[1], &rectptr );
  if ( iret == 0L ) {
    MC_budspe ( 2L, 1L, 1L, &rectptr, 0L, &lenerr, ierrar, 1L );
  }
      .
      .

exit:
  if ( rectptr != 0L ) {
    MC_budspe ( 3L, 1L, 1L, &rectptr, 0L, &lenerr, ierrar, 1L );
    MC_budel ( 1L, 1L, &rectptr, 2L );
  }
    
```

1.矩形のポインターを保持する変数

2回目のIND指示後、範囲を示す矩形作成処理に移る

2.INDされた位置のペーパー座標で矩形作成関数を呼ぶ

3.矩形が作成できた時は表示する

4.プログラム終了時、矩形が作成されたままなので削除する

4.先に画面から要素表示を消去した後、要素を削除する

矩形を作成する部分を内部関数として作成します。要素をペーパー座標系で作成するため、簡易関数 (MC_rct)ではなく、MC_buaddを使用します。MC_buaddは要素の種類を指定し、要素の種類ごとに定められた形式(モデル・データ形式)でデータを渡して要素を作成します。オプションによりビュー座標系、ペーパー座標系どちらでも作成できます。

『ACCESS関数解説書』の「モデル・データ形式」参照

1. 要素のデータはlong型の配列変数で渡しますが、さまざまなデータ型の情報を混在して設定するため共用体として宣言すると便利です。
2. 各データは配列内の位置を示すインデックスを使用して参照します。そのインデックスを分かりやすい文字で定義します。
3. 矩形は中心のX座標、Y座標、幅、高さの情報を与えて作成するため、INDされた位置からそれらを計算します。

```
static long CreateRect ( long mdlno, double xy1[], double xy2[],
                        long *elmptr )
{
    short   ret;
    long    mptr;
    double  xctr, yctr, width, height;

    union ELM — 1.モデル・データを参照するための共用体を宣言する
    {
        long   iarray[300];
        float  rarray[300];
        double darray[150];
        char   carray[300][4];
    };
    union  ELM  ELM;

    /* Index */
    #define  INDXG      ELM.iarray[0]
    #define  INDXS      ELM.iarray[1]

    /* Get center, width, height */
    xctr = ( xy1[0] + xy2[0] ) / 2.0;
    yctr = ( xy1[1] + xy2[1] ) / 2.0;
    width = fabs( xy2[0] - xy1[0] );
    height = fabs( xy2[1] - xy1[1] );
    — 3.INDされた対角の2点から
    矩形の中心のX,Y座標、幅、
    高さを計算する

    if ( width <= 0.0 || height <= 0.0 )
    {
        return ( 1 );
    }
}
```


4. 各要素に共通する共通情報と各要素個別の要素別情報への配列内でのインデックスを設定します。
5. 共通情報を設定します。
精度は倍精度(2)、色はデフォルト色(0)を指定
6. 要素別情報を設定します。
精度に倍精度を指定していますので、実数値はdouble型になります。インデックスはlong型配列内での位置を示しますので注意してください。

```
iarray[0] } darray[0]
iarray[1] }
iarray[2] } darray[1]
iarray[3] }
iarray[4] } darray[2]
iarray[5] }
```

共用体のdouble型配列の添え字の値はlong型の1/2の値なので [(INDXS+5)/2] などになります。

7. MC_buaddをオプションはペーパー座標(1)で要素種別番号は矩形(1210)で呼び出します。
8. 作成された要素のポインタを返します。
9. コンパイル・リンクしてください。実行するとIND,INDで矩形とエラーメッセージが表示され[OK]を押すと矩形が消去されて処理が終了することを確認してください。

```
/*----- Indes Parameters -----*/
ELM. iarray[0] = 5L;
ELM. iarray[1] = 25L;

/*----- Global Information -----*/
ELM. iarray[INDXG-1] = 0;
ELM. iarray[INDXG ] = 2;
ELM. iarray[INDXG+1] = 0;
ELM. iarray[INDXG+2] = 0;

/*----- Specific Information -----*/
ELM. darray[(INDXS-1)/2] = xctr;
ELM. darray[(INDXS+1)/2] = yctr;
ELM. darray[(INDXS+3)/2] = width;
ELM. darray[(INDXS+5)/2] = height;
ELM. iarray[INDXS+8 ] = 0L;

ret = MC_buadd( 1L, mdlno, 1210, ELM. iarray, &mptr, 20L );

if ( ret == 200 ) return ( 200 );
else if ( ret!=0 && ret!=1 ) return( 1 );

