

# AJAN

# 基本マニュアル



Interface Corporation

## ■製品ドキュメントのご紹介

本製品に関する情報を下記の通りご用意しております。  
必要に応じて、適切なものをご利用ください。

マニュアル	概要
AJAN基本マニュアル(本書)	AJANの基礎知識や使い方など、AJANの全体像をご確認いただけます。まずはこちらをご覧ください。
PageGeneratorユーザーズマニュアル	Webページを視覚的かつ直感的な操作で作成できる PageGeneratorの使い方を記載しています。
AJANコマンドリファレンス	AJANに収録されているコマンドの一覧と使い方が記載されているドキュメントです。以下のカテゴリに分かれています。 <ul style="list-style-type: none"><li>・ 標準コマンド</li><li>・ 拡張コマンド</li><li>・ システム監視コマンド</li><li>・ ネットワークコマンド</li><li>・ GUIコマンド</li><li>・ 数学統計コマンド</li><li>・ エッジサーバコマンド</li><li>・ IOコマンド シリアル通信</li><li>・ IOコマンド CAN</li></ul>

無償:弊社Web site ([www.interface.co.jp](http://www.interface.co.jp))からのダウンロードは無料です。

有償:ドキュメント類の冊子(印刷物)およびソフトウェアのCD-ROMでの提供は有料販売となります。

以下に目的別の説明をご用意していますので、ご覧ください。

目的	参照先	
AJANを触ってみたい。	本書	『第2章 AJANを動かしてみよう』
AJANの基礎的な事項を確認したい。	本書	『第3章 AJANのプログラミング』
AJAN統合開発環境を使いたい。	本書	『第4章 AJAN 統合開発環境の使い方』
AJANの文法を知りたい。	本書	『第3章 AJANのプログラミング』
簡単にWebページを作りたい。	別冊	PageGeneratorユーザーズマニュアル
標準コマンドを知りたい。	別冊	コマンドリファレンス「標準コマンド編」
AJANでCANを制御するコマンドを知りたい。	別冊	コマンドリファレンス「IOコマンド CAN編」
AJANでシリアル通信するコマンドを知りたい。	別冊	コマンドリファレンス「IOコマンド シリアル通信編」
AJANでネットワークを制御するコマンドを知りたい。	別冊	コマンドリファレンス「ネットワークコマンド編」
C言語用共有ライブラリをAJANから利用する方法を知りたい。	別冊	コマンドリファレンス「拡張コマンド編Vol.2」
AJANでシステムを監視するコマンドを知りたい。	別冊	コマンドリファレンス「システム監視コマンド編」
AJANでGUIを作成・描画するコマンドを知りたい。	別冊	コマンドリファレンス「GUIコマンド編」
AJANでFFTなどの数学関数を使いたい。	別冊	コマンドリファレンス「数学統計コマンド編」
AJANでエッジサーバを制御するコマンドを知りたい。	別冊	コマンドリファレンス「エッジサーバコマンド編」
AJANで「Azure IoT Hub」を利用する方法を知りたい。	別冊	Azure IoT Hub 連携チュートリアル

# 目 次

<b>第1章 AJANって何?.....</b>	<b>8</b>
1.1 AJANとは.....	8
1.2 AJANの特長.....	8
1.3 AJANプログラミングの基本.....	9
1.3.1 コマンドとは.....	9
1.3.2 コマンドの書き方.....	9
1.3.3 プログラムとは.....	9
<b>第2章 AJANを動かしてみよう.....</b>	<b>10</b>
2.1 AJANを起動する.....	10
2.2 プログラム実行.....	11
2.3 プロジェクトの説明.....	11
2.4 サンプルプログラムで動作確認.....	12
2.5 新規プログラムの作成と実行.....	13
2.6 デバッグ機能を使ってみよう.....	15
2.7 AJANを終了する.....	18
2.8 サンプルプログラム.....	19
<b>第3章 AJANのプログラミング.....</b>	<b>20</b>
3.1 基礎知識.....	20
3.1.1 AJANはコマンド(命令)により動く.....	20
3.1.2 AJANは電卓として使える.....	22
3.1.3 AJANは数値と文字列を区別する.....	23
3.1.4 コマンドリファレンスでコマンドの説明を読む.....	24
3.1.5 AJANはエラーを検知する.....	27
3.1.6 AJANは2進数、10進数、16進数を扱える.....	29
3.1.7 ビット(bit)とバイト(byte).....	30
3.1.8 AJANは変数という箱を持つ.....	31
3.1.9 AJANは型を持つ.....	33
3.1.10 整数型と実数型.....	35
3.1.11 INPUTで変数に値を入力する.....	39
3.1.12 IF～THEN～ELSE～END IF で処理を判断・分岐する.....	40
3.1.13 インデントを使って見やすくする.....	41
3.1.14 DO WHILE～LOOP で処理を繰り返す.....	42
3.1.15 FOR～NEXT で処理を指定回数繰り返す.....	43
3.1.16 AJANは関数という魔法の杖を持つ.....	44
3.1.17 全ての文字はコード化されている.....	46
3.1.18 文字列長は文字数とバイト数の測り方がある.....	48
3.1.19 AJANはサブルーチンで共通化する.....	49
3.1.20 AJANはユーザ定義関数で独自の関数を作れる.....	51
3.1.21 AJANはコメントでプログラムに注釈を入れる.....	52
3.2 応用知識.....	53
3.2.1 文字コード 0とPRINTの因果な関係.....	53
3.2.2 文字を数値に数値を文字に.....	54
3.2.3 IF と ELSEIF を組み合わせて 3択、4択を作る.....	55
3.2.4 SELECT は、IF の亜種.....	57
3.2.5 FOR ループの中にFORループを作る.....	58
3.2.6 配列を使って多くのデータを扱う.....	59
3.2.7 実行時に大きさが変化する配列を扱う.....	60
3.2.8 配列の大きさを調べる.....	61
3.2.9 ファイルからデータを読み取って記録する.....	63
3.2.10 ファイルの読み書きの別のやり方.....	66
3.2.11 Linux外部コマンドの力を借りる.....	67
3.2.12 表計算ソフトのCSVデータを配列に読み取る.....	69
3.2.13 サブルーチン内でローカル変数を使って独立性を高めるべき.....	70
3.2.14 AJANはエラーが起きても処理を続けられる.....	73
3.2.15 アプリケーション独自のエラーあるいは異常処理を作る(1).....	74
3.2.16 アプリケーション独自のエラーあるいは異常処理を作る(2).....	76

3.2.17 AJANは割り込みができる.....	78
3.2.18 INCLUDEで他のファイルを取り込む.....	80
3.2.19 AJANはマルチスレッドで並行処理ができる.....	82
3.2.20 複数ファイルを扱う為の、グローバル変数の名前付けの工夫事例.....	84
3.3 定数 / 変数 / 式 / 演算.....	86
3.3.1 定数について.....	86
a) 文字型定数.....	87
b) 数値型定数.....	87
c) 整数型定数.....	88
d) 実数型定数.....	89
e) 論理型定数.....	90
f) 列挙型定数.....	90
3.3.2 変数について.....	91
a) 変数名.....	91
b) 変数の型.....	91
c) 論理型変数.....	92
d) 日付時刻型変数.....	93
e) 構造体型変数.....	93
3.3.3 配列、連想配列について.....	95
a) 配列.....	95
b) 範囲配列.....	96
c) 固定長の配列と可変長の配列.....	97
d) 連想配列.....	100
3.3.4 式と演算について.....	103
a) 算術演算子.....	103
b) 関係演算子.....	103
c) 論理演算子.....	104
d) シフト演算子.....	104
e) 演算の優先順位.....	105
3.4 特殊記号について.....	106
3.5 戻り値について.....	107
3.6 特殊な構文.....	108
a) 1つの文を複数行に渡って記述する.....	108
b) 1行の一部分だけをコメントにする.....	111
c) 改行を含めた文字列をそのまま記述する.....	112
3.7 制限事項.....	114
a) 配列の添え字に、更に配列の添え字を与えることは出来ない.....	114
b) 未初期化のローカル変数へのアクセスは動作不定.....	114
c) グローバル変数の宣言は、プログラムの先頭にまとめる事を推奨する.....	115
3.8 コマンドリファレンスの説明.....	116
3.8.1 コマンドリファレンスやマニュアルを表示する.....	116
<b>第4章 AJAN 統合開発環境の使い方.....</b>	<b>117</b>
4.1 AJAN統合開発環境とは.....	117
4.2 起動と終了方法.....	118
4.2.1 起動方法.....	118
4.2.2 終了方法.....	118
4.3 画面説明.....	119
4.3.1 エディタ画面.....	119
4.3.2 デバッグ画面.....	120
4.3.3 画面レイアウトの切り替え.....	121
4.3.4 各種ウインドウの開き方/閉じ方.....	122
4.3.5 メニューバー.....	123
4.3.6 機能ボタン.....	125
4.4 管理フォルダの階層構造.....	126
4.4.1 プロジェクト管理.....	127
a) プロジェクトの種類.....	127
b) プロジェクト新規作成.....	128
c) プロジェクトを閉じる.....	129
d) プロジェクトを開く.....	129
e) プロジェクトを外部公開する.....	130

4.4.2 ワークスペースの切り替え.....	132
4.5 プログラムの編集/検索/置換.....	133
4.5.1 プログラムの編集.....	133
4.5.2 自動補完機能.....	134
4.5.3 コンテンツアシスト機能.....	134
4.5.4 ポップアップメニュー.....	135
4.5.5 指定行ヘジヤンプ.....	135
4.5.6 プログラムの検索と置換.....	136
4.5.7 ブックマーク機能.....	137
4.5.8 コメントの切り替え.....	138
4.5.9 インデントガイド線.....	138
4.5.10 ソースコード保護.....	139
4.5.11 ファイル自動回復機能.....	142
4.6 ファイル操作.....	143
4.6.1 プログラムの新規作成.....	143
4.6.2 プログラムを開く.....	144
4.6.3 プログラムを保存.....	145
4.6.4 プログラムを別名で保存.....	146
4.6.5 プログラムのインポート.....	147
4.7 プログラムの実行/中断/終了/コンパイル.....	149
4.7.1 プログラムの実行.....	149
4.7.2 プログラムの中断.....	150
4.7.3 プログラムの終了.....	150
4.7.4 プログラムのコンパイル.....	150
4.7.5 コマンドライン引数を指定してプログラムを実行.....	151
4.7.6 スーパーユーザーモードでプログラムを実行.....	152
4.8 エクスプローラウインドウ.....	154
4.8.1 ポップアップメニュー.....	154
4.8.2 機能ボタン.....	155
4.8.3 ドラッグ&ドロップ.....	156
4.8.4 保存履歴と比較.....	157
4.9 ヘルプ機能.....	159
4.9.1 ヘルプウインドウ.....	159
4.9.2 キーワード検索.....	160
4.9.3 索引検索.....	161
4.9.4 ヘルプタブ.....	163
4.9.5 コマンドツールチップ機能.....	164
4.9.6 コマンドヘルプジャンプ機能.....	164
4.9.7 専用PDFビューワ.....	165
4.10 アウトライン機能.....	166
4.10.1 シンボル一覧.....	166
4.10.2 シンボル並び替え.....	167
4.10.3 フィルタメニュー.....	167
4.10.4 カーソル移動.....	167
4.11 プログラムのデバッグ.....	168
4.11.1 デバッグの実行.....	168
4.11.2 画面レイアウトの切り替え.....	168
4.11.3 ブレークポイント設定/解除.....	169
4.11.4 変数ウォッチ.....	171
a) ウォッチ変数を追加する.....	174
b) ウォッチ変数を削除する.....	175
c) 変数の値を変更する.....	175
4.11.5 トレースを実行する.....	176
4.11.1 スタックトレース機能.....	177
4.11.2 WebブラウザにURLを直接指定した場合にデバッグする方法.....	178
4.12 各種設定.....	179
4.12.1 ウィンドウの色とフォント.....	180
<b>第5章 PageGenerator.....</b>	<b>182</b>
5.1 新規プログラムの作成.....	182
5.2 PageGeneratorの起動.....	183

---

5.3 サブルーチンの編集.....	184
5.4 デバッグを行う.....	184
<b>第6章 マルチスレッドプログラム.....</b>	<b>185</b>
6.1 動かしてみよう.....	185
6.1.1 動作確認.....	185
6.2 マルチスレッドのデバッグ.....	186
6.2.1 デバッグ開始.....	186
6.2.2 デバッグ操作.....	188
<b>第7章 FAQ.....</b>	<b>190</b>
7.1 FAQ.....	190
7.2 困ったときのチェックポイント.....	191
7.3 Tips.....	193
a) Windowsのファイルを利用する場合.....	193
b) 大文字と小文字の取り扱いについて.....	193
c) 変数とメモリの関係.....	193
d) 実数型を扱う際の注意点.....	194
e) 配列変数を扱う際の注意点.....	195
f) 指定範囲の配列使用(範囲配列)指定.....	196
g) コマンドおよび関数の引数渡し時の範囲配列指定.....	197
h) AJANの情報を表示する.....	197
i) 有効桁を超えた実数演算時の注意.....	198
j) プログラムを実行するとデバイスオープン時にエラーとなる場合。.....	198
7.4 AJAN 統合開発環境を使用しないでAJANを実行する方法.....	199
a) 「端末」から実行させる方法.....	199
b) 作成したAJANプログラムを「アプリケーション」メニューに追加する方法.....	200
c) 作成したAJANプログラムを自動実行させる方法.....	202
7.5 AJAN Webサーバの停止方法.....	204
<b>第8章 重要な情報.....</b>	<b>205</b>
<b>改訂履歴.....</b>	<b>206</b>

## 第1章 AJANって何?

### 1.1 AJANとは

AJANは、「ええじゃん」と読みます。

みなさんは、計測制御や製品検査、FAとOAの連携システム等、ますます進化していくIoT等のプログラミングの課題を一挙に解決する言語に心当たりがありますか？

AJANをその問い合わせたい、そんな想いを込めたIoT用プログラミング言語です。

弊社が提供するCDシリーズやインターフェースモジュール、CoolIOsシリーズを末端のフィールドから上位クラウドのサーバまで繋ぐ架け橋をAJANが担います。

### 1.2 AJANの特長

#### ●平易なコマンド体系

AJANは、平易なコマンドの組み合わせで目的のプログラムを作ることが出来ます。

実現したい機能毎にコマンドが用意されていますので、複雑なプログラム記述をすることなく処理を実現出来ます。また、コマンド名から機能を想像できる大変読みやすいコマンドです。

#### ●必要なのはAJANだけ

AJANには、実行環境やプログラム作成に必要な機能全てが揃っています。開発、テスト、現場運用の全てを行うことができます。無駄な追加作業がなく、コストと時間を省けます。

#### ●実用性の高いコマンド

配列をそのままI/Oに出力したり、配列それぞれに演算を施したりと、今まで別にプログラムを組んで行う必要があった処理を予めコマンドとして用意しています。処理したいことが直ぐに効率よく、直感的にできます。

#### ●Webページ作成機能

AJANを使ってWebページを、視覚的かつ直感的な操作で作成できるツール(PageGenerator)を搭載しています。Webサーバー機能を搭載しており、Webブラウザから、本PCに接続すれば、ネットワーク上のどのPCからでも作成したWebページを参照出来ます。

#### ●高機能なAJAN統合開発環境を装備

AJAN統合開発環境は、「実行」ボタンを押すと自動でコンパイル等を行いプログラムが実行されますので、難しくて煩わしい操作は意識する必要はありません。

プログラミング用のエディタやデバッガが内蔵されていますので、ステップ実行や変数の内容もすぐにチェック出来ます。また、入力中のコマンドの候補を予測表示したり、コマンドの使用方法の表示や目的から最適なコマンドを逆引きできる機能を備えています。

#### ●ビジュアルなアプリケーションを簡単に作成

AJANはビジュアル的に美しい画面を簡単に作成出来ます。ボタン等のグラフィカルな部品を使ってウインドウを簡単に作成出来るので、Windowsのプログラムに引けを取らない、美しい画面インターフェースが実現出来ます。

#### ●コンパイラ方式

ソースファイルをコンパイルして実行ファイルを作成するため、高速処理が可能です。

また、実行ファイルだけで動作するため、ソースファイルを非公開にできます。

#### ●多言語連携

AJANからWeb画面などを表示したり、PythonやC言語などの他の言語を呼び出す連携システムが構築できるため、応用範囲は無限に広がります。

## 1.3 AJANプログラミングの基本

AJANからコンピュータに様々な処理を実行(命令)させるには、AJANコマンドを使います。

### 1.3.1 コマンドとは

コンピュータは人間が話す言葉を理解できないため、コンピュータが理解できる言葉(コマンド)で指示を与える必要があります。

人と話をする際、言葉をたくさん使える方が思っていることをより正確、端的に伝えることができます。

それと同じように、コマンドをたくさん使える方が、正確、端的(プログラムを短く)にコンピュータに指示を与えることができます。一部のコマンドだけでもプログラムは書けます。

そのため、最初から全てのコマンドを使える必要はありません。プログラムを書いていくにつれて、徐々に使えるコマンドは増えていきます。

### 1.3.2 コマンドの書き方

AJANは、1行ごとにコマンドを記述していきます。

各1行の記述は、コマンドの名前とコマンドに与える情報(パラメータ)からなります。

例:文字列を表示するPRINT コマンド

```
PRINT "Hello, AJAN World!"
```

↑半角スペース

コマンドの名前とコマンドに与える情報(パラメータ)の間は半角スペースを入れる必要があります。

パラメータには定数、変数の他、これらを組み合わせた式等を指定することができます。

詳細な書式は別冊の各AJAN コマンドリファレンスを参照してください。

#### ■ 基本ルール

- ・コマンドは大文字/小文字のどちらでも動作します。
- ・変数名は大文字/小文字も同じものとして扱われます。
- ・ファイルパス/ファイル名は大文字/小文字で区別されます。

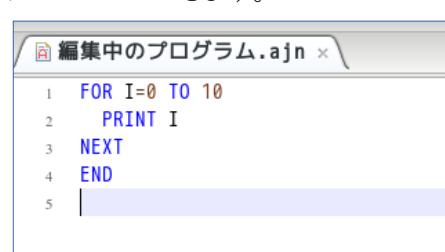
### 1.3.3 プログラムとは

プログラムとは複数のコマンドを繋げたものです。

複数のコマンドを繋げることで、コンピュータに意味のある動作を行わせることができます。

プログラムは入力した時点では実行されることではなく、

AJAN 統合開発環境から実行することで、プログラムが行番号の順番で処理されます。詳細は、『第3章 AJANのプログラミング』を参照してください。



```
編集中のプログラム.ajn x
1 FOR I=0 TO 10
2 PRINT I
3 NEXT
4 END
5 |
```

## 第2章 AJANを動かしてみよう

AJANコンパイラの統合開発環境(以降**IDE**)で基本的な使用方法を以下に記載します。

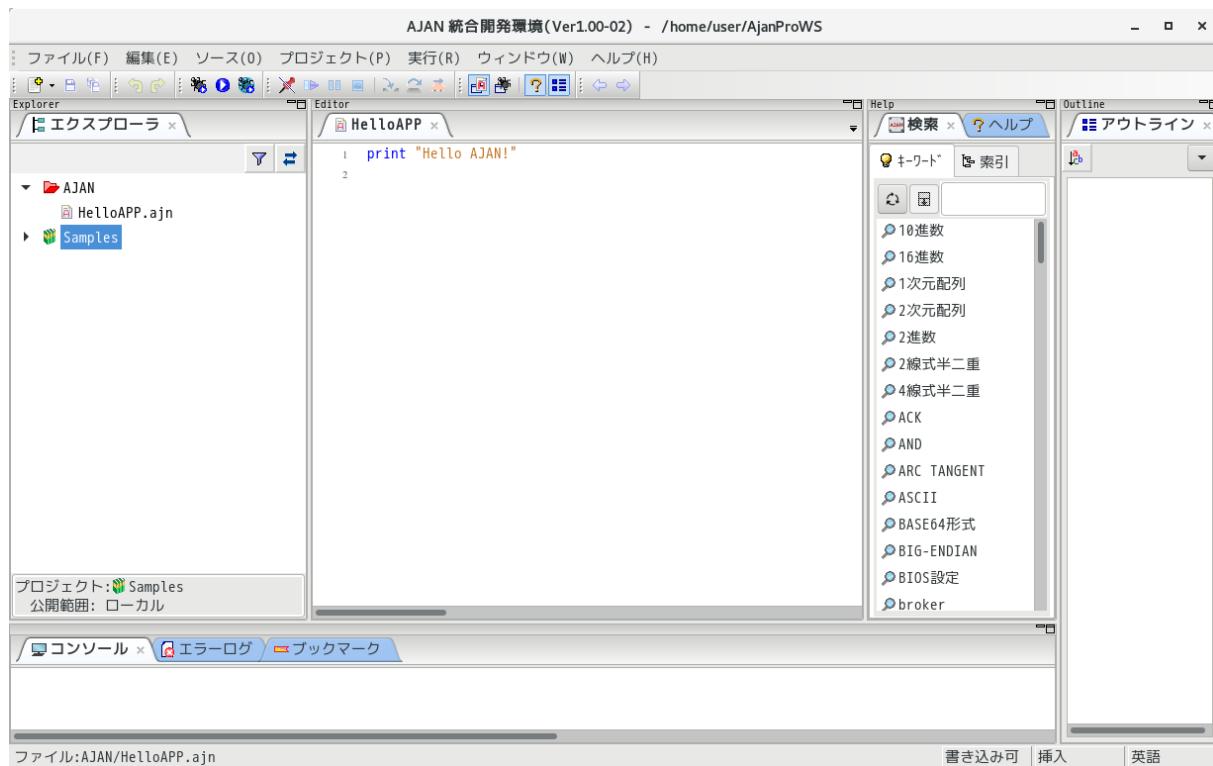
AJAN 統合開発環境の詳細な使い方は、『第4章 AJAN 統合開発環境の使い方』をご参照ください。

### 2.1 AJANを起動する

- 1) デスクトップ画面上部のメニューから「アプリケーション」 → 「Interface」 → 「AJAN」 → 「AJAN」をクリックします。



- 2) 起動が完了すると、以下の画面が立ち上がります。



## 2.2 プログラム実行

初回起動時に簡単なプログラムが記述されたテンプレートが開かれています、

手始めに、このプログラムをコンパイルして実行してみましょう。

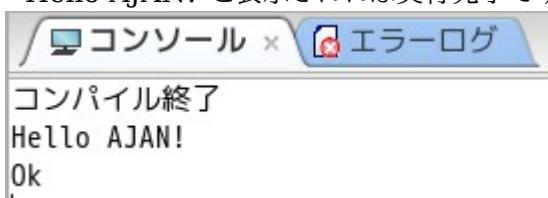


```
Editor
HelloAPP ×
1 print "Hello AJAN!"
2
```

- 1) (実行)ボタンをクリックしてください。



- 2) プログラムHelloAPP.AJNのコンパイルが行われた後に実行が開始され、コンソールウィンドウに "Hello AJAN!"と表示されれば実行完了です。



```
コンパイル終了
Hello AJAN!
Ok
```

## 2.3 プロジェクトの説明

エクスプローラーウィンドウにAJANとSamplesと表示されたプロジェクトが確認できます。

AJANプロジェクトには、先程、実行したテンプレートHelloAPP.ajnが収録されている事が確認できます。

Samplesプロジェクトには、AJANのサンプルプログラムが多数収録されています。



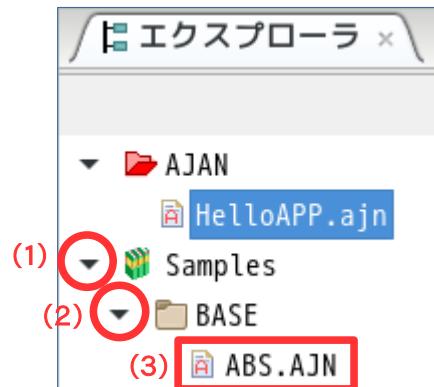
	<ul style="list-style-type: none"> <li>各プロジェクトの詳細は、『4.4.1 a) プロジェクトの種類』をご参照下さい。</li> <li>プロジェクト内のサンプルプログラムは読み込み専用(削除不可)となっています。 修正して実行したい場合は、AJANプロジェクト内にコピーして下さい。 コピー方法は、『4.8 エクスプローラーウィンドウ』をご参照下さい。</li> <li>サンプルプログラムについて、『2.8 サンプルプログラム』も併せてご参照ください。</li> </ul>
---	---

## 2.4 サンプルプログラムで動作確認

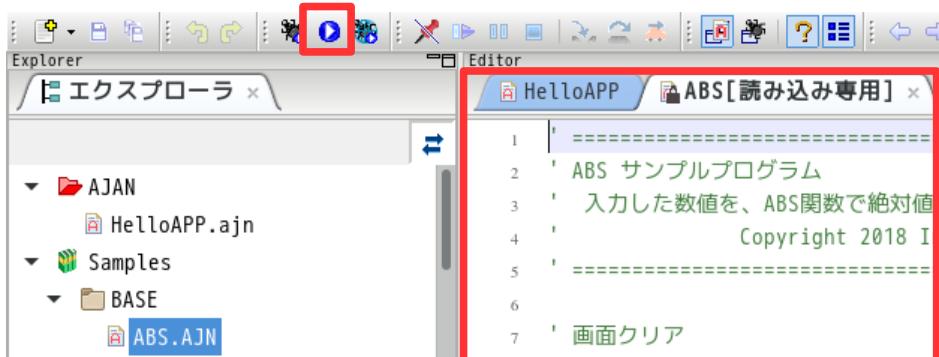
サンプルプログラムを動かしてみましょう。

入力した数値の絶対値を求めるプログラムを実行してみます。

- エクスプローラーウィンドウで「Samples」→「BASE」の順番でアイコン横の三角(▶)をクリックすると、BASEフォルダの中身が展開されます。次にABS.AJNをダブルクリックしてください。



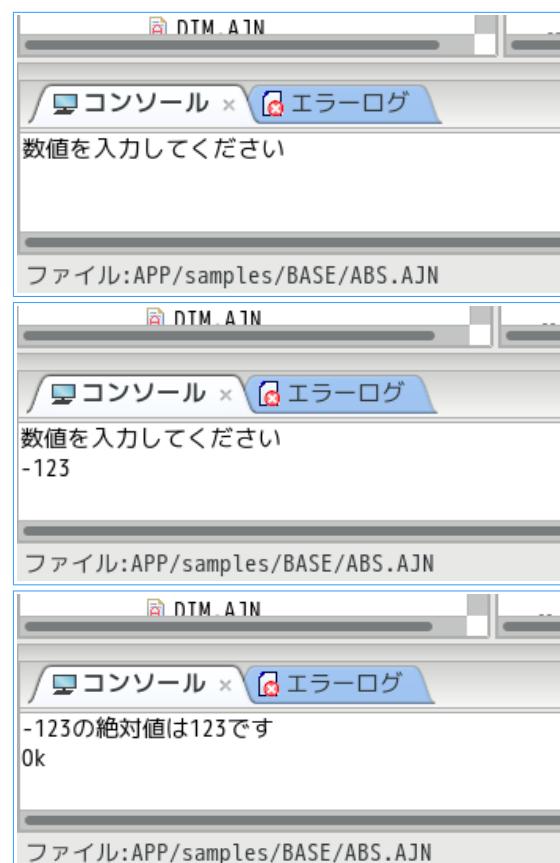
- 次のようにABS.AJNファイルが開きますので、(実行)ボタンをクリックしてください。



- プログラムABS.AJNのコンパイルが行われた後に実行が開始され、コンソールウィンドウに実行内容が表示されます。

- キーボードで数値を入力してみてください。ここでは-123と入力してみます。入力後にEnterキーを押してください。

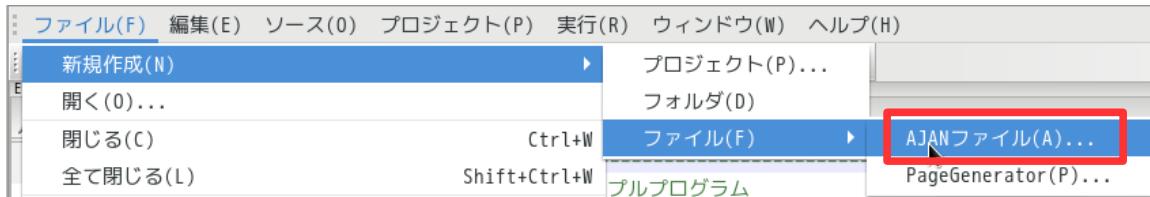
- 入力した-123に対して、絶対値が計算されて123が表示されました。



## 2.5 新規プログラムの作成と実行

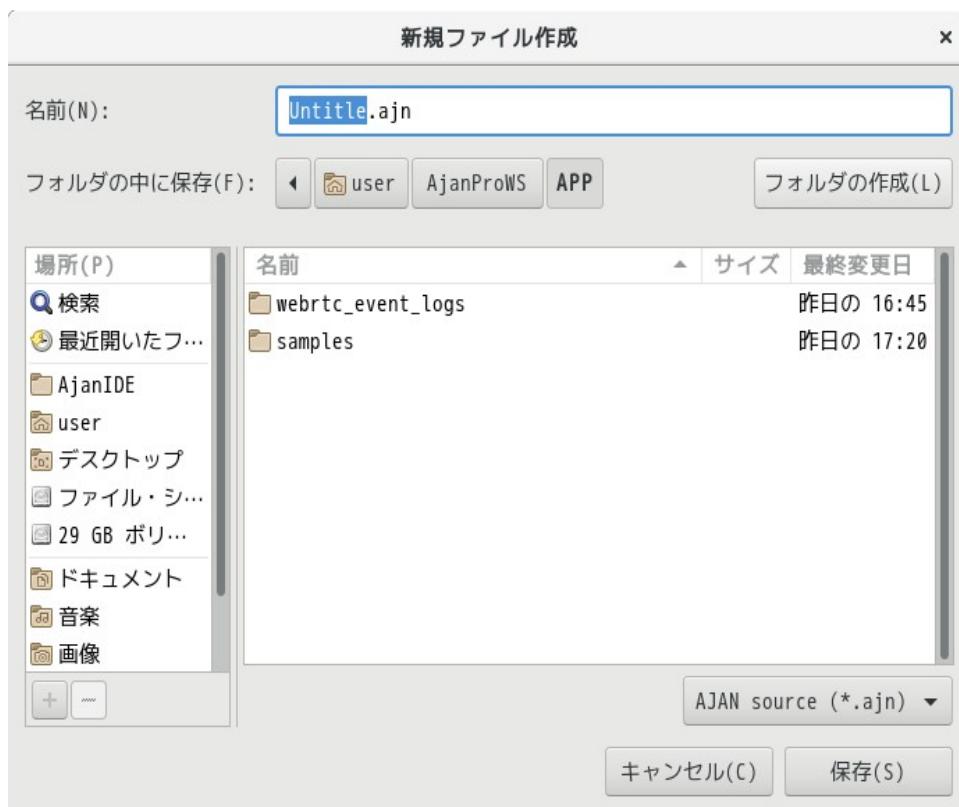
新たにプログラムを作成する場合はプログラム用のファイルを新規に作成します。

- 1) エクスプローラーウィンドウでファイルを作成するプロジェクトをクリックします。  
ここでは「AJAN」をクリックします。
- 2) 「ファイル(F)」→「新規作成(N)」→「ファイル(F)」→「AJANファイル(A)」をクリックします。



※「PageGenerator(P)...」は、Webページを作成する機能です。詳しくは、「第5章 PageGenerator」を参照して下さい。

- 3) 新規ファイルの作成を要求してきますので、デフォルトの「Untitled.ajn」のまま、もしくは任意に名前を変更して「保存(S)」ボタンをクリックします。



- 4) 以下のプログラム(7行)を、先程作成したファイルにエディタで入力してください。

```
S = CLOCK
FOR I = 0 TO 2000
    PRINT I
NEXT I
E = CLOCK
PRINT "経過時間"; E - S
END
```

経過時間を表示するプログラムです。

CLOCK : 経過時間の指標となる値を秒単位で得る関数です。  
指定した変数(ここでは S と E )に読み取った値が代入されます。

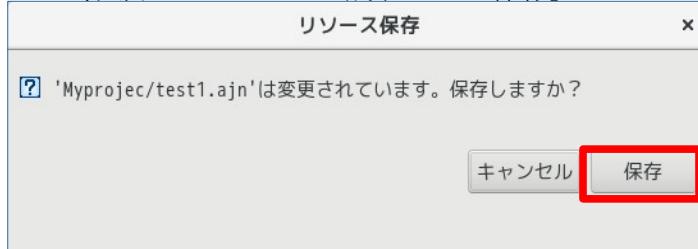
FOR ~ NEXT : FOR から NEXT までの区間中にある一連の命令を繰り返して実行します。  
指定した変数(ここでは I )に 0 ~ 2000 の値が代入され、その値を  
PRINT コマンドで繰り返し表示しています。