*elmptr = mptr;

return ( 0 );
}
```

4.インデックスを設定
インデックスは解説書の表記に合わせ奇数値で設定すると混乱がない

5.共通情報を設定

/* 未使用 */
/* 精度 */
/* 色番号 */
/* 未使用 */

6.要素別情報を設定

/* 中心のX座標 */
/* 中心のY座標 */
/* 幅 */
/* 高さ */
/* 塗りつぶしなし */

7.ペーパー座標で矩形を作成する

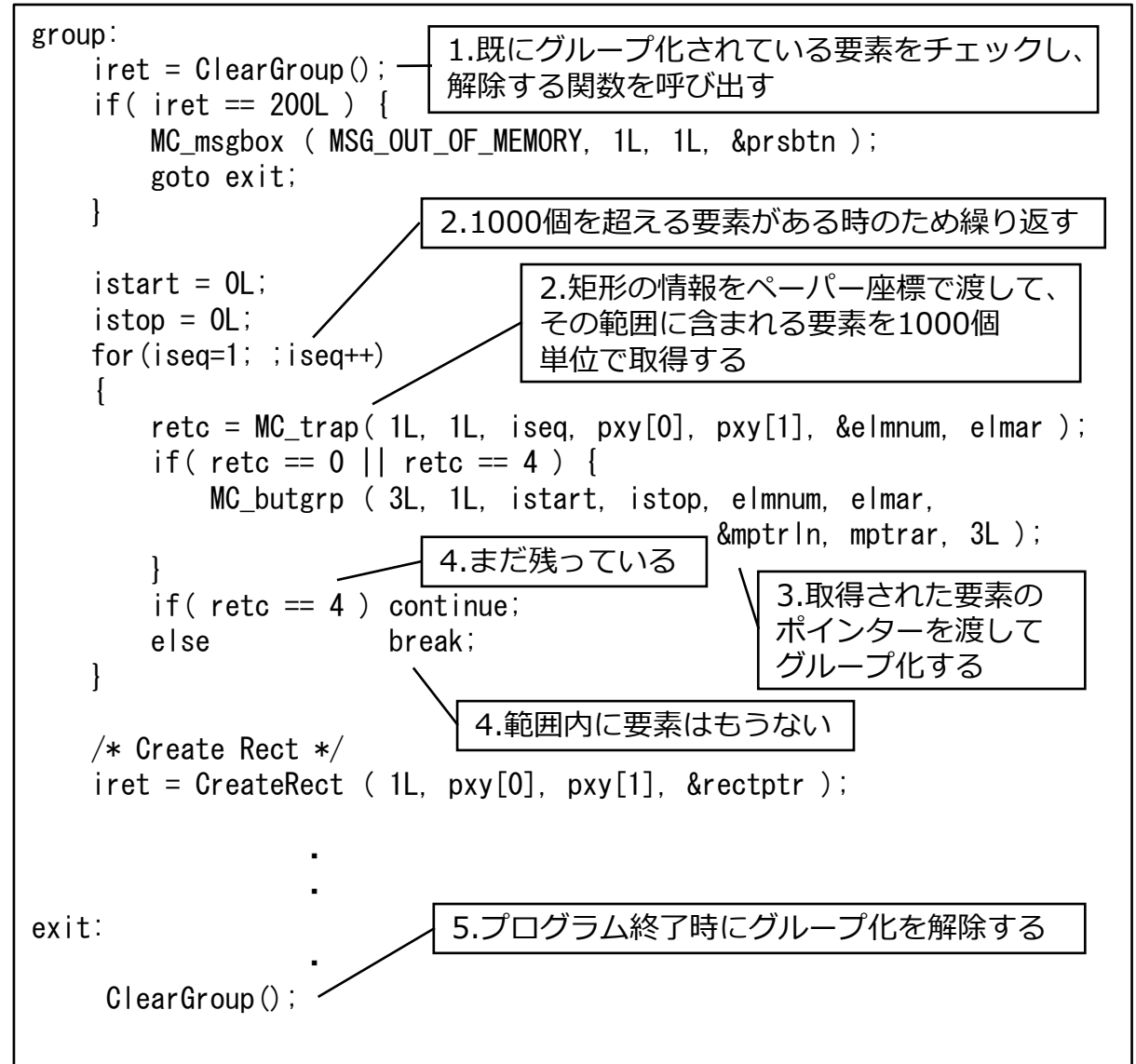
6.long型で渡す

8.作成された矩形のポインタを返す

7. グループ化処理(1/2) 4 B

IND,INDで矩形が指定されたら矩形内の要素を取得してグループ化します。

1. 既にグループ化されている要素が存在する場合
その要素群に追加してグループ化されるため
既存グループは解除します。
解除処理は内部関数を呼び出して行います。
2. MC_trapにより矩形に囲まれた範囲の要素を取得します。
3. 取得できた場合、その要素群をMC_butgrpで
グループ化します。2Dモジュールでグループ化
したのと同じ状態になり、既存コードの
グループ化要素を取得する部分につながります。
(グループ化を省略し、MC_trapで取得した
要素を直接処理対象とすることも可能です。)
4. 要素がまだ存在している場合は、さらに取得し、
終了の場合は次の処理に進みます。
5. プログラム終了時にはグループ化を解除します。1. と同じ内部関数を呼び出します。



グループ化を解除する内部関数を作成します。

1. グループ化されている要素の数を取得します。
グループ化要素が存在しない場合は何もせず
戻ります。
2. 要素が存在する場合は要素の個数分の領域を
確保します。
3. グループ化されている要素のポインターを
取得します。
4. 取得した要素のグループ化を解除します。
5. 確保した領域を解放して戻ります。
6. 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。

実行して矩形内に含まれる点がハイライト表示
していることを確認してください。

```

static long ClearGroup( void )
{
    long    istart, istop;           /* butgrp    */
    long    mptrln, mptrar[2];      /* butgrp    */
    long    numptr;
    long    *nptrar=NULL;

    istart = 0L;
    istop  = 0L;
    mptrln = 0L;
    MC_butgrp ( 900L, 1L, istart, istop, mptrln, mptrar,
                &numptr, nptrar, 5L );

    if ( numptr > 0 )
    {
        nptrar = malloc ( numptr * sizeof(long) );
        if ( nptrar == NULL ) return( 200L );
    }

    MC_butgrp ( 5L, 1L, istart, istop, mptrln, mptrar,
                &numptr, nptrar, 5L );

    MC_butgrp ( 4L, 1L, istart, istop, numptr, nptrar,
                &mptrln, mptrar, 4L );

    if ( nptrar != NULL ) {
        free ( nptrar );
        nptrar = NULL;
    }
}
return( 0L );
}
    
```

1.グループ化されている要素の数を取得する

2.個数分の領域を確保する

3.グループ化されている要素のポインターを取得する

4.取得したポインターを渡してグループ化を解除する

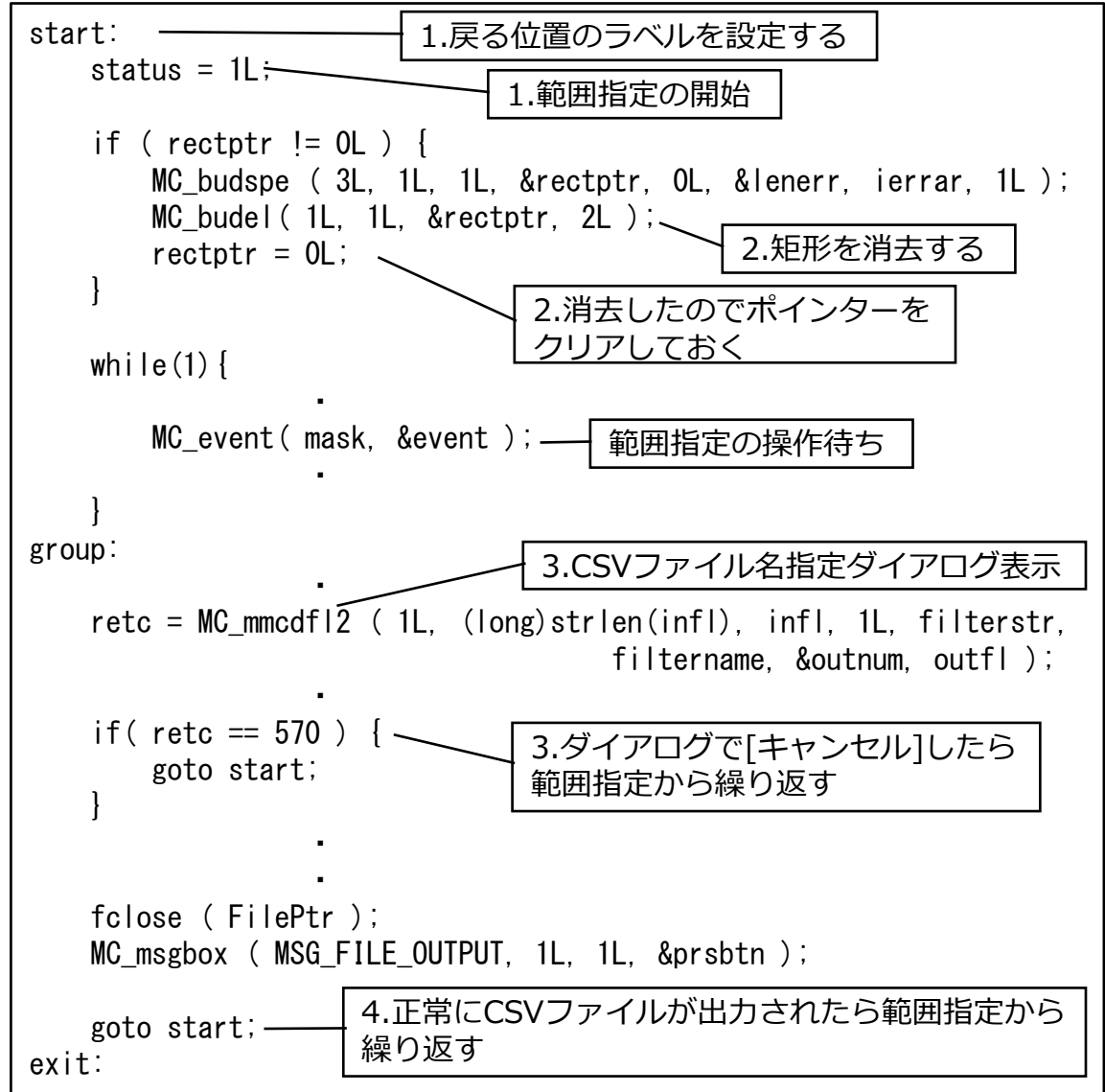
5.確保した領域を解放する

8. 処理の繰り返し 5

プログラム作成 2 ではCSV出力を一度行くと2Dモジュールに戻ってしまいますが、グループ化の範囲を変更して繰り返しCSV出力できるようにします。

1. 繰り返しは範囲指定の操作待ちの前、進行状態1に戻るよう、“start”ラベルを設定します。
 2. 前の処理で矩形が作成・表示されていた場合、削除します。
 3. CSVファイル名の指定ダイアログで[キャンセル]した場合、“start”に戻ります。
 4. CSVファイルを正常に出力した後も“start”に戻ります。その他にグループ化された要素がなく「グループ化された要素が見つかりません」というエラーが発生した場合も“start”に戻ります。
 5. 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。
- 実行して範囲指定、CSV出力が繰り返し行えること、

ファイル名指定で[キャンセル]すると範囲指定に戻ること、矩形が削除されることなどを確認してください。



9. CSV出力の対象要素種別をメニューから取得(1/2) 6

点のみが対象要素でしたが、直線・円を追加し、どの要素種別を対象とするかメニュー選択で指定できるようにし、対象要素を抽出します。

1. 対象要素種別を選択するメニューのチェックON/OFFの状態と対象要素の要素種別番号を保持するため構造体を宣言します。

```

flag    メニューの状態  OFF:0
                                ON :1
itype   対象要素種別番号 点   :100
                                直線:200
                                円   :300
    