	<b>自動補完機能</b> キー入力に応じて、自動的にAJANコマンドの候補がリストで表示されます。この機能を自動補完といいます。 • 候補リストから入力する場合は、EnterキーかTABキーを押して下さい。 • 候補は小文字で入力すると小文字で表示され、大文字で入力すると大文字で表示されます。
---	---

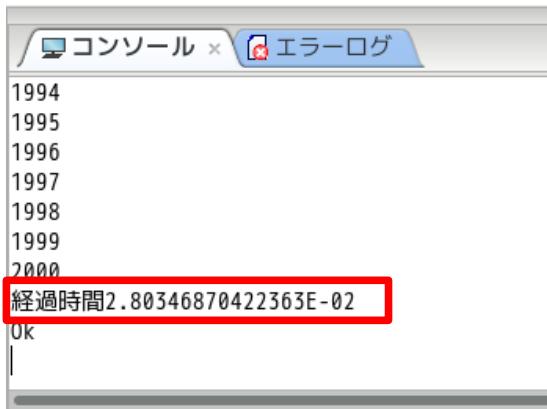
5)  (実行) ボタンをクリックしてください。



6) ここでは、入力したプログラムを残すために「保存」ボタンをクリックください。



7) コンソールウィンドウに実行結果が表示されます。プログラムが正しく動作すれば処理時間が表示されます。



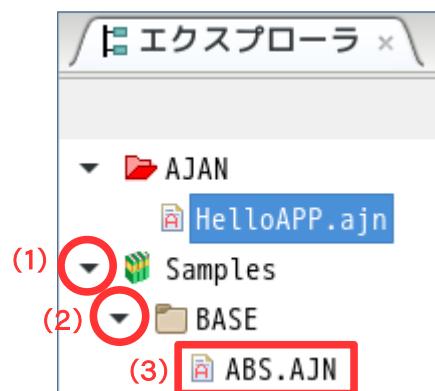
	本プログラムの実行時間は、表示が完了するまで約3秒程度掛かりますが、表示されている経過時間は約0.03秒となります。 この時間は、プログラムの実際の実行時間を表します。表示が完了するまで約3秒掛かるのは、プログラム自体は一瞬で終了しますが、IDEがバッファリングされたテキストを表示するのに時間が掛かっているためで、実行時間とは異なることにご留意下さい。
---	--

## 2.6 デバッグ機能を使ってみよう

- 1) デバッグ実行するファイルを開きます。

ここでは入力した数値の絶対値を求めるプログラムABS.AJNを実行してみましょう。

エクスプローラウィンドウで「Samples」→「BASE」の順番でアイコン横の三角(▶)をクリックすると、BASEフォルダの中身が展開されます。次にABS.AJNをダブルクリックしてください。



- 2) ブレークポイントを追加します。

ここでは14行目にブレークポイントを追加します。行番号の14の位置をダブルクリックすると画鉛のマークが表示されます。

```

4  =====
5  '入力した数値の絶対値を表示します
6  '画面クリア
7  CLS
8  FLAG=FALSE
9  DO WHILE FLAG=FALSE
10    PRINT "数値を入力してください"
11    LINE INPUT A$
12    '先頭から数値の部分だけを変換
13    '文字列で0を渡されていないのに0に変換される
14    IF VAL(A$) <> 0 OR ASC(A$) = 48 THEN
15      FLAG=TRUE
16    ELSE

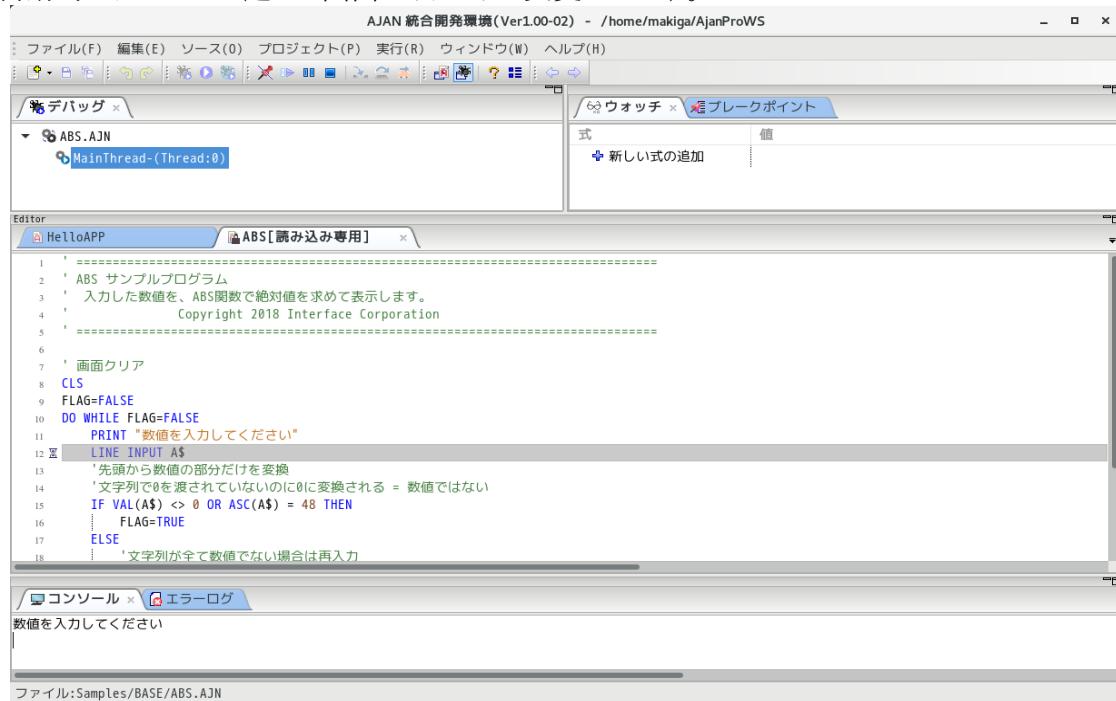
```

- 3) デバッグ実行を開始します。

(デバッグ実行)ボタンをクリックしてください。



- 4) 自動的にデバッグに適した画面レイアウトに変更されます。



- 5) 下記のようにブレークポイントに到達する前に11行目で停止しました。

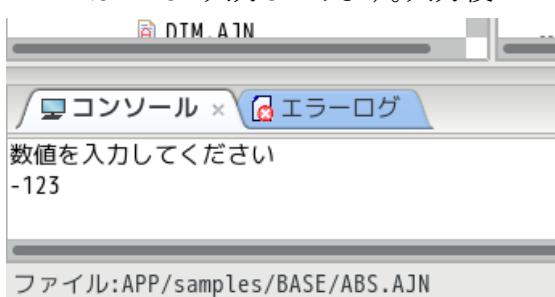
これは、LINE INPUT命令を実行しており、処理がOS側に移っている事を意味します。本命令は、キーボード入力を求めています。

```

8  FLAG=FALSE
9  DO WHILE FLAG=FALSE
10   PRINT "数値を入力してください"
11   LINE INPUT A$           ← ブレークポイント
12   '先頭から数値の部分だけを変換
13   '文字列で0を渡されていないのに0
14   IF VAL(A$) <> 0 OR ASC(A$) =

```

- 6) コンソールウィンドウにメッセージが表示されますので、キーボードで数値を入力してみてください。ここでは-123と入力してみます。入力後にEnterキーを押してください。



7) 設定したブレークポイントで停止しました。

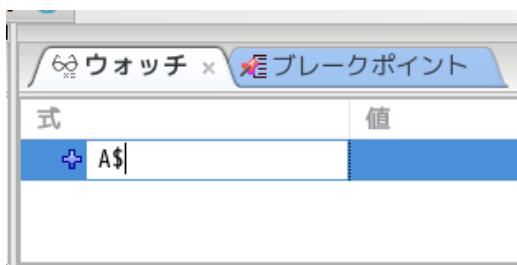
```

8  FLAG=FALSE
9  DO WHILE FLAG=FALSE
10   PRINT "数値を入力してください"
11   LINE INPUT A$
12   '先頭から数値の部分だけを変換
13   '文字列で0を渡されていないのに0に変換される = 数値
14  IF VAL(A$) <> 0 OR ASC(A$) = 48 THEN
15    FLAG=TRUE
16  ELSE

```

8) 変数の値を見てみましょう。

ウォッチウインドウの「新しい式の追加」をクリックして、A\$と入力してEnterキーを押します。



9) 変数の値が表示されました。

先程入力した、-123が代入されている事が確認出来ます。



10) プログラムを1行づつ実行してみましょう。

▶(ステップ・イン)ボタンを押す度に、1行づつプログラムが実行されます。

サブルーチン内もステップ実行されます。

```

10  PRINT "数値を入力してください"
11  LINE INPUT A$
12  '先頭から数値の部分だけを変換
13  '文字列で0を渡されていないのに0に変換される =
14  IF VAL(A$) <> 0 OR ASC(A$) = 48 THEN
15    FLAG=TRUE
16  ELSE
17    '文字列が全て数値でない場合は再入力
18    PRINT "数値ではありません"
19  END IF

```

11) ▶▶(再開)ボタンをクリックすると、次のブレークポイントに到達するか、プログラムが終了するまでプログラムが実行されます。

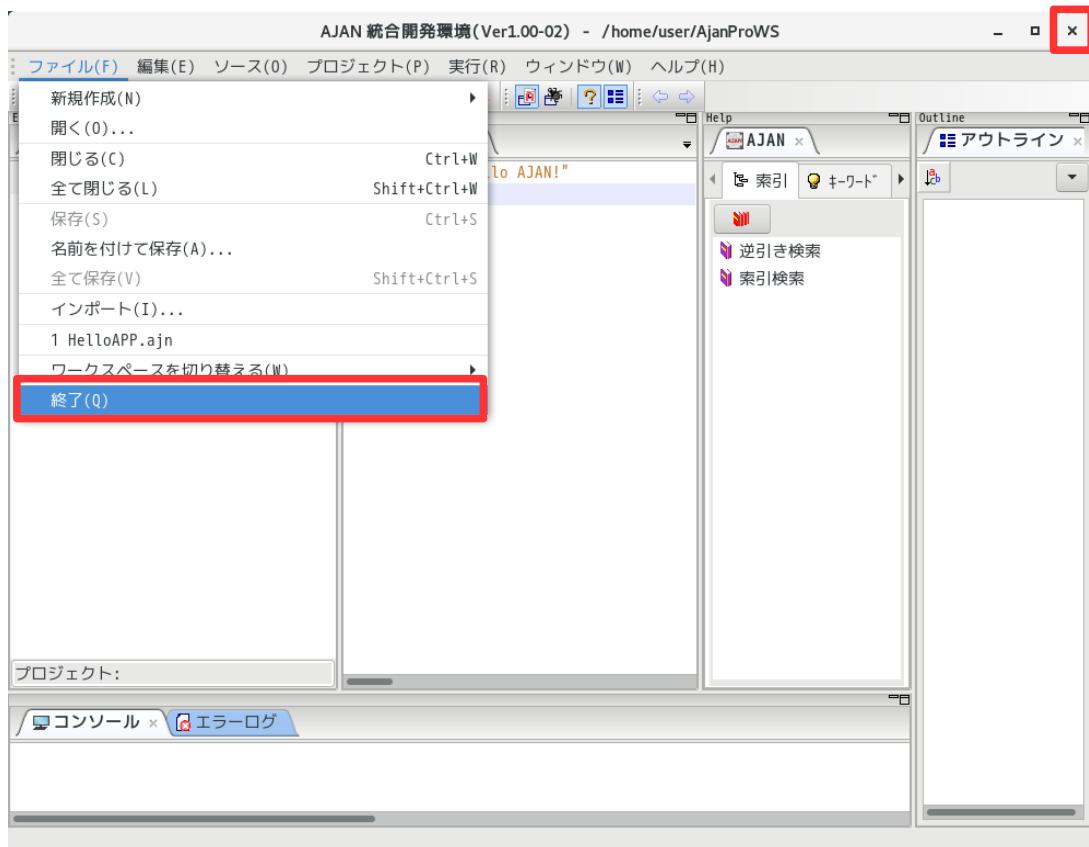
その他、プログラムの実行を強制的に終わらせる □(終了)ボタンがあります。

詳しくは、『4.11 プログラムのデバッグ』を参照して下さい。

12) プログラムの編集に適したエディタ画面に戻るには、 E(エディタ)ボタンを押して下さい。

## 2.7 AJANを終了する

AJANを終了するには、メニューの「ファイル(F)」 → 「終了(Q)」を選びます。  
右上の「×」ボタンでも終了できます。



	プログラムの保存(『4.6.3 プログラムを保存』を参照)していない場合、変更した内容は保存されません。プログラムの保存をしてから終了してください。
	AJANプログラムが実行中の場合は、強制停止して終了します。

## 2.8 サンプルプログラム

AJANでは、豊富なサンプルプログラムをご用意しています。

### 説明

サンプルプログラムは「/usr/share/interface/AJANPro/samples」以下にあります（プロジェクト内に同じものが複製されています）。

Samplesディレクトリ以下には、各カテゴリによってディレクトリが分けられています。

表2-1 サンプルプログラム一覧

コマンドの種類	コマンドリファレンス	フォルダ
表示, 入力, 変数, 演算, 文字列操作, ファイル操作, 分岐, ループ, 配列, スレッド	標準コマンド編	BASE
数学処理	数学統計コマンド 編	CAL
エッジサーバ	エッジサーバコマンド編	ESC
拡張コマンド	拡張コマンド編、拡張コマンド編Vol2	EXT
GUI	GUIコマンド編	GUI
シリアル通信	IOコマンド シリアル通信編	IML/UART
CAN / CANFD	IOコマンド CAN編	IML/CAN
ネットワーク	ネットワークコマンド 編	NWL
システム監視制御	システム監視コマンド編	SML
応用例	Azure IoT Hub連携チュートリアル	TUT
書籍サンプル	書籍「IoT言語AJAN(R)超入門」他	TUT/BOOK

サンプルプログラムの詳細は、AJANがインストールされているコンピュータの各コマンドリファレンスのサンプルプログラムの章を参照してください。

コマンドリファレンスは、デスクトップ画面上部のメニューバーから「アプリケーション」 → 「Interface」 → 「AJAN」 → 「AJANヘルプファイル」と進んで、参照したいドキュメントをクリックして下さい。

## 第3章 AJANのプログラミング

### 3.1 基礎知識

#### 3.1.1 AJANはコマンド(命令)により動く

コンピュータを使って計測器の電圧を測ったり、表にまとめた お金の計算をするとき、表計算ソフトなどのソフトウェアを使う他に、プログラミング言語を使って 自分の思うままの処理を行うことが出来ます。 AJANは、自分の思うままに処理を実現するためのプログラミング言語の一つです。

AJANは、コンピュータに対して特定の処理を行うコマンド(命令)が用意されており、これらのコマンドを組み合わせる事で、思うがままの処理を実現できます。

「2.5 新規プログラムの作成と実行」の手順を参考に、以下のプログラムを入力して実行してください。

The diagram shows the command `PRINT "Hello, AJAN World!"` enclosed in a rectangular box. Three blue curly braces point to different parts of the command: one to the word `PRINT`, another to the opening double quote, and a third to the closing double quote. Below the box, the word `コマンド` (Command) is written under the `PRINT` word, and the word `パラメータ` (Parameter) is written under the quoted text.