```

2. 初期値を設定します。

配列は[0]から順に点、直線、円に対応します。

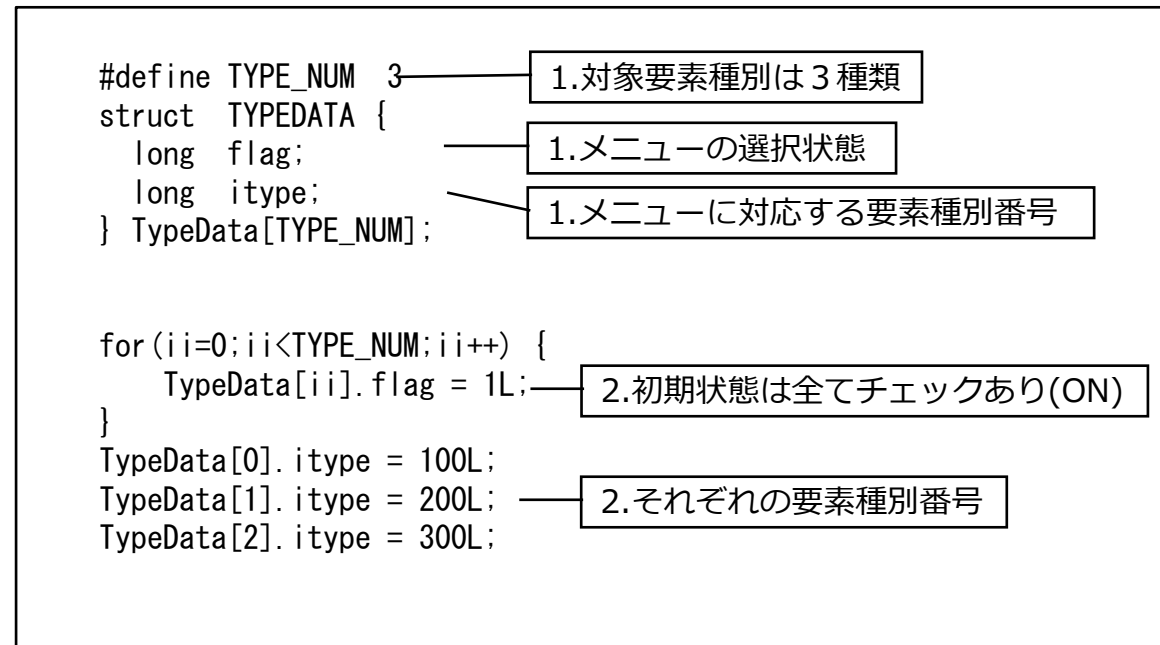
「メニュー・メッセージ作成支援ツール」で初期表示の選択状態を定義する時に点、直線、円ともに「チェックあり」としましたので、flagはそれに合わせてON(1)をセットします。

itypeはそれぞれ100,200,300をセットします。

```

#define TYPE_NUM 3
struct TYPEDATA {
    long flag;
    long itype;
} TypeData[TYPE_NUM];

for (ii=0; ii<TYPE_NUM; ii++) {
    TypeData[ii].flag = 1L;
}
TypeData[0].itype = 100L;
TypeData[1].itype = 200L;
TypeData[2].itype = 300L;
    
```



3. メニューが選択された時の処理に要素種別メニューのチェック状態がどうなっているかを取得し構造体のflagに設定する記述を追加します。
点、直線、円に関してそれぞれ判定、設定します。
4. グループ化されている要素から対象要素を抽出する部分では点のみを対象にしていたが要素種別番号を切り替えて抽出を繰り返します。
その時、3. で取得した対象種別かどうかの情報により抽出するかスキップするか判定します。
抽出した要素のポインターが現行ビューに属しているかチェックし、変数に保持する部分は既存コードのままです。
5. コンパイル・リンクしてください。
実行後、メニューのチェック状態を変更してグループ化するとハイライトされる要素が変更されることを確認してください。

```

case MENU:
    MC_menusel( &menuno, &itemno );
    if( menuno == MENU_SP8_ROOT_END && itemno == MI_END ) {
        goto exit;
    }
    else if( menuno == MENU_SP8_ROOT_PT ) {
        if ( itemno == MI_PT_ON ) TypeData[0].flag = 1L;
        else TypeData[0].flag = 0L;
    }
    :
    :

ptnum = 0L;
for( ii=0; ii<TYPE_NUM; ii++ ) {
    if( TypeData[ii].flag == 0L ) continue;
    istndx = 1L;
    itype = TypeData[ii].itype;
    iatno = 0L;
    for ( ; ; )
    {
        retc = MC_bufnd2 ( 1L, 1L, numptr, nptrar, istndx, &itype,
                        iatno, &index, &matr, 20L );

        if ( retc != 0 ) break;
        istndx = index+1;

        if ( MC_bivdt ( 1L, nptrar[index-1] ) != save_ivudet ) continue;
        ptptrar[ptnum] = nptrar[index-1];
        ptnum++;
    }
}
    
```

3. 選択されたメニューの情報取得

「終了」メニュー

3. 「□点」メニュー

3. ONが選択状態の時はチェックありなのでflagに1を設定し、対象要素種別とする
直線、円も同様に設定する

4. 要素種別は3種類なので繰り返す

4. 対象外の要素種別なので抽出はスキップ

4. 抽出する要素種別番号を指定する

4. 要素種別により要素を抽出する

1 0 . 追加対象要素の情報取得とCSV出力(1/2) 7

対象要素種別として直線、円が追加されましたのでその情報取得とCSV出力用の文字列をセットする部分を追記します。

1. MC_biitypで要素の種別を取得します。
2. それぞれの種別に応じた要素情報の取得関数を使用して点の座標を取得します。

直線 : MC_gtlnで両端点の座標を取得

始点 : xpt,ypt

終点 : xpt2,ypt2

円 : MC_gtcrcで中心点の座標を取得

中心点 : xpt,ypt

3. それぞれの要素の種類を示す文字列をセットします。
4. 要素の種類を1列目に表示できるように出力用文字列に設定します。

```
for ( ii=0; ii<ptnum; ii++ )
{
    pos = 0;
    memset ( csv_str, 0x00, sizeof(csv_str) );

    memset ( WorkBuf, 0x00, sizeof(WorkBuf) );
    itype = MC_biityp( 1L, ptptrar[ii] ); 1.要素種別を取得

    if( itype == 100L ) {
        MC_gtpt ( 1L, ptptrar[ii], &xpt, &ypt );
        strcpy( WorkBuf, "Point" ); 3.要素の種類を示す文字列を設定
    }
    else if( itype == 200L ) {
        MC_gtln ( 1L, ptptrar[ii], &xpt, &ypt, &xpt2, &ypt2 );
        strcpy( WorkBuf, "Line" ); 2.直線の両端点取得
    }
    else if( itype == 300L ) {
        MC_gtcrc ( 1L, ptptrar[ii], &xpt, &ypt, &rad, &ang1, &ang2 );
        strcpy( WorkBuf, "Circle" ); 2.円の中心点取得
    }
    else continue;

    memcpy ( &csv_str[pos], WorkBuf, strlen(WorkBuf) );
    pos += (long)strlen ( WorkBuf );

    csv_str[pos] = 0x2c; 4.要素の種類を1列目として設定
    pos++;
}
```

5. 点、直線の始点、円の中心点の座標であるxpt, yptを出力文字列にセットする部分は以前と同じです。
6. 直線の場合、終点(xpt2,ypt2)を4,5列目として出力するため処理を追記します。
7. コンパイル・リンクしてください。
実行後、メニューのチェック状態を変更して出力し対象となる要素のデータが出力されていることを確認してください。

```
Point,-29.640000,4.940000
Line,-62.847778,-23.327778,-10.154444,-19.622778
Line,-3.293333,24.837222,-29.640000,4.940000
Circle,-21.286451,2.712639
```

```
memset ( WorkBuf, 0x00, sizeof(WorkBuf) );
sprintf ( WorkBuf, "%8.6f", xpt );
memcpy ( &csv_str[pos], WorkBuf, strlen(WorkBuf) );
pos += (long)strlen ( WorkBuf );

csv_str[pos] = 0x2c;
pos++;

memset ( WorkBuf, 0x00, sizeof(WorkBuf) );
sprintf ( WorkBuf, "%8.6f", ypt );
memcpy ( &csv_str[pos], WorkBuf, strlen(WorkBuf) );
pos += (long)strlen ( WorkBuf );

if( itype == 200L ) {
    csv_str[pos] = 0x2c;
    pos++;

    memset ( WorkBuf, 0x00, sizeof(WorkBuf) );
    sprintf ( WorkBuf, "%8.6f", xpt2 );
    memcpy ( &csv_str[pos], WorkBuf, strlen(WorkBuf) );
    pos += (long)strlen ( WorkBuf );

    csv_str[pos] = 0x2c;
    pos++;