実行すると、コンソールウインドウに「Hello, AJAN World!」と出力されます。

ここでは「PRINT」がコンピュータに対するコマンド(命令)です。

「PRINT」コマンドと一緒に指示されたパラメータ(引数)に従って画面に出力する機能を持ちます。  
「PRINT」の後の「"Hello, AJAN World!"」という文字列が、「PRINT」コマンドに対するパラメータです。

このプログラムを実行すると、「PRINT」コマンドは「"Hello, AJAN World!"」という文字列を、画面に出力します。

!	コマンドとパラメータ(引数)の間には半角スペースを入れて、区切りをAJANに教える必要があります。
---	---

次に、プログラムを以下のように書き換えて、再び実行してください。

```
SLEEP 3  
PRINT "Hello, AJAN World!"
```

実行すると、3秒ほど経過してから、コンソールウインドウに前と同じ「Hello, AJAN World!」と表示されます。

新しく追加した行の「SLEEP」が新たなコマンドで、次の「3」という値が「SLEEP」コマンドに対するパラメータです。

「SLEEP」コマンドは、パラメータで指定された秒数、コンピュータの実行を待機する機能を持ちます。

従って、プログラムを実行すると、「SLEEP」コマンドは、パラメータ「3」で指定された秒数ほど待機してから、次のコマンドが実行されます。

このように、AJANはコマンドを組み合わせて、コンピュータに処理を行わせることができます。

### 3.1.2 AJANは電卓として使える

以下のプログラムを入力して実行してください。

```
PRINT 5 + 8
```

実行すると、コンソールウィンドウに「13」と出力されます。

前の説明で、「PRINT」コマンドは、次のパラメータを画面に出力する機能を持つ。と説明しました。  
「PRINT」コマンドは、パラメータをそのまま画面に出力するだけでなく、パラメータに書かれた式を計算して出力することができます。  
ここでは、「5 + 8」という式を計算して得られた結果の値「13」を画面に出力しています。

式を使って、電卓のように四則演算ができます。

足し算は「+」、引き算は「-」、掛け算は「\*」、割り算は「/」を記号に用います。

他にも、色々な演算が行えます。詳細は、「3.3.4 式と演算について」を参照ください。

複雑な式も計算可能です。

例えば、「3-2」を計算した後に「4」を掛ける場合、以下のように 丸括弧で囲います。

```
PRINT (3 - 2) * 4
```

算術の四則演算と同様に、AJANでは 掛け算は足し算に優先して計算しますが、複雑な式では 判りやすいよう、丸括弧を用いるよう お奨めします。

### 3.1.3 AJANは数値と文字列を区別する

「3.1.2 AJANは電卓として使える」のプログラムで、式を「PRINT」コマンドに渡すと、計算した結果を画面に出力しました。

式全体を、そのまま画面に出力したい場合は、以下のようにプログラムを書きます。

```
PRINT "5 + 8"
```

実行すると、コンソールウインドウに「5 + 8」と出力されます。

さきのプログラムとの違いは、前が「5 + 8」をパラメータとしたのに対して、今回は「"5 + 8"」をパラメータとしました。

AJANでは「」(ダブルクオーテーション)で囲った文字を「文字列」と呼びます。

対して、「5」や「8」など、「」(ダブルクオーテーション)で囲まれてない数字を、「数値」と呼びます。

「+」や「\*」などの特別な記号は、「演算子」と呼びます。

(ダブルクオーテーションで囲まれてない数字以外の文字や、その他の記号については、後述します)

上のパラメータで「"5 + 8"」は文字列ですが、先のプログラムのパラメータ「5 + 8」は、「5」と「8」が数値で、「+」が演算子で構成された「式」と呼びます。

「PRINT」コマンドは、パラメータの種類によって表示が異なります。

- パラメータが文字列の場合 文字列そのまま画面に出力します。
- パラメータが数値の場合 丸め表示されて出力されます。
- パラメータが式の場合 演算を行った結果が文字列ならば文字列として、数値ならば数値として画面に出力します。

例えば、数値同士を「+」演算子で繋ぐと、数値式として数値を足し算した結果が画面に出力されます。

以下のプログラムを実行すると、画面には「13」と出力されます。

```
PRINT 5+8
```

「+」演算子は、文字列同士にも適用できます。

例えば、文字列同士を「+」演算子で繋ぐと、文字列式として文字列を連結した結果が画面に出力されます。

以下のプログラムを実行すると、画面には「Hello,AJAN」と出力されます。

```
PRINT "Hello,"+"AJAN"
```

### 3.1.4 コマンドリファレンスでコマンドの説明を読む

「3.1.3 AJANは数値と文字列を区別する」のプログラムで、「PRINT」コマンドは、数値と文字列を画面に出力する機能を持つ事がわかりました。

「PRINT」コマンドで、他にどのような使い方ができるのか知るには、コマンドリファレンスを見ます。

コマンドリファレンスは、コマンド毎の詳細説明が書いてあり、いわば「コマンド辞書」のようなものです。

AJANのコマンドの説明を辿るには、いくつか方法があります。

#### 1. AJAN統合開発環境のヘルプウィンドウからコマンド説明を辿る

使用用途や分類からコマンド説明を辿って見たい場合は、こちらを使います。

ヘルプウィンドウ内は、ヘルプ、逆引き検索の2つのタブが用意されており、それぞれ 分類から辿るか、使用用途から探すか選択可能です。

詳しくは、「4.9 ヘルプ機能」を参照ください。

#### 2. AJAN統合開発環境のコマンドヘルプジャンプ機能からコマンド説明を辿る

あらかじめコマンド名がわかっていて、その詳細説明を読みたい場合、コマンドヘルプジャンプ機能が便利です。

エディタ上でコマンド名を入力して、カーソルをコマンド名の上に移動して、メニューから「ヘルプ」→「コマンドヘルプを開く」を選択すると、ヘルプウィンドウの該当コマンドの説明ページにジャンプします。

詳しくは、「4.9.5 コマンドツールチップ機能」「4.9.6 コマンドヘルプジャンプ機能」を参照ください。

#### 3. Linuxメニュー内のコマンドリファレンスを辿る

コマンド説明を本のようにペラペラ眺めたり、印刷したい場合は、こちらを使います。

Linuxのデスクトップから左上の「アプリケーション」メニューから、「Interface」→「AJAN」→「AJAN ヘルプファイル」の中の、コマンド種類毎のドキュメントファイルを開きます。

詳しくは、「3.8 コマンドリファレンスの説明」を参照ください。

コマンドリファレンスの「PRINT」コマンドの説明ページを開くと、下図のような説明となっています。

### 2.7.13 PRINT, ?

命令														
機能	文字列や数値等のデータを画面、またはファイルに出力します。													
書式1	PRINT <①文字列/数値> [;   , <①文字列/数値> …] [;   , ] 文字列や数値等のデータを画面に出力します。													
書式2	? <①文字列/数値> [;   , <①文字列/数値> …] [;   , ] 書式1と同じです。『?(クエスチョンマーク)』で「PRINT」の代替えとします。													
書式3	PRINT #<②ファイル番号> <①文字列/数値> [;   , <①文字列/数値> …] [;   , ] 文字列や数値等のデータをファイルに出力します。													
書式4	? #<②ファイル番号> <①文字列/数値> [;   , <①文字列/数値> …] [;   , ] 書式3と同じです。『?(クエスチョンマーク)』で「PRINT」の代替えとします。													
パラメータ	<table border="1"> <tr> <td>①</td> <td>&lt;文字列/数値&gt;</td> <td>値</td> </tr> <tr> <td colspan="3">出力する文字列/数値を指定します。セミコロン(;)で区切ると、各文字列/数値を連続して出力します。カンマ(,)で区切ると、各文字列/数値間にタブを出力します。 最後にセミコロンやカンマを指定すると、改行が起ります。</td> </tr> <tr> <td>②</td> <td>&lt;ファイル番号&gt;</td> <td>数値</td> </tr> <tr> <td colspan="3">出力先のファイル番号を指定します。 ファイル番号に変数を与えた場合、他の書式の区別に、「#&lt;変数名&gt;」の記述としてください。</td> </tr> </table>		①	<文字列/数値>	値	出力する文字列/数値を指定します。セミコロン(;)で区切ると、各文字列/数値を連続して出力します。カンマ(,)で区切ると、各文字列/数値間にタブを出力します。 最後にセミコロンやカンマを指定すると、改行が起ります。			②	<ファイル番号>	数値	出力先のファイル番号を指定します。 ファイル番号に変数を与えた場合、他の書式の区別に、「#<変数名>」の記述としてください。		
①	<文字列/数値>	値												
出力する文字列/数値を指定します。セミコロン(;)で区切ると、各文字列/数値を連続して出力します。カンマ(,)で区切ると、各文字列/数値間にタブを出力します。 最後にセミコロンやカンマを指定すると、改行が起ります。														
②	<ファイル番号>	数値												
出力先のファイル番号を指定します。 ファイル番号に変数を与えた場合、他の書式の区別に、「#<変数名>」の記述としてください。														
備考	ファイルに出力する場合、OPENコマンドでFOR OUTPUT/APPENDを指定してください。													
注意	構造体の表示はできません。													
使用例1	A = 50 PRINT "A="; A													
	A=50 と表示します。													

- 一番上はAJANコマンドの種別です。

AJANのコマンドは、値を返さない「命令」と値を返す「関数」があり、ここで「PRINT」コマンドは、値を返さない「命令」である事が判ります。

- 「機能」は、コマンドが、どのような機能を持つか説明します。
- 「書式」は、コマンドが、どのようなパラメータを持つか示します。パラメータは、「<」と「>」記号で囲まれた名前で示されています。
- 書式中の「[」と「]」で囲まれているパラメータと記号は、これを省略できる事を意味します。
- AJANコマンドの種別が「関数」の場合、「戻り値」項目があります。  
これは関数から得られる値の説明が記載されています。
- 各パラメータの詳細は「パラメータ」項目にあり、右側で、このパラメータは数値で指定するのか、文字列で指定するのか簡単な説明があります。
- 「備考」は、コマンドの機能説明の詳細事項が記載されています。
- 使用上 注意を要する事項が「注意」欄に記載されています。
- 「使用例」は、コマンドの使用事例が断片的に記載されています。

「PRINT」コマンドの説明を改めて確認すると、「,(カンマ)」および「;(セミコロン)」で繋いで、複数のパラメータを列挙できる事が記述されています。

以下のプログラムで試してみると、画面に「1 2 3」という具合に横に列挙して出力される事が判ります。

```
PRINT 1, 2, 3
```

また、別の書式で「PRINT」でなく「?」と書かれている事に気づきます。

実は、「PRINT」コマンドは「?」記号で代用する事ができます。

以下のプログラムで試してみると、前のプログラムと同じ結果が得られる事が判ります。

```
? 1, 2, 3
```

このように、コマンドリファレンスは、AJANのコマンドの詳しい使い方が書かれています。

### 3.1.5 AJANはエラーを検知する

AJANは、コマンドを使ってプログラミングする時、記述上の間違いをエラーとして報告する機能があります。

```
PRINT 1 2 3
```

上のプログラムを実行しようとすると、画面下部が「エラーログ」ウィンドウに切り替わり、以下のようなエラーメッセージが表示されます。

```
Error! &H01000002:命令が書式通りになっていません(構文エラー、予期しない integer です。予期されるのは line feed code です)[/home/user/AjanProWS/APP/Untitled.ajn:Line 1]
```

下図に画面例を示します。



エディタ上の該当箇所に赤い「×」印が表示されています。

AJANは、「PRINT」コマンドのパラメータの記述に問題があるため、コンパイルできないことを示します。

「3.1.4 コマンドリファレンスでコマンドの説明を読む」のコマンドリファレンスの説明で、「PRINT」コマンドは、パラメータを、「,(カンマ)」または「;(セミコロン)」で繋ぐとありました。

上のプログラムを見ると、各パラメータが単に空白のみで「,」または「;」で繋がれてない事が判ります。各パラメータを「,」で繋ぐと、正しく実行できる事がわかります。

```
PRINT 1, 2, 3
```

前の例では、コマンドの記述上のミスにより、エラーが発生していました。

これとは別に、記述上の問題は無いが、実行中にエラーが起きる場合もあります。

```
PRINT 10 / 0
```

上のプログラムを実行するとコンパイルはできますが、実行時に以下のようなエラーメッセージが「コンソール」ウィンドウに表示されます。

```
Error! &H0100000A:0 による 除 算 が 実 行 さ れ ま し た (0 で 除 算 し て い ま す)[/home/user/AjanProWS/APP/Untitled.ajn:Line 1]
```

0で割り算は出来ないので、例えば「10 / 2」という具合に式を修正する事で、このプログラムは最後まで実

行できます。

このように、文法上は問題なくとも、実行時にエラーが出る場合があります。

	<p>プログラムでエラーや意図しない問題を引き起こすさまを、ソフトウェアの世界では一般に「バグ」といいます。 「バグ」は単にエラーでプログラムが止まるだけでなく、意図しない挙動を起こすときにも使われます。</p> <p>例えば、CPUの温度を計測するプログラムを作るつもりが、間違って 基板の温度を計測していた。とか、 摂氏で温度表示すべきなのを、華氏で温度表示していた。などです。 また、このような「バグ」を取り除く、不具合を修正する行為を、一般に「デバッグ」と呼びます。</p>
---	---

### 3.1.6 AJANは2進数、10進数、16進数を扱える

私たちが普段数字を扱うとき、1桁は 0から9までです。これを専門用語で 10進数と呼びます。

例えば、時計では 0から59秒までいくと次は1分になるので、60進数であるといえます。

コンピュータの世界では、私たちが普段扱う 10進数とは異なる基準で数字を扱います。

すなわち、電気のオンとオフ。1と0だけの世界です。1と0で数値を表現するので、2進数と呼びます。

AJANでは、2進数で数字を表記できます。10進数との違いを見分ける為に、「&B」という文字を手前に付けます。(「&B」の「B」は、Binaryの頭文字です)

例えば、以下のように。

```
PRINT 0, &B0
PRINT 1, &B1
PRINT 2, &B10
PRINT 3, &B11
```

2進数で数字をあらわしていると冗長で長いので、まとめて扱う別の表記がよく用いられます。

コンピュータでよく使われるのは16進数です。

16進数は、0から9、A、B、C、D、E、Fまでの16個を1桁にしたものです。10進数との違いを見分ける為に、「&H」という文字を手前に付けます。(「&H」の「H」は、Hexadecimalの頭文字です)

2進数で言うと、4桁を一つにまとめる事ができ、コンピュータでメモリに情報を格納する最小単位である「バイト」との親和性が高いので、よく使われます。(1バイトは、16進数2桁で表せます)

表3-1 10進数、2進数、16進数相互変換

10進数	2進数	16進数
0	&B0000	&H0
1	&B0001	&H1
2	&B0010	&H2
3	&B0011	&H3
4	&B0100	&H4
5	&B0101	&H5
6	&B0110	&H6
7	&B0111	&H7
8	&B1000	&H8
9	&B1001	&H9
10	&B1010	&HA
11	&B1011	&HB
12	&B1100	&HC
13	&B1101	&HD
14	&B1110	&HE
15	&B1111	&HF
16	&B10000	&H10

このように、AJANは、10進数の他に、2進数、16進数を容易に扱えます。

### 3.1.7 ビット(bit)とバイト(byte)

コンピュータを扱う製品では、「ビット(bit)」と「バイト(byte)」という単語がよく使われます。

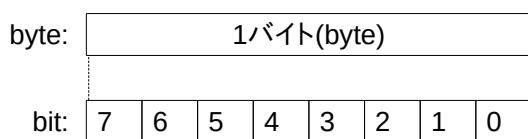
例えば、「64ビットCPU搭載」とか「256ギガバイト大容量フラッシュメモリ搭載」とかです。

「ビット」とは、コンピュータが扱う情報の最小単位です。すなわち、電気のオンとオフ。2進数の1と0に相当します。

「64ビットCPU」とは、2進数で言えば、CPUが64桁の情報を一度に扱えますよ。という意味合いで

「バイト」は、細かい定義において、コンピュータでメモリに情報を格納する最小単位が「バイト」です。

1バイトは8ビットで構成されています。1バイトは10進数では 0から255まで表せます。



1バイトが $10^3$ 集まると「キロバイト」となり、 $10^6$ 集まると「メガバイト」、 $10^9$ 集まると「ギガバイト」、 $10^{12}$ 集まると「テラバイト」と、単位が切り替わっていきます。

詳細は、コンピュータの関連書籍をご覧ください。

AJANは、コンピュータの情報を扱うため、「ビット」や「バイト」が用語として頻出しますので、ご留意ください。

### 3.1.8 AJANは変数という箱を持つ

以下のプログラムでは、「5 + 8」の式を計算して結果を画面に出力します。

この式は、数値が「5」や「8」など数字で構成されています。

```
PRINT 5 + 8
```

この数値は明らかに変化しないので、これを「定数」と言います。(数値の場合 数値型定数、文字列の場合 文字型定数と言います)

式は定数で構成されているので、なんど実行しても、同じ結果「13」が画面に出力されます。

数学で習う一次関数「 $y=ax+b$ 」のように、「x」および「y」の値を任意に当て嵌めたいとき、AJANでは「変数」を用います。

「変数」は、英文字で始まる英数文字および「\_（アンダーバー）」で組み合わせた文字です。(ダブルクオーテーションでは囲みません、囲むと文字列になります)