    memset ( WorkBuf, 0x00, sizeof(WorkBuf) );
    sprintf ( WorkBuf, "%8.6f", ypt2 );
    memcpy ( &csv_str[pos], WorkBuf, strlen(WorkBuf) );
    pos += (long)strlen ( WorkBuf );
}
}
```

2,3列目

6.直線の終点座標出力

4,5列目

ラバーバンド表示やドラッグ表示にはMC_draggingを使用します。処理を開始したい操作待ちの前でこの関数により『要素を作成する関数』のアドレスと必要なデータを宣言、指定します。

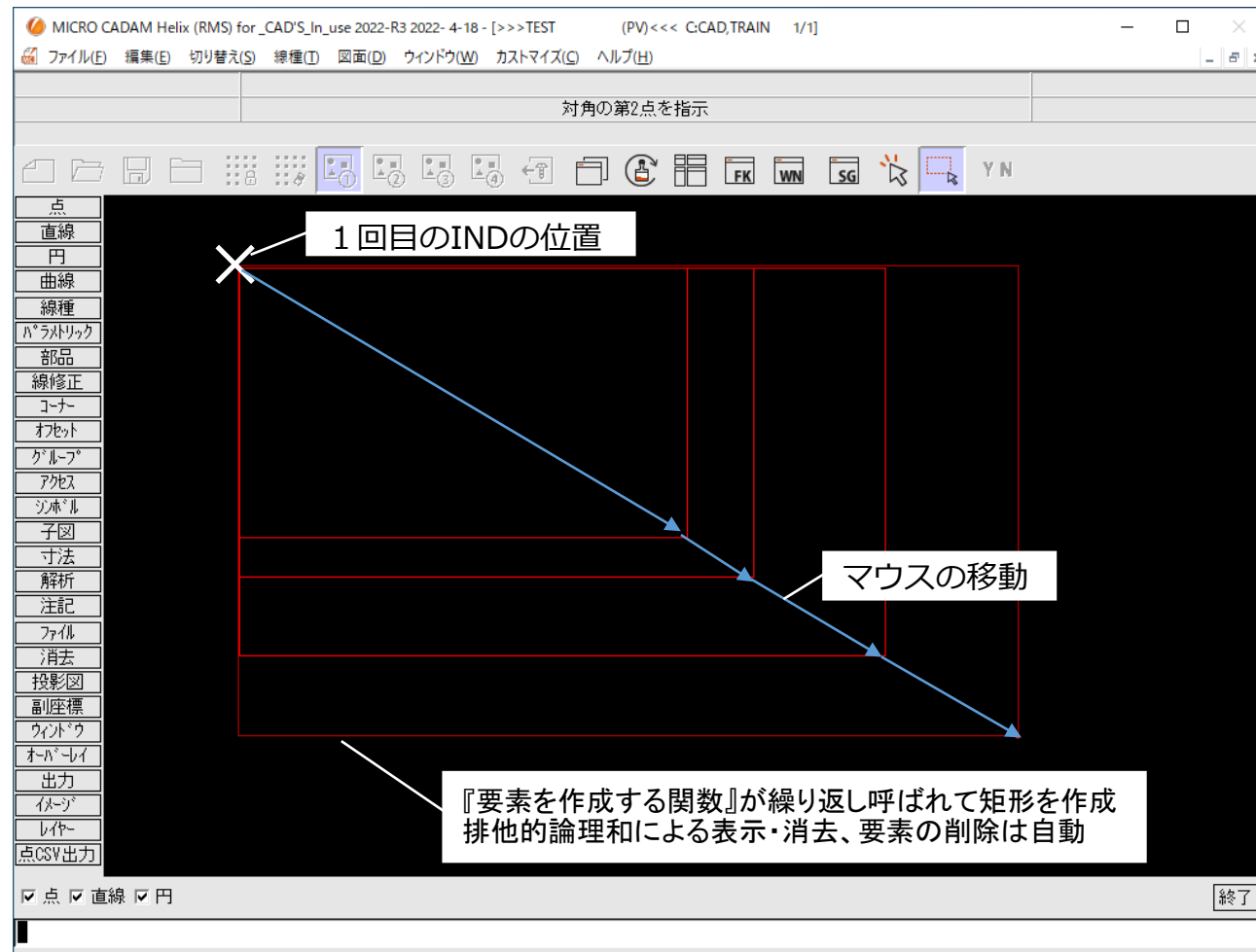
『要素を作成する関数』は内部関数として作成する必要があります。

操作待ちの関数(MC_event)内ではその情報に基づき『要素を作成する関数』を呼び出して要素を作成し、排他的論理和による表示・消去(2回同じ図形を描くとその図形は消えて元の背景に戻る)、要素の削除が自動的に繰り返されます。

作成プログラムでは範囲指定の1回目のINDの後2回目のINDのためのマウスの移動に合わせて矩形をラバーバンド表示します。

『要素を作成する関数』では1回目のINDの位置と呼び出された時点でのマウスの位置により矩形を作成します。

排他的論理和による表示・消去、削除などは自動的に行われます。



ラバーバンド表示を行う宣言をします。

1. 『要素を作成する関数』を内部関数として作成するため関数宣言します。
2番目の引数の型は受け渡すデータ型に合わせます。今回はINDの位置の座標なのでdouble型にします。
2. 2回目のINDの操作待ちの前にMC_draggingで宣言します。

カーソル移動に合わせて処理するのでオプションは"1"
『要素を作成する関数』のアドレスは関数名を指定
引き渡すデータは1回目のINDのペーパー座標
作成する要素の最大数は矩形が1個なので"1"

ラバーバンド表示の宣言をしているとMC_event内では処理が繰り返されますが、マウス移動以外の操作(例えばIND)があればMC_eventを終了し、制御が戻ってきます。MC_eventから戻ってくると今まで同様INDの位置が確定したのでIND位置を取得して実際に矩形を作成・表示します。

```

static long f_elm( long, double[], long*, long[] );
        :
while(1) {
    switch( status ) {
        case 1:
            MC_msgset( MSG_IND1 ); /* "対角の第1点を選択" */
            break;
        case 2:
            MC_msgset( MSG_IND2 ); /* "対角の第2点を選択" */

            /* Dragging */
            MC_dragging( 1L, (DRAGGING)f_elm, pxy[0], 1L );
            break;
    }
}
MC_event( mask, &event );

switch( event ) {
    case IND:
        MC_position( &vxy[status-1][0], &vxy[status-1][1],
                    &pxy[status-1][0], &pxy[status-1][1] );
        :
}
    
```

1. 『要素を作成する関数』の関数宣言

2. 『要素を作成する関数』の名前を指定

2. 1回目のINDの位置

2. 作成する要素の数は1個(矩形)

操作待ち

2. 1回目のINDの位置を保持しておく

『要素を作成する関数』を作成します。

1. 現在のマウスの位置を取得します。
 2. MC_draggingから渡された1回目のINDの位置を矩形の対角として、1. で取得したマウスの位置をもう一方の対角とします。
 3. この関数からは要素を作成したモデルの番号を返さなければならないため、カレントなモデル番号を取得します。
 4. カレントなモデルに矩形を作成するため、P.40~41で作成した矩形作成用の内部関数を呼び出します。
 5. 矩形が作成できた場合、そのポインターと作成した要素の数"1"を返します。失敗した場合"0"を返します。
 6. コンパイル・リンクしてください。
実行後、範囲指定中にマウスの移動に合わせて矩形がラバーバンド表示されることを確認してください。
- 以上でプログラムは完成です。

```
static long f_elm( long iopt,
                  double data[],
                  long *modno,
                  long mptrar[] )
{
    double xy1[2], xy2[2];
    double viewx;
    double viewy;
    double paperx;
    double papery;
    long mptr;
    long iret;

    MC_position( &viewx, &viewy, &paperx, &papery );

    xy1[0] = data[0];
    xy1[1] = data[1];

    xy2[0] = paperx;
    xy2[1] = papery;

    MC_curmdl( 1L, modno );

    iret = CreateRect ( *modno, xy1, xy2, &mptr );
    if( iret ) return( 0 );
    else {
        mptrar[0] = mptr;
        return( 1 );
    }
}
```

2番目の引数の型はMC_draggingで渡すデータの型に合わせる
それ以外は『ACCESS関数解説書』の通りに宣言する

1.現在のマウスの位置を取得する

2.MC_draggingで渡された1回目のINDの位置を矩形の対角とする

2.上で取得した現在のマウスの位置をもう一方の対角とする

4.カレントなモデル番号を取得し、要素を作成したモデル番号として返す

4.内部関数として作成済みの矩形を作成する関数を呼び出す

5.作成した要素のポインターと個数を返す



※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。