変数は数値や文字列を入れる一種の箱のようなものです。

この箱に値を入れるには、代入と呼ばれる書き方を用います。

```
A = 5
B = 3
PRINT A + B
```

上のプログラムは、「A」という変数の箱に「5」を、「B」という変数の箱に「3」を格納しています。これを代入といいます。

変数と演算子を組み合わせて式を構成できます。「PRINT」コマンドのパラメータ「A + B」は、変数「A」と変数「B」を「+」で足し算する数値式です。

実行すると、画面に「8」と出力されます。

文字列を代入する場合、変数名の後ろに「\$(ダラー)」記号を付けて、数値の変数と異なる事を明示します。

```
A$ = "Hello,"
B$ = "AJAN"
PRINT A$ + B$
```

上のプログラムは、「A\$」という変数の箱に「Hello,」という文字列を、「B\$」という変数の箱に「AJAN」という文字列を代入しています。

「PRINT」コマンドのパラメータ「A\$ + B\$」は、変数「A\$」と変数「B\$」を連結する文字列式です。

実行すると、画面に「Hello,AJAN」と出力されます。

代入は単に数値や文字列を代入するだけでなく、式で演算した結果を代入できます。

```
A = 5  
B = 3  
C = A + B  
PRINT C
```

上のプログラムで、「C = A + B」の箇所は、「A + B」は数値同士を足し算する式です。「C =」の部分は、「A + B」の式の結果を「C」という変数の箱に代入します。

### 3.1.9 AJANは型を持つ

AJANは、数値や文字列を記録する為に、型という箱を使って区別します。

文字型、数値型、構造体型の各型は独立しており、例えば、文字型の変数に数値型の値を直接代入する事はできません。

型にはいくつか種類があります。特に数値の場合、使用する数字の範囲に合わせて、いくつか用意されています。

用意されている型を、以下に紹介します。

表3-2 変数型一覧

変数の型			変数を宣言する方法	例	
文字型			変数名の後に「\$(ダラー)」を付けます	A\$ = "Hello"	
数値型	整数型	単精度整数型	変数名の後に「%(パーセント)」を付けます	A% = 123%	
		倍精度整数型	変数名の後に「&(アンパサンド)」を付けます	A& = 123456&	
	実数型	単精度実数型	変数名の後に「!(感嘆符)」を付けます	A! = 123.456!	
		倍精度実数型	変数名の後に「#(シャープ)」を付けるか、何も付けません	A# = 123.456# または A = 123.456	
論理型			「BOOL」コマンドで宣言します	BOOL A A = TRUE	
列挙型			「ENUM」コマンドから「END ENUM」コマンドの間で宣言します	ENUM A = 1 B = 2 END ENUM	
日付時刻型			「DATETIME」コマンドで宣言します	DATETIME A A = "2020/1/1" A = "2020/1/2 12:23:45"	
構造体型			「DEFINE STRUCT」コマンドから「END STRUCT」コマンドの間で定義し、「STRUCT」コマンドで宣言します	DEFINE STRUCT SITE ID ADDR\$ END STRUCT  STRUCT SITE A A.ID = 123 A.ADDR\$ = "Hello"	

これだけの型が用意されているのは、メモリの使用量と演算速度、表現できる値の範囲など、いくつか違いが生じる為です。

いくつか相違点を紹介します。

表3-2 各変数の特長

変数の型		型の特長	備考
文字型		"ABC"などの文字列を入れられます	"ABC"などASCII文字と呼ばれるものは1文字1バイト分のメモリを、"あいう"などの日本語は1文字数バイトのメモリが必要です。 (「3.1.17 全ての文字はコード化されている」も参照ください)
数値型	整数型	単精度整数型	-2,147,483,648～2,147,483,647の範囲を扱う整数
		倍精度整数型	-9,223,372,036,854,775,808～9,223,372,036,854,775,807の範囲を扱う整数
	実数型	単精度実数型	-3.402823E+38～3.402823E+38の範囲を扱う実数(小数点以下使用可能)。有効桁7桁。それ以上だと誤差が生じます。
		倍精度実数型	-1.79769313486232E+308～1.79769313486232E+308の範囲を扱う実数(小数点以下使用可能)。有効桁15桁。それ以上だと誤差が生じます。
	論理型	TRUE(真)とFALSE(偽)の2状態を扱う整数。	
	列挙型	「ENUM」コマンドおよび「END ENUM」コマンドの間で、複数の列挙名から選択できる整数。	
日付時刻型		日付を整数に、時刻を小数で表現した倍精度実数。	
構造体型		DEFINE STRUCT コマンドおよびEND STRUCTコマンドの間で、複数の変数を組み合わせてまとめた特殊な構造。	組み合わせる変数の型の数だけメモリが必要です。

詳細については、「3.3.1 定数について」と「3.3.2 変数について」を参照ください。

### 3.1.10 整数型と実数型

「3.1.9 AJANは型を持つ」の説明で、数値型にはいくつか種類がある事を紹介しました。

この中で、同じ数値でも「整数型」と「実数型」の区分があることに気づきます。

「整数型」は、正に「整数」を扱う数値型です。0, 1, 2... など、小数点数が付かない数値です。

「実数型」は、小数点数も扱える数値型です。123.456 など、小数点数付きの数値を扱えます。

AJANの「実数型」は「浮動小数点数」と呼ばれる数値情報の格納形式により管理されています。コンピュータの世界では、一般に「IEEE 754」と呼ばれる規格に従って「浮動小数点数」を扱います。

「浮動小数点数」で扱う実数は、私たち人間が数学で数値を扱うのに比べて、いくつか制限があります。

- 「浮動小数点数」は、非常に大きな範囲の値を小さなサイズの箱に収めて管理する為に、指数部と仮数部、符号部に数値情報を分けています。  
結果、数値に精度の限界が生じます。具体的には、単精度実数型は有効桁7桁。倍精度実数は有効桁15桁です。
- コンピュータ内部は、全ての計算を2進数で行います。  
2進数は小数点以下を表すとき、ほとんど正確に表す事ができず近似値となります。  
具体的には、10進数で「0.1」を、2進数で表そうとすると「0.00011001100...」という永遠に割り切れない値となります。(循環小数と呼ばれます)
- 「浮動小数点数」同士を計算していると、数値をうまく表現できなくなる桁あふれによる「無限大」や「不定」の結果が得られてしまう事があります。  
AJANでは「無限大」かチェックする「ISINF」関数や、「不定」かチェックする「ISNAN」関数があります。

	AJAN以外の、ほとんど全てのプログラミング言語は、「浮動小数点数」を使って実数計算を行います。
---	--

「実数型」を使って計算する場合、一般的に「定石」と言われるような注意事項がいくつかあります。

以下に、いくつか紹介します。

## (1) 実数同士で直接の等値比較を避ける

整数の場合「A = 123」という風に等値比較できますが、実数で「A = 0.1」のような等値比較は避けるべきです。

例えば、以下のようなプログラムでは、直感的には「等しい」と出力されると思います。しかし、結果は「等しくない」です。

これは、「1.0 - 0.9」を計算した際に循環小数となり、定数値 0.1と比べて、非常に小さな差異が生じるためです。

```
A# = 1.0 - 0.9      ' 直感的には、1.0 - 0.9 = 0.1のハズ
IF A# = 0.1 THEN
    PRINT "等しい"
ELSE
    PRINT "等しくない"
END IF
```

このため、実数で等値比較のような事をしたい場合、2つの値の差分を取って、その差が「プログラムを行う人が決めた誤差」の範囲内であるかどうか、で判定します。

```
EPS# = 0.00001      ' プログラムする人が決めた便宜上の誤差
A# = 1.0 - 0.9
IF ABS(A# - 0.1) < (A# * EPS#) THEN
    PRINT "等しい"
ELSE
    PRINT "等しくない"
END IF
```

「MATH ISEQ」関数を使って、以下のように簡単に書く事もできます。

```
EPS# = 0.00001      ' プログラムする人が決めた便宜上の誤差
A# = 1.0 - 0.9
IF MATH ISEQ(A#, 0.1, EPS#) = TRUE THEN
    PRINT "等しい"
ELSE
    PRINT "等しくない"
END IF
```

本説明は、「7.3」章の「実数型を扱う際の注意点」も合わせて参照ください。

## (2) 有効桁の範囲で計算する

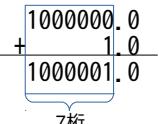
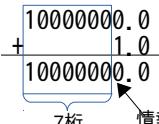
整数の場合、大きな値と小さな値を加算したとき、意図した通りの値が得られます。

実数の場合、有効桁を越えて大きな値と小さな値を加算すると、計算結果が変わらない事が起きます。(一般に「情報落ち」といいます)

```
' ↓ある程度の(有効)桁数以内での演算だと、加算できているが...
PRINT 1000000! + 1.0!      ' 1000001 と表示
' ↓ある程度の(有効)桁数以上での演算だと、加算できてない
PRINT 10000000! + 1.0!    ' 1E+07 と表示
```

上のプログラム例は、単精度実数同士を加算しています。単精度実数は有効桁7桁です。

この為、7桁を越えて加算すると、小さな値の情報が落とされてしまうのです。

精度桁内の計算時  7桁	精度桁外の計算時  7桁      情報落ち
---	---

本説明は、「7.3」章の「有効桁を超えた実数演算時の注意」も合わせて参照ください。

## (3) 0で割らない

割り算するとき、0で割ってはいけません。エラーになります。

```
PRINT 10 / 0
```

割る数が 0のとき、「IF」コマンドなどで特別な処理を行うなどして避けます。

例えば、以下のように。

```
A = 0
IF A <> 0 THEN
  ' 0以外なので普通に割り算
  B = 10 / A
ELSE
  ' 0で割ろうとしているので、特別な処理
  B = 0
END IF
PRINT B
```

(4) 実数には無限大(inf)と非数(NaN)が存在する

あまりに大きな数値を扱っていると、桁あふれ(overflow)となり、実数値が「無限大(infinity)」となります。

以下は、指指数値を求める「EXP」関数で、あまりに大きな数式を与えたために「inf」と出力されます。(「inf」は「無限大(infinity)」の略です)

<code>PRINT EXP(750)</code>	' inf と出力されます
-----------------------------	---------------

計算結果が「無限大」か否かは、「ISINF」関数で判定できます。

<code>PRINT ISINF(EXP(750))</code>	' 正の無限大なので 1 と出力されます
<code>PRINT ISINF(-EXP(750))</code>	' 負の無限大なので -1 と出力されます
<code>PRINT ISINF(EXP(1))</code>	' 無限大ではないので 0 と出力されます

実数を扱っていると、浮動小数点数として表現できないときに、実数値が「非数(not a number)」となります。

以下は、平方根を求める「SQR」関数で、負数を与えたために「nan」と出力されます。(「nan」は「非数(not a number)」の略です)

<code>PRINT SQR(-1)</code>	' -nan と出力されます
----------------------------	----------------

計算結果が「非数」か否かは、「ISNAN」関数で判定できます。

<code>PRINT ISNAN(SQR(-1))</code>	' 非数なので TRUE と出力されます
<code>PRINT ISNAN(SQR(1))</code>	' 非数ではないので FALSE と出力されます

なお、無限大同士を割り算すると、非数になります。

<code>PRINT EXP(750) / EXP(750)</code>	' -nan と出力されます
--	----------------

このように、実数は小数点数を扱えるし、大きな値を扱えるので非常に便利ですが、扱いには注意が必要です。

### 3.1.11 INPUTで変数に値を入力する

「3.1.10 整数型と実数型」で、「A# = 1.0 - 0.9」という風に書くことで、変数に値を代入できましたが、あらかじめ値が決められていました。

実行毎に、変数に任意の値を代入したいとき、「INPUT」コマンドでキーボードから値を代入する方法があります。

```
INPUT A  
INPUT B  
PRINT A + B
```

上のプログラムを実行すると、コンソールウィンドウに「?」と出力され、キーボードからの入力待ちになります。

ここで、適当な数値を入力してEnterキーを、2回ほど繰り返すと、入力した2つの数値を足し算した結果がOutputされます。

「INPUT」コマンドは、キーボードから値を入力すると、パラメータで指定した変数に代入します。

最初の「INPUT」コマンドでは、「A」変数が指定されているので、入力した数値が「A」変数に代入されます。

次の「INPUT」コマンドでは、「B」変数が指定されているので、次に入力した数値が「B」変数に代入されます。

その後、「PRINT」コマンドで、式「A + B」の演算結果が画面に出力されます。

このように、「INPUT」コマンドは、変数に数値や文字列を代入することができます。

### 3.1.12 IF～THEN～ELSE～END IF で処理を判断・分岐する

「IF」コマンドは、条件式により判断し、処理を分岐する為のコマンドです。

以下のプログラムを入力して、実行してください。

```
INPUT ONDO
IF 100 <= ONDO THEN
    PRINT "100以上"
ELSE
    PRINT "100未満"
END IF
```

「INPUT」コマンドの入力に対して、100以上の数字を入力すると「100以上」と画面に出力され、100未満の数字を入力すると「100未満」と画面に出力されます。

上のプログラムは、以下のように処理&実行されます。

1. 「INPUT」コマンドの入力した数値が「ONDO」変数に代入されます。
2. 次の「IF」コマンドのパラメータ「100 <= ONDO」を条件式として評価されます。
3. 例えば、入力した数値が「200」の場合、「ONDO」変数へ代入された値「200」は、条件式の 100 より大きいので、条件式の結果は真となります。

このとき、「THEN」以降の文「PRINT "100以上"」が実行され、画面に「100以上」と出力されます。

なお、「THEN」の処理を行う際、「ELSE」以降の文は実行されません。

4. 例えば、入力した数値が「50」の場合、「ONDO」変数へ代入された値「50」は、条件式の 100 より小さいので、条件式の結果は偽となります。

このとき、「ELSE」以降の文「PRINT "100未満"」が実行され、画面に「100未満」と出力されます。

なお、「ELSE」の処理を行う際、「THEN」以降「ELSE」までの文は実行されません。

このように処理を判断・分岐したいとき、「IF」コマンドおよび「THEN」、「ELSE」を使って処理を分けて記述します。

### 3.1.13 インデントを使って見やすくする

「3.1.12 IF～THEN～ELSE～END IF で処理を判断・分岐する」で紹介したプログラムは、何箇所かインデント(字下げ)をしていました。

以下に、再掲します。

```
INPUT ONDO
IF 100 <= ONDO THEN
    PRINT "100以上"
ELSE
    PRINT "100未満"
END IF
```

インデントは、プログラムを読む人が、プログラムの流れが判りやすいうように使います。

上のプログラムの例で言えば、「THEN」の処理を行う際、次の「PRINT」が実行されて、「ELSE」まで処理が終わります。また、「ELSE」の処理を行う際、次の「PRINT」が実行されて、「END IF」まで処理が終わります。

仮に、インデントをなくすと、「THEN」の処理や「ELSE」の処理で、どこからどこまでが実行されるのか、ぱっと見て判りにくくなります。

```
INPUT ONDO
IF 100 <= ONDO THEN
PRINT "100以上"
ELSE
PRINT "100未満"
END IF
```

インデント自体は、コンピュータにとって、プログラムの解釈・実行に際して意味あるものではありませんが、人間にとっては、インデントのあるなしで、プログラムの読みやすさが格段に異なります。

プログラムを書く際、詰めてプログラムを書かずに、インデントを付けて、より判りやすいプログラムを書くよう心がけてください。

### 3.1.14 DO WHILE～LOOP で処理を繰り返す

「DO WHILE」コマンドおよび「LOOP」コマンドは、条件を満たす間、処理を繰り返すコマンドです。

以下のプログラムを入力して、実行してください。

```
KANE = 10000
Y = 0
DO WHILE Y < 10
    KANE = KANE * 1.03
    Y = Y + 1
    PRINT Y, KANE
LOOP
```

実行すると、「1 10300」、「2 10927.27」、…「10 13439.1637...」という風に画面に出力されます。

上のプログラムは、以下のように処理&実行されます。

1. 最初の行で「KANE」変数に 10000 を、次の行で「Y」変数に 0 を代入します。
2. 次の行「DO WHILE」コマンドのパラメータ「Y < 10」の条件式で、「Y」変数が 10 未満である事を確認します。
3. ループの最初は、条件を満たしているので、次の行「KANE = KANE \* 1.03」で「KANE」変数に 1.03 を掛け算し、更に次の行「Y = Y + 1」で「Y」変数に 1 を足します。
4. そして次の行の「PRINT Y, KANE」で、「Y」変数と「KANE」変数の値を画面に出力します。
5. 次の行「LOOP」コマンドに達すると、「DO WHILE」コマンドの行に処理がジャンプします。
6. 再び「Y < 10」の条件式を満たすか判断します。
7. これらの処理を 10 回ほど繰り返すと、「DO WHILE」コマンドの「Y < 10」の条件式判定時、「Y」変数の値が 10 になり、条件を満たさない時が来ます。
8. 条件を満たさないと、「LOOP」コマンドの次の行に抜けて、処理が終わります。

このように、同じ処理を条件式を満たす間 繰り返したい場合、「DO WHILE」コマンドおよび「LOOP」コマンドの間に、繰り返したい処理を書きます。

### 3.1.15 FOR～NEXT で処理を指定回数繰り返す

「FOR」コマンドおよび「NEXT」コマンドは、処理を一定回数繰り返す為のコマンドです。

以下のプログラムを入力して、実行してください。

```
FOR N=1 TO 10 STEP 1
    PRINT N
NEXT N
```

実行すると、「1」「2」…「10」の順に画面に出力されます。

上のプログラムは、以下のように処理&実行されます。

1. 「FOR」コマンドのパラメータ「N=1」で、「N」変数に1をまず代入します。
2. 次の行「PRINT N」で、「N」変数の値を画面に出力します。  
「N」変数は 1 なので、画面には「1」が出力されます。
3. 次の行「NEXT N」で、「FOR」コマンドの行に 処理がジャンプします。  
ここで「N」変数の値が 1 足されて 2 になります。  
「TO」の次のパラメータが「10」になってないので、再び次の行「PRINT N」を実行します。
4. 次の行「NEXT N」で、再び「FOR」コマンドの行に 処理がジャンプし、「N」変数の値が「STEP」の  
次に指定された値分を足されます。ここでは、1 足されて 3 になります。  
(「STEP」は省略可能です。省略すると、1足すとなります)
5. これらの処理を10回繰り返すと、「FOR」コマンドの行に 処理がジャンプした時、「N」変数の値が  
10 になります。  
「TO」の次のパラメータが「10」になったので、「NEXT N」の次の行に抜けて、処理が終わります。

このように、同じ処理を指定回数 繰り返したい場合、「FOR」コマンドおよび「NEXT」コマンドの間に、繰り  
返したい処理を書きます。

### 3.1.16 AJANは関数という魔法の杖を持つ

数学の世界では、三角関数など、ある値を入れると従属して一つの値が得られる「関数」という概念があります。

AJANでも「関数」があります。一つないしは複数のパラメータ(無しの場合も)を渡すと、従属して異なる値が得られたり、あるいは何か動作を行った結果を得られます。

ただし、数学の「関数」と異なり、AJANの「関数」は、後述するように様々な機能を持ちます。

AJANの「命令」は値が返ってきませんが、「関数」は値が返ってきます。

例えば、正弦(サイン)の値を得られる SIN関数の呼び出し例を、以下に示します。

```
PI= 3.14159265358979
PRINT SIN(30 * PI / 180)      ' 実行すると、「0.5」と表示されます
```

AJANには、正弦(SIN関数)、余弦(COS関数)、正接(TAN関数)など、さまざまな数学関数が用意されています。

AJANの関数は数学に関するものだけではありません。文字列に関するものもあります。

例えば、文字列に対して任意の箇所から文字列を取り出す事ができる「MID\$」関数の呼び出し例を、以下に示します。

このMID\$関数では、複数のパラメータを指定することに注目してください。

```
A$ = "Interface"
PRINT MID$(A$, 2, 3)      ' 実行すると、「nte」と表示されます。
```

また、日付を得る「DATE\$」関数や、乱数を得る「RND」関数など、パラメータなしに呼び出せる関数もあります。

```
PRINT "今日の日付:"; DATE$
PRINT "乱数:"; RND
```

また、システムの空きメモリを得られる「MEMINFO」関数や、ファイル一覧を得られる「FILELIST\$」関数など、OSの内部情報やハードウェアに関する情報を得られる関数もあります。

```
PRINT "システム全体の空きメモリ量:"; MEMINFO(0)
PRINT "実行フォルダのファイル一覧:"; FILELIST$("*")
```

AJANの「命令」と異なる特徴として、AJANの「関数」は式の中に混ぜて使用する事ができます。

```
FOR I=0 TO 3.14 STEP 0.05
    Y = SIN(I) * COS(I) * 100 + 50
    X = SIN(I) * SIN(I) * 100
    PRINT X, Y
NEXT I
```

このようにAJANの「関数」は、さまざまな機能のものが用意されています。

### 3.1.17 全ての文字はコード化されている

AJANで「PRINT "ABC"」を実行すると画面に「ABC」と出力されます。

何の気なしに「ABC」とよんでいますが、コンピュータの中では「ABC」という文字情報を記憶・管理するために、1文字ずつ文字コードと呼ばれる数値を割り当てています。

以下のプログラムを実行してみてください。画面に「ABC」と出力されます。

```
PRINT CHR$(65)+CHR$(66)+CHR$(67)
```

AJANで文字コードから文字を得る関数として「CHR\$」関数があります。

上のプログラムは、文字コード 65、文字コード 66、文字コード 67 を、それぞれ文字に置き換えて画面に出力します。コンピュータの中では、文字コードがそれぞれ「A(文字コード65)」、「B(文字コード66)」、「C(文字コード67)」に割り当てられているので、「ABC」と画面に出力されるのです。

文字コードと各文字の割り当てには規則があり、英数字および各種記号については、今のコンピュータ一般では「アスキーコード(ASCII)」と呼ばれる割り当てが使用されています。

詳しくは、コマンドリファレンス「標準コマンド編」の「第5章 アスキーコード一覧」を参照ください。

英数字はアスキーコードが使用されていますが、日本語は別の規則に基づいて、文字毎に文字コードが割り当てられています。

AJANが動作するコンピュータで使用されているのは、「ユニコード(Unicode)」と呼ばれる規則が使用されています。

ユニコードは、より大きな値の文字コードが割り当てられるようになっており、0から111万4111まで割り当て可能です。(今でも世界中の文字を割り当てる作業が続けられています)

例えば、「夢」という文字を出力するには、以下のように書けます。

```
PRINT CHR$(22818)      ' 22818がユニコード値です
```

ある文字の文字コード(ユニコード値)を調べるには、「ASC」関数を使用します。

例えば、「あ」という文字の文字コードを調べるには、以下のように書けます。

```
PRINT ASC("あ")      ' 実行すると「12354」が画面に表示されます
```

ユニコードは大きな値を使用する為、コンピュータに記録する際に、定められた方法で情報を格納します。定められた方法を文字符串化形式と呼び、AJANが動作するコンピュータで使用されているのは「UTF-8」と呼ばれる文字符串化形式です。

文字符串化形式はさまざまな種類があります。

WindowsというOSは、Shift\_JIS(シフトJIS)が一般によく使われていますが、内部では ユニコードの文

字符串化形式の一つである UTF-16 が使われていました。Windows 10 のある時期からは、ノートパソコンというソフトのデフォルト保存形式が UTF-8 になりました。

LinuxというOSも、AJANが動作する環境では UTF-8 が採用されていますが、かつては EUC と呼ばれる文字符串化形式が使われていた事があります。

Windowsからテキストデータの入ったファイルを持ってきて、Linuxで開くと 意味不明の文字の羅列になるのは、異なる文字符串化形式で表示しようとするからです。これを一般に「文字化け」と呼びます。

「文字化け」を避けるには、相互のOS間で使用する文字符串化形式を揃えます。

テキストデータを扱うテキストエディタの中には、好みの文字符串化形式でテキストデータを保存・開くことができるものがあるので、これを使用します。

あるいは、文字符串化形式を変換するツールがあります。Linuxで代表的なのは「nkf」や「iconv」と呼ばれる文字変換ツールです。

以下に、Shift\_JISで作られた「sjis.txt」ファイルを、UTF-8に変換して「utf8.txt」ファイルに変換する例を示します。(この呼び出しは、Linuxの端末上で行います)

```
$ iconv -f Shift_JIS -t UTF-8 sjis.txt > utf8.txt
```

	iconvの詳しい使い方は、「man iconv」で確認ください。 (「man」コマンドは、指定したコマンドの使い方を表示するコマンドです)
---	---

上では「iconv」ツールを使用する例を紹介していますが、「7.3」章の「Windowsのファイルを利用する場合」では、「nkf」ツールを使った事例を紹介しています。参考にしてください。

### 3.1.18 文字列長は文字数とバイト数の測り方がある

AJANには、文字列の長さを測るための2種類の関数があります。

LEN	文字列の長さを文字数で返します
LENB	文字列の長さをバイト数で返します

試しに「abcあいう」という文字列の長さを、「LEN」関数と「LENB」関数を使って調べてみます。

PRINT "LEN="; LEN("abcあいう")	' 6と表示される
PRINT "LENB="; LENB("abcあいう")	' 12と表示される

実行すると、それぞれの結果が異なるハズです。

このように結果が異なるのは、関数ごとに長さの測り方の違いがあるからです。

「LEN」関数の結果が「6」というのはわかりやすいです。「abcあいう」は6文字で構成されているからです。

「LENB」関数の結果が「12」というのは、「abcあいう」という文字列がコンピュータの中で記録される際に、どの程度メモリを占有するか、「バイト」という単位で見ているからです。

「バイト(byte)」はコンピュータが扱う単位の一つで、メモリに格納するデータは「バイト」単位で格納されます。

1バイトは、0から255までの範囲の値を扱えます。

前の説明で、「ユニコード」は 0から111万4111 までを文字コードに割り当てて扱う。と説明しましたが、これだけの大きな値では、1バイトに収まりません。なので、実際には「あ」の日本語文字は、実は 3バイトでメモリに格納しているのです。

あ		
&HE3	&H81	&H82

3バイト

という訳で、「abcあいう」という文字列がメモリに実際に格納されている長さは、「a(1バイト)」「b(1バイト)」「c(1バイト)」「あ(3バイト)」「い(3バイト)」「う(3バイト)」で、合計12バイトという訳です。

!	バイト単位で文字列を扱う時、「バイトデータ」列や「バイナリ文字列」と表現する事があります。
---	---

### 3.1.19 AJANはサブルーチンで共通化する

AJANでプログラムを書いていると、パラメータが若干変化するだけで、似たような処理が連続するケースが出ます。

こんなとき、サブルーチン化することで、似たような処理を一つにまとめて共通化すると良いです。

```

SUB DECO_PRT(MSG$)
    PRINT MSG$
    CNT = LEN(MSG$)
    D$ = ""
    FOR I=0 TO CNT-1
        D$ = D$ + "="
    NEXT I
    PRINT D$
END SUB

CALL DECO_PRT("Hello AJAN")

PRINT "サブルーチンは、SUBとEND SUBの間で"
PRINT "記述した処理を呼び出す事ができます"
PRINT ""

CALL DECO_PRT("Thanks")

```

上のプログラムは、画面に出力するメッセージの一部で、次行に「=」を文字数分ならべて装飾する処理を、サブルーチン化しています。

サブルーチンの処理は、「SUB」コマンドから「END SUB」コマンドの間の処理です。

「SUB」コマンドの次の名前の「DECO\_PRT」が、サブルーチン名です。サブルーチンを呼び出すとき、このサブルーチン名を指定して呼び出すことができます。

丸括弧の中の変数名は、サブルーチンを呼び出す際にパラメータを渡してやることで、共通化処理の変化する部分を記述できます。

サブルーチンを呼び出しているのは、「CALL」コマンドで、パラメータに「DECO\_PRT」と書いてあり、これは「DECO\_PRT」のサブルーチンを呼び出してください。という指示です。

「CALL DECO\_PRT("Hello AJAN")」でサブルーチンを呼び出すと、「Hello AJAN」を画面に出力した後、次行に同じ文字数だけ「=」を出力します。

その後の「CALL DECO\_PRT("Thanks")」でサブルーチンを呼び出すと、「Thanks」を画面に出力した後、次行に同じ文字数だけ「=」を出力します。

このように、サブルーチンを用いることで、処理の共通化をおこなうことができます。

### 3.1.20 AJANはユーザ定義関数で独自の関数を作れる

先の説明では、サブルーチンを使って処理の共通化することができましたが、これは処理を呼び出すだけで、値は返ってきませんでした。

呼び出すと、自分用の処理を行って、何かしらの値を返ってくるようにしたい場合、ユーザ定義関数を用います。

```
FUNCTION HERON(A, B, C)
    D = (A + B + C) / 2
    HERON = SQR(D * (D - A) * (D - B) * (D - C))
END FUNCTION

PRINT HERON(5, 6, 7)
```

上のプログラムは、3辺から三角形の面積を求めるヘロンの公式を、ユーザー定義関数で定義して呼び出しています。

ユーザ定義関数の処理は、「FUNCTION」コマンドから「END FUNCTION」コマンドの間の処理です。「FUNCTION」コマンドの次の名前の「HERON」が、ユーザ定義関数名です。ユーザ定義関数を呼び出すとき、この関数名を指定して呼び出すことができます。

丸括弧の中の変数名は、ユーザ定義関数を呼び出す際にパラメータを渡してやることで、関数内処理の計算に用いる部分を記述できます。

関数内処理中に ユーザ定義関数名に値を代入すると、ユーザ定義関数の戻り値を設定できます。

ユーザ定義関数は、他の関数と同じように呼び出すことができ、「PRINT」コマンドの式中の「HERON」という名前の後に、パラメータを記述することで、定義した「HERON」関数を呼び出すことができます。

このように、ユーザ定義関数を用いることで、独自の関数を作ることができます。

### 3.1.21 AJANはコメントでプログラムに注釈を入れる

プログラムを書いていると、主に英文字で構成されたコマンドの呼び出しと、式と関数の組み合わせが続いていきます。

このようなプログラムを書いていると、暫くたって見直したとき、「あれ?ここは何の処理をするんだったつけ?」とか「この変数名、何に使うんだっけ?」というように、判らなくなることがあります。

こういうときに、プログラム中に注釈が入れられると便利です。

AJANは、「'(シングルクオーテーション)」記号または「REM」コマンドの後に、自由なコメント(注釈)を入れる事ができます。

```
' ヘロンの公式により、A, B, Cの各辺から、三角形の面積を求めます
FUNCTION HERON(A, B, C)
    D = (A + B + C) / 2
    HERON = SQR(D * (D - A) * (D - B) * (D - C))
END FUNCTION

PRINT HERON(5, 6, 7)
```

上のプログラムでは、ユーザ定義関数にコメントを付けて、どんな機能を持つのか判り易いようにしています。

このように、コメントを用いると、人間にとってプログラムがより判りやすくすることができます。

## 3.2 應用知識

### 3.2.1 文字コード 0とPRINTの因果な関係

「PRINT」コマンドに文字列をパラメータに渡すと、指定した文字列を画面に出力します。

例えば、以下のプログラムを動かすと、どのような出力結果になるでしょうか。

```
PRINT "Hello"+CHR$(0)+"World"
```

「Hello World」と表示されるかと思いきや、「Hello」としか表示されません。

どうなっているのか手がかりとして、「"Hello"+CHR\$(0)+"World"」という文字列が、実際にどのようにメモリに格納されているのか調べてみます。これには「STR\_DUMP\$」関数が最適です。

「STR\_DUMP\$」関数は、文字列がどのようにメモリに格納されるのか、16進数ダンプと一般に呼ばれる形式の文字列に変換します。16進数ダンプすると、文字列のメモリのバイト単位の情報がわかり易い形式で得られます。

```
PRINT STR_DUMP$("Hello"+CHR$(0)+"World")
```

上のプログラムを実行すると、例えば以下のような出力結果が得られます。

```
dump addr:0x55ce866ee6f0~0x55ce866ee6fa size:11
+0 +1 +2 +3 +4 +5 +6 +7 - +8 +9 +A +B +C +D +E +F 0123456789ABCDEF
-----+-----+
0x000055ce866ee6f0 : 68 65 6c 6c 6f 00 77 6f - 72 6c 64 : hello.world
```

出力結果を見ると、+5の個所の値が 00 となっています。これは文字コード 0 を意味します。

文字コード 0は、「A」や「B」などの文字に割り当てられていません。

「PRINT」コマンドで文字列を画面に出力するとき、「PRINT」コマンドは「文字として表記可能な部分のみ」画面に出力します。

このため、「PRINT」コマンドは、「Hello」まで画面に出力した後、それ以降の出力を打ち切ってしまいます。

結果、画面には「Hello」としか表示されないです。



文字コード 0(CHR\$(0)のこと)は、AJANやその他のプログラム言語で特別な扱いをされています。  
例えば、C言語と呼ばれるプログラム言語は、コード 0は、文字列の終端用の特殊記号として扱われています。

### 3.2.2 文字を数値に数値を文字に

AJANは、文字列を数値に変換したり、数値を文字列に変換するための関数があります。

これは、ファイルから文字列を読み出した後、中の数字部分を取り出して数値に変換したり、数値を右寄せに揃えて出力したい。などに用います。

VAL	数文字で構成された文字列を数値に変換します。
STR\$	数値を数文字で構成された文字列に変換します

```
A$ = STR$(12345) ' 数値→文字列へ
V  = VAL(A$)      ' 文字列→数値へ
```

他に、10進数の数値を2進数の文字列、16進数の文字列に変換する関数があります。

BIN\$	数値を2進数の文字列に変換します
HEX\$	数値を16進数の文字列に変換します

2進数や16進数に変換された文字列を、元の数値に変換するには、以下のように「VAL」関数を使用します。

```
B$ = BIN$(123)
PRINT B$, VAL("&B" + B$)
H$ = HEX$(123)
PRINT H$, VAL("&H" + H$)
```

このように、関数を使って、数値と文字列の相互変換が簡単に行えます。

### 3.2.3 IF と ELSEIF を組み合わせて 3択、4択を作る

「3.1.12 IF～THEN～ELSE～END IF で処理を判断・分岐する」の例では、判断・分岐は、「THEN」以降の処理と「ELSE」以降の処理の2択でした。

例えば、入力した数値が 0～100未満の場合、100以上の場合、マイナスの場合のように、3択に処理を判断・分岐したい場合、以下のようなプログラムを書きます。

```
INPUT ONDO
IF 100 <= ONDO THEN
    PRINT "100以上"
ELSEIF 0 <= ONDO THEN
    PRINT "0～100未満"
ELSE
    PRINT "マイナス"
END IF
```

上のプログラムは、以下のように処理&実行されます。

1. 「INPUT」コマンドで入力を求め、入力した数値が「ONDO」変数に代入されます。
2. 次に、「IF」コマンドの「100 <= ONDO」条件式で、「ONDO」変数の値が 100以上か判断します。
3. ここで 100以上の数値が入力されていたら、「THEN」の後の「PRINT」コマンドが実行され、「100以上」と画面に出力されます。
4. 「ONDO」変数の値が 100以上でない場合、次の「ELSEIF」コマンドの「0 <= ONDO」条件式で、「ONDO」変数の値が 0以上か判断します。
5. ここで、0以上の数値が入力されていたら、「THEN」の後の「PRINT」コマンドが実行され、「0～100未満」と画面に出力されます。
6. 「ONDO」変数の値が 0以上でない場合、「ELSE」の後の「PRINT」コマンドが実行され、「マイナス」と画面に出力されます。

このように、「ELSEIF」を並べて記述することで、3択、4択、…と、何択もの判断・分岐の処理を書けます。

「IF」および「ELSEIF」を使った条件式の記述は、条件判断の順番に気を付けてください。

例えば、以下のように 条件判断の順番を入れ替えてみます。

```
INPUT ONDO
IF 0 <= ONDO THEN
    PRINT "0～100未満"
ELSEIF 100 <= ONDO THEN
    PRINT "100以上"
ELSE
    PRINT "マイナス"
END IF
```

例えば、「ONDO」変数の値が 200の場合、最初に提示したプログラムでは、「100以上」と画面に出力されますが、上のプログラムでは、「0～100未満」と画面に出力されます。

何故かというと、最初に提示したプログラムでは、最初の条件式「 $100 \leq ONDO$ 」で「ONDO」変数の値が 100以上か判断した後、条件が満たさない時に、次の条件式「 $0 \leq ONDO$ 」で「ONDO」変数の値が 0以上か判断していました。

しかし、上のプログラムでは、最初の条件式が「 $0 \leq ONDO$ 」の為、「ONDO」変数の値 200の条件を満たしてしまったが為、次の「ELSEIF」の条件判定に行かず、そのまま「THEN」の処理に分岐してしまったのです。

このように、3択、4択になるような処理を記述する際は、条件式の順番に留意する必要があります。

### 3.2.4 SELECT は、IF の亜種

「3.2.3 IF と ELSEIF を組み合わせて 3択、4択を作る」のプログラムでは、複数の選択を「IF」コマンドと「ELSEIF」コマンドを使って実現しました。

複数の選択肢から判断・分岐するコマンドとして、他に「SELECT」コマンドがあります。

```
INPUT "1~3の間で入力してください"; NO
SELECT CASE NO
CASE 1
    PRINT "1が入力されました"
CASE 2
    PRINT "2が入力されました"
CASE 3
    PRINT "3が入力されました"
CASE ELSE
    PRINT "1~3以外が入力されました"
END SELECT
```

上のプログラムは、以下のように処理&実行されます。

1. 「INPUT」コマンドでユーザーが数値を入力してもらい、「NO」変数に代入します。
2. 次に「SELECT CASE」コマンドで、次のパラメータ「NO」変数の値を評価します。
3. 1なら「CASE 1」へ、  
2なら「CASE 2」へ、  
3なら「CASE 3」へ、  
それ以外なら「CASE ELSE」の各行に判断・分岐し、下行の「PRINT」コマンドを実行します。

「IF」コマンドとの違いは、「IF」コマンドの場合 条件式で判断するのに対して、「SELECT CASE」コマンドは、一つの値を評価します。

「IF」コマンドより判断の記述上の制約がありますが、その分 わかりやすく書けます。

### 3.2.5 FOR ループの中にFORループを作る

「3.1.15 FOR～NEXT で処理を指定回数繰り返す」の項で、処理を一定回数繰り返す説明を行いました。

これを応用して、かけ算の九九表を作つてみましょう。

```
FOR Y=1 TO 9
    FOR X=1 TO 9
        PRINT X * Y,
    NEXT X
    PRINT
NEXT Y
```

上のプログラムで、「FOR」コマンドの中に、更に「FOR」コマンドを書いています。

最初の「FOR」コマンドでは、「Y」変数を使って繰り返すようにしており、次の「FOR」コマンドでは、「X」変数を使って繰り返すようにしています。

このように、ループの中にループを入れる構造を、一般に多重ループといいます。

多重ループを組むとき、同じ変数を使つたり、「NEXT」コマンドの変数を入れ子(ネスト)にしても構いませんが、入れ子がクロス(入れ替わり)させてはいけません。

例えば、以下のような、入れ子がクロスするようなプログラムは、エラーとなります。

```
FOR Y=1 TO 9
    FOR X=1 TO 9
        PRINT X * Y,
    NEXT Y
    PRINT
NEXT X
```

「FOR」コマンドでループを組む際は、「NEXT」コマンドの変数名と必ず対になるように気をつけてください。

### 3.2.6 配列を使って多くのデータを扱う

「3.1.8 AJANは変数という箱を持つ」で、値を入れる変数があることを説明しました。

多くの値を変数で管理したいとき、「A1」、「A2」、「A3」、…という具合に変数を記述していたら大変です。

そんな時、配列変数を使って、まとめて管理する事ができます。

配列変数は、複数の同じ型の値を入れる変数の箱のようなものです。

```
DIM X(9)      ' 配列変数Xを宣言する。0から9まで、10の箱を用意する
FOR N=0 TO 9
    X(N) = N + 1
NEXT N
PRINT X      ' 配列変数Xの内容を出力する
```

上のプログラムで、配列変数は、「DIM」コマンドの次のパラメータ「X」で宣言します。丸括弧の中の数値で、箱の大きさ（「要素数」と呼びます）を指定します。

普通の変数は、いきなり代入という形で使用できていましたが、配列変数は必ず宣言する必要があることに留意ください。

宣言した配列変数を使用しているのは、「X(N)」の所です。配列変数の場合、用意した箱の何番目に読み書きするか、丸括弧の中の数値で指定します。これを「添字」と呼びます。

添字で指定する数値は、用意した箱の大きさを越えてはいけません。越えた添字を指定すると、エラーとなります。

表計算ソフトのように、行と列のような大きさの配列変数を作ることもできます。

```
DIM ARY(9, 9)  ' 表計算ソフトの表のような、二次元の配列変数の箱を用意している
FOR Y=0 TO 9
    FOR X = 0 TO 9
        ARY(X, Y) = (X+1) * (Y+1)
    NEXT X
NEXT Y
```

上のプログラムは、行と列に相当する、二次元の配列変数「ARY」を宣言して利用しています。

丸括弧の中で、「,(カンマ)」区切りで、多次元の配列変数を宣言できます。

配列の詳細については、「3.3.3 配列、連想配列について」を参照ください。

### 3.2.7 実行時に大きさが変化する配列を扱う

「3.2.6 配列を使って多くのデータを扱う」の説明では、「DIM」コマンドを使って、複数の値をまとめて管理できる配列変数を作つてみました。

「DIM」コマンドは、あらかじめ管理する大きさ(=要素数)がわかる時に用います。

管理する大きさが、実行するときまで分からぬ場合は、「LIST」コマンドで配列変数を宣言して、「REDIM」コマンドで大きさを変えることができます。

```

LIST ARY          ' 可変長の配列変数を宣言
INPUT "配列の大きさを入力してください="; CNT
REDIM ARY(CNT-1)    ' 配列変数の大きさ(要素数)を変える

FOR I=0 TO CNT-1
    ARY(I) = I + 1
NEXT I
PRINT ARY        ' 配列変数の内容を出力する

```

「DIM」コマンドでは宣言時に大きさが確定していましたが、「LIST」コマンドは宣言時は要素数が1しかありません。

上のプログラムは、以下のように処理&実行されます。

1. 「INPUT」コマンドで配列の大きさを指定してもらい、「CNT」変数に数値を代入しています。
2. 次の「REDIM」コマンドで、「LIST」コマンドで指定した「ARY」配列変数に対して、配列の大きさを指定します。  
この呼び出しで、「ARY」変数は、「CNT」変数で指定した大きさの配列変数となります。
3. その後は、「DIM」コマンドと同じ配列変数のように扱うことができます。

なお、配列の大きさを変える「REDIM」コマンドの他に、少しずつ増減できる、「ONEDIM INSERT」「ONEDIM REMOVE」、「TWODIM INSERT」「TWODIM REMOVE」コマンドもあります。  
詳細は、「3.3.3 配列、連想配列について」の「c) 固定長の配列と可変長の配列」を参照ください。

このように、配列の大きさが予め判つている場合は「DIM」コマンド。判つてない場合は「LIST」コマンドを使います。

### 3.2.8 配列の大きさを調べる

さきに紹介したプログラムでは、「DIM」コマンドを使ったプログラムも、「LIST」コマンドを使ったプログラムも、配列変数の大きさは判っていました。

AJANの「関数」の中には、大きさが変化する配列を返すものがあります。

例えば、「SPLIT\$」関数は、パラメータ中の文字列の区切り文字に従って、文字列を分割し配列にして返します。

```
PRINT SPLIT$("Hello,AJAN,World", " ")
```

上のプログラム例では、「Hello,AJAN,World」という文字列に対して、「,(カンマ)」文字を区切りとして分割し、「Hello」、「AJAN」、「World」の3つの要素数を持つ配列して返します。

大きさの変わる配列変数は「LIST」コマンドで宣言できるので、この関数の受け取りは、「LIST」コマンドで配列変数を宣言して受け取る事ができます。

```
LIST ARY$  
ARY$ = SPLIT$("Hello,AJAN,World", ",")  
PRINT ARY$
```

このように大きさの変わる配列を受け取る事ができますが、配列の大きさが判らないと操作ができません。配列の大きさを調べる関数には、以下のようなものがあります。

CDIM	配列の次元数を調べます
LDIM	配列の要素数を調べます
UBOUND	配列の指定した次元の添え字最大値を得ます

例えば、先の「SPLIT\$」関数で得た結果に対して、配列の内容を調べるには、以下のように書くことができます。

```
LIST ARY$  
ARY$ = SPLIT$("Hello,AJAN,World", ",")  
FOR I=0 TO UBOUND(ARY$)  
    PRINT "ARY$("; I; ")=["; ARY$(I); "]"  
NEXT I
```

多次元配列の場合も、各次元の要素数や全体の要素数を調べることができます。

```
DIM ARY(2, 3, 4)
PRINT "次元数="; CDIM(ARY)          ' 3が出力されます。
PRINT "1次元目の要素数="; LDIM(ARY, 1)    ' 3が出力されます。
PRINT "2次元目の要素数="; LDIM(ARY, 2)    ' 4が出力されます。
PRINT "3次元目の要素数="; LDIM(ARY, 3)    ' 5が出力されます。
PRINT "全ての要素数="; LDIM(ARY)          ' 60が出力されます。
```

これらの関数を使用することで、可変長の配列を自由自在に操作できます。

### 3.2.9 ファイルからデータを読み取って記録する

例えば、購入した部品の単価の合計をコンピュータで計算したい時。

製品の単価の情報を、何かの方法でプログラムに教えてやらなければなりません。

「INPUT」コマンドで、キーボードから入力することも出来ますが、どこかのファイルに情報を書いておき、プログラムで読み込ませて計算したいこともあるでしょう。

AJANは、ファイルに対する読み書きができます。

以下のプログラムでは、ファイル「DATA.TXT」から製品の名前と単価を読み取り、各単価を合計して画面に出力します。

```
OPEN "DATA.TXT" FOR INPUT AS #1
SUM = 0
DO WHILE EOF(1) = FALSE
    INPUT #1, NM$, TANKA
    PRINT NM$, TANKA
    SUM = SUM + TANKA
LOOP
CLOSE #1
PRINT "合計="; SUM      ' 実行すると、「合計=1150」が最後に出力されます
```

ここで読む対象となる「DATA.TXT」の中身は、以下のようにだったとします。

```
BOOK, 1000
CANDY, 50
EGG, 100
```

ファイルを読み書きするには、「OPEN」コマンドでどのファイルを開くか指定します。

「OPEN」コマンドの次のパラメータで、「"DATA.TXT"」というファイルを開くことを指定します。さらに「FOR」の後のパラメータ「INPUT」で、このファイルを入力(=読み出す)として開く事を明示します。なお、書き込むには「OUTPUT」と指定します。

さらに「AS」の後のパラメータ「#1」で、ファイル番号を指定します。このファイル番号は、その他のコマンドで実際にファイルに対して読み書きしたり状態を調べたりする際に、どのファイルに対して行うのか指定する為に使用します。

これにより、同時に複数のファイルを扱うことができます。



ファイル番号が数値の前に「#」を付けるのは、ファイルを扱うコマンドで、これがファイル番号である事を明示する為です。  
「EOF」関数など、「#」を付けずに指定するものもあります。

開いたファイルからデータを読み出すのは「INPUT」コマンドです。

今までの説明の「INPUT」コマンドの使い方では、キーボードから入力を促していましたが、最初のパラメータ「#1」があると、「OPEN」コマンドで開いたファイル番号のファイルからデータを読み出します。

次のパラメータは「NM\$」変数と「TANKA」変数です。これにより、ファイルから、まず文字列を読み取って「NM\$」変数に、更に数値を読み取って「TANKA」変数に読み取ります。文字列の区切りは、「,(カンマ)」記号です。

「INPUT」コマンドで、ファイルからデータを各変数に読み取りますが、「DO WHILE」コマンドから「LOOP」コマンドで囲っているので、この読み取り処理が繰り返されます。

「DATA.TXT」の中身は、3行分しかないので、全て読み取つたら 繰り返し処理を終えて欲しいです。

ファイルからデータを読み取ったかの判断は、「DO WHILE」コマンドの条件式に書かれた「EOF」関数です。

「EOF」関数は、指定したファイル番号のファイルに対して、全てデータを読み終えてファイルの終端に達したら TRUE 値を返す関数です。

ここでの条件式は「EOF(1) = FALSE」のため、ファイル番号 1 のデータが、全て読み終えるまで繰り返しなさい という意味になります。

ファイルから全てのデータを読み終えると、ループを抜けて次の「CLOSE」コマンドにいきます。

「CLOSE」コマンドは使い終わった ファイルを閉じる命令です。

閉じ終えると、同じファイル番号でオープンして使いまわす事ができます。

「CLOSE」コマンドでファイルを閉じた後、今まで「TANKA」変数の値を「SUM」変数に加算して、求めた合計値を画面に出力して、プログラムを終えます。

このプログラムを実行すると、おおよそ以下のような実行結果が得られます。

BOOK 1000
CANDY 50
EGG 100
合計=1150
Ok

上のプログラムでは、ファイルからデータを読み取って合計値を画面に出力しましたが、これを改造して合計値をファイルに出力してみましょう。

```
OPEN "DATA.TXT" FOR INPUT AS #1
SUM = 0
DO WHILE EOF(1) = FALSE
    INPUT #1, NM$, TANKA
    PRINT NM$, TANKA
    SUM = SUM + TANKA
LOOP
CLOSE #1
PRINT "合計="; SUM
OPEN "RESULT.TXT" FOR OUTPUT AS #1
PRINT #1, SUM
CLOSE #1
```

追加した行「OPEN "RESULT.TXT" FOR OUTPUT AS #1」は、「FOR」の次が「OUTPUT」なので、「RESULT.TXT」ファイルを出力(=書き込む)用にオープンします。

次に、「PRINT」コマンドの最初のパラメータ「#1」で、書き込み用にオープンしたファイルに対して、「SUM」変数の値を書き込みます。

更に次の「CLOSE」コマンドで、書き込んだファイルを閉じます。

このように、AJANはファイルへの読み書きが簡単にできます。

### 3.2.10 ファイルの読み書きの別のやり方

「3.2.9 ファイルからデータを読み取って記録する」のプログラムでファイルの読み書きの例では、開いたファイルを少しづつ読み取っていました。

目的によっては、ファイルから一気に全てのデータを読み取ったり、一気に書き込んでしまいたい事もあるでしょう。

AJANでは、ファイルから一気に全てのデータを読み取るには「STR\_FREADALL\$」関数を、一気に書き込むには「STR\_FWRITEALL」命令を用います。

```
DATA$ = STR_FREADALL$("DATA.TXT")
LIST ARY$
ARY$ = SPLIT$(DATA$, CHR$(10))
FOR I=0 TO UBOUND(ARY$)
    PRINT I; "行目="; ARY$(I)
NEXT I
```

上のプログラムでは、まず「STR\_FREADALL\$」関数で、「DATA.TXT」ファイルのデータ内容を「DATA\$」変数に一気に取り込みます。

次に、「SPLIT\$」関数で、「CHR\$」関数を使って文字コード10を区切り文字にして、配列変数に代入します。

Linuxでは、テキスト文書で改行は文字コード10なので、この呼び出しにより「ARY\$」変数には行単位に分割されて代入されることになります。

	参考までに、Windowsでは、改行は CHR\$(13)+CHR\$(10) の2バイトを使用します。 Linuxは CHR\$(10) の1バイトです。
---	---

データをファイルに一気に書き込むには、「STR\_FWRITEALL」命令を呼び出します。

以下の例では、「STR\_FREADALL\$」で読み出した文字列データを、「LCASE\$」関数で文字列中の大文字を小文字に変換し、「STR\_FWRITEALL」命令で書き込んでいます。(参考までに、小文字から大文字に変換するには、「UCASE\$」関数を使います)

```
DATA$ = STR_FREADALL$("DATA.TXT")
DATA$ = LCASE$(DATA$)
STR_FWRITEALL "DATA2.TXT", DATA$
```

「OPEN」コマンドを使ってファイルをオープンして、「INPUT」や「PRINT」コマンドを使って少しづつ読み書きするのに比べて、一気に処理するので、メモリを大量に消費する反面、使い方によっては処理の高速化に繋がります。ケースに応じて、使い分けてください。

### 3.2.11 Linux外部コマンドの力を借りる

AJANは多くのコマンドがありますが、森羅万象全ての要求に対応できる訳ではありません。

そんな時には、Linuxの外部コマンドの力を借りましょう。

例えば、とある PDFファイルをビューワで表示したいと思います。

Linuxでは、「evince」という外部コマンドがPDFファイルを表示できるビューワです。

まずは、Linuxの端末上で試して見ましょう。ビューワが開いてPDFファイルが見られます。

```
$ evince /usr/share/doc/interface/AJANPro/Help.pdf
```

	Linuxの外部コマンドは大文字・小文字を区別します。要注意してください。
---	---------------------------------------

これを、AJANから呼び出せるようにします。

AJANから、Linuxの外部コマンドを呼び出すには「SHELL SYSTEM」関数を使います。

```
PRINT "呼び出し結果="; SHELL SYSTEM("evince /usr/share/doc/interface/AJANPro/Help.pdf")
```

同じように、ビューワが開いてPDFファイルが見られます。

Linuxは、システムの内部情報を簡単に見られる作りとなっており、例えば 下図のように、「cat /proc/cpuinfo」とコマンドを入力すると、使用しているコンピュータのCPU情報が得られます。

```
$ cat /proc/cpuinfo
processor      : 0
vendor_id     : GenuineIntel
cpu family    : 6
model         : 55
model name    : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
stepping      : 9
microcode     : 0x907
...
...
```

CPUのモデル名は、このうち「model name」とある箇所です。

この出力結果を、AJANで読み取って「model name」のある行の情報を取り出せば、簡単にCPUのモデル名が得られます。

早速AJANで読み取りたい所ですが、読み取りを簡単にするために、Linuxの外部コマンドを更に組み合わ

せて抽出してみます。

Linuxの外部コマンドは、出力されるテキスト文字列を、別の外部コマンドに引き継いで処理させる機能があります。(技術的にはシェルと呼ばれるソフトの機能です)

これを行うには、外部コマンドと外部コマンドを「|(パイプ)」記号で繋ぎます。

まず、テキスト文字列を、「model name」とある行だけ抽出してみます。

Linuxの外部コマンドで文字列を抽出するには、「grep」コマンドを使います。

```
$ cat /proc/cpuinfo | grep "model name"
model name      : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
model name      : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
model name      : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
model name      : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
```

長かったテキスト文字列が、数行にまとめました。

同じ文字列などを一つにまとめたいです。同じ内容の文字列を1つにまとめるには、「uniq」コマンドを使います。

```
$ cat /proc/cpuinfo | grep "model name" | uniq
model name      : Intel(R) Atom(TM) CPU E3845 @ 1.91GHz
```

出力が1行に収束したので、これをAJANからLinux外部コマンドを呼び出して、テキスト文字列を読み取りましょう。

Linux外部コマンドを呼び出して、そのテキスト文字列の出力結果を得るには「SHELL CALLOUT\$」関数を使います。

```
PRINT SHELL CALLOUT$("cat /proc/cpuinfo | grep ""model name"" | uniq")
```

Linux外部コマンドを呼び出した際、「"(ダブルクオーテーション)"」記号のあった部分は、2つ続けて「""」という風に置き換える必要があります。

それ以外は、さきほど端末上で Linux外部コマンドを呼び出して得た結果と同じものが、AJAN上でも得られる事がわかります。

このように、Linux外部コマンドと組み合わせる事で、より多くの仕事ができます。

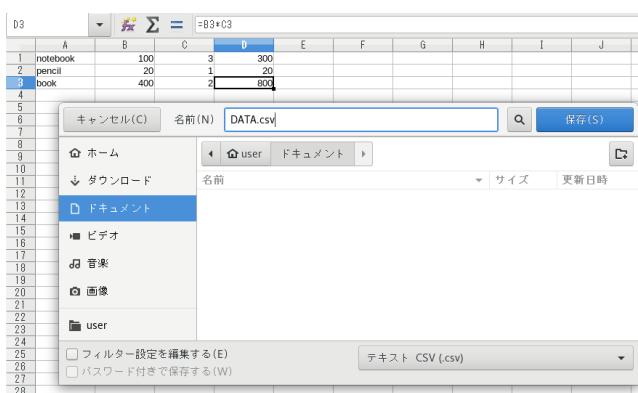
### 3.2.12 表計算ソフトのCSVデータを配列に読み取る

表計算ソフトは、表のような升目に数値やデータを入力し、縦横の合計値を出したり、グラフを表示させたりと、大変使い勝手の良いソフトです。

ここで入力したデータを、AJANで読み取って処理してみましょう。

まずは、表計算ソフト(Calcなど)で、いくつかのデータを入力した後、AJANが読み取れる形式のファイルに保存します。

ファイルに保存する際、ファイル形式を「テキストCSV(.csv)」にして保存してください。



次に、保存したファイル(仮に「DATA.csv」とします)を読み取って、文字列の配列に変換します。

```

DATA$ = STR_FREADALL$("DATA.csv")           ' CSVファイルのデータを読み取ります
LIST ARY$                                     ' 可変長の文字列配列変数を宣言します
ARY$ = CSV2ARRAY$(DATA$, ",", """", FALSE)   ' 読み取ったCSVデータを、文字列配列に変換します
PRINT "行数="; LDIM(ARY$, 1)
PRINT "列数="; LDIM(ARY$, 2)
FOR Y=0 TO UBOUND(ARY$, 1)
    FOR X=0 TO UBOUND(ARY$, 2)
        PRINT X; ". "; Y; "="; ARY$(Y,X)
    NEXT X
NEXT Y

```

上のプログラムでは、まず「STR\_FREADALL\$」関数で、CSVファイルの内容を読み取ります。

次に、「LIST」コマンドで配列変数を用意しておき、「CSV2ARRAY\$」関数を使って、CSV形式の文字列を文字列の配列変数に変換して代入します。

文字列配列に変換して代入した後は、表のようにアクセスしながら処理が可能となります。

### 3.2.13 サブルーチン内でローカル変数を使って独立性を高めるべき

「3.1.19 AJANはサブルーチンで共通化する」や「3.1.20 AJANはユーザ定義関数で独自の関数を作れる」で、処理を共通化する事ができる事を知りました。

処理を共通化できることで、プログラムは短くできて見通しが良くなり、万が一 サブルーチン内にミスが見つかった場合、そこを修正するだけで良いので、これほど良いことはありません。

	今まで説明していた変数は、異なるルーチン全てに対して有効でした。これを、本項で説明するローカル変数と対になる言葉として、特にグローバル変数と呼ぶことがあります。
	サブルーチン内でしか使用しない変数は、本項で紹介するローカル変数を使用する事を、強くお奨めします。

サブルーチンやユーザ定義関数を作つて共通化するのは非常にお奨めしたいことですが、一つ注意すべき事項があります。

共通化の処理を書くとき、中で変数を作つて使用する事があります。この時、ローカル変数という変数を宣言して使用すべきです。

例えば、以下に問題の起きる事例を紹介します。

```

' 文字列を16進数の文字コードの羅列に変換します
FUNCTION MSG2HEX$(S$)
    R$ = ""
    FOR I=1 TO LEN(S$)
        R$ = R$ + HEX$(ASC(MID$(S$, I, 1)))
    NEXT I
    MSG2HEX$ = R$
END FUNCTION

' 文字列をカンマ(,)区切りで、16進数の文字コードの羅列を出力します
SUB MSG_LIST(S$)
    LIST ARY$
    ARY$ = SPLIT$(S$, ",")
    FOR I=0 TO UBOUND(ARY$)
        PRINT I, MSG2HEX$(ARY$(I))
    NEXT I
END SUB

CALL MSG_LIST("Hello,AJAN,World")

```

上のプログラムは、本来以下のような出力を得たいと思っています。

「Hello,AJAN,World」という文字列を、プログラムを実行すると、「MSG\_LIST」サブルーチンを実行し、

中の「**SPLIT\$**」関数を呼び出してカンマ(,)区切りの配列変数に代入します。

次に、「I」変数でループを回しながら、「Hello」、「AJAN」、「World」という文字列を、「MSG2HEX\$」ユーザ定義関数を呼び出して、文字列に対して1文字ずつ16進数変換した文字コードの結果を得て、画面に出力しようとしています。

0	48656C6C6F
1	414A414E
2	576F726C64

しかし、上のプログラムを実行すると、結果は以下のようになります。

6	48656C6C6F
---	------------

問題箇所は、「**MSG\_LIST**」サブルーチン内のFORループに使用している「I」変数と、「**MSG2HEX\$**」ユーザ定義関数内のFORループに使用している同じ名前の「I」変数です。

何故期待した通りの動きにならないのか？コンピュータの立場にたって、考えてみましょう。

(判り難い場合は、「4.11 プログラムのデバッグ」を参考に、「I」変数を変数ウォッチしながら、1行ずつステップ実行すると良いです)

1. まず、「**MSG\_LIST**」サブルーチンに入った後、「**SPLIT\$**」関数を使って文字列を配列に区切って、「**ARY\$**」配列変数に代入します。
2. 次に、「**FOR**」コマンドで、「I」変数を使ってループを始めます。最初は 0 です。
3. 「**PRINT**」コマンドを実行する際、「**MSG2HEX\$**」ユーザ定義関数に入ります。
4. この中で、更に「**FOR**」コマンドで、同じ名前の「I」変数を使ってループを始めます。「**MSG\_LIST**」サブルーチン内で「I」変数は 0 だったのが、ここでいきなり 1 に書き換えられてしまいます。
5. 「**MSG2HEX\$**」ユーザ定義関数の「I」変数は、TOの先の「**LEN**」関数分ループを回します。
6. 最初に評価される文字列は「Hello」なので、5文字分です。「I」変数の値は、 $5 + 1 = 6$  の時点で、「**FOR**」ループを抜ける事になります。
7. 「**MSG2HEX\$**」ユーザ定義関数の実行を終えて、「**MSG\_LIST**」サブルーチンに戻ってきます。
8. 「I」変数の値は、戻ってきた時点で 6 に書き換えられており、「**FOR**」ループのループ条件を超えてしまっているので、いきなりプログラムが終了してしまう次第です。

この例では、2つの処理で同じ「I」変数が被ってしまった為、想定しない動作になってしまいました。

これを避けるには、異なる変数名に変えれば良いです。例えば片方を「I」変数でなく、「J」変数など。

しかし、このようにサブルーチンやユーザ定義関数ごとに、異なる変数名をふっていくのは、非常に面倒な事です。同じ変数名で、例えば定義したルーチンでのみ変数が使用可能にできないでしょうか？

ローカル変数は、このように定義したルーチンでのみ変数が使用可能になります。

例えば「**MSG\_LIST**」サブルーチンでのみ有効な「I」変数。「**MSG2HEX\$**」ユーザ定義関数でのみ有効

な「I」変数。というのを、ローカル変数を宣言することで、使用可能になります。  
以下に修正したプログラム例を示します。修正した箇所を、赤文字で示します。

```
' 文字列を16進数の文字コードの羅列に変換します
FUNCTION MSG2HEX$(S$)
    LOCAL R$, I
    R$ = ""
    FOR I=1 TO LEN(S$)
        R$ = R$ + HEX$(ASC(MID$(S$, I, 1)))
    NEXT I
    MSG2HEX$ = R$
END FUNCTION

' 文字列をカンマ(,)区切りで、16進数の文字コードの羅列を出力します
SUB MSG_LIST(S$)
    LOCAL I
    LOCAL LIST ARY$
    ARY$ = SPLIT$(S$, ",")
    FOR I=0 TO UBOUND(ARY$)
        PRINT I, MSG2HEX$(ARY$(I))
    NEXT I
END SUB

CALL MSG_LIST("Hello, AJAN, World")
```

前のプログラムと異なっているのが、「LOCAL」コマンドと「LOCAL LIST」コマンドです。  
「LOCAL」コマンドは、サブルーチンないしはユーザ定義関数内でのみ有効な、ローカル変数を宣言する命令です。  
「LOCAL LIST」コマンドは、可変長のローカル配列変数を宣言する命令です。  
「MSG\_LIST」サブルーチンと「MSG2HEX\$」ユーザ定義関数のそれぞれに、「I」ローカル変数が宣言されています。

これを実行すると、「MSG\_LIST」サブルーチンから、「MSG2HEX\$」ユーザ定義関数の処理に入って、「I」変数を使ってFORループの処理を済ませた後「MSG\_LIST」サブルーチンに戻ってきたとき、「I」変数の値は「MSG\_LIST」サブルーチンで宣言・使用された値に戻っています。(先の例で言えば、0に戻ります)

これにより、期待したプログラムの実行結果が得られます。

### 3.2.14 AJANはエラーが起きても処理を続けられる

「3.1.5 AJANはエラーを検知する」で、プログラムの不具合などにより、実行時にエラーが発生する場合がある事を説明しました。

このように実行時にエラーが発生したときでも、それが想定範囲の出来事であれば、処理を続けられるようになります。

```

ON ERROR CALL CB_ERR          ' エラー処理ルーチンを定義します
ERROR ON                      ' エラー処理ルーチンを有効にします
PRINT 10 / 0                  ' ここで実行時エラーが起きます
PRINT "終わり"

SUB CB_ERR(I_ERR, I_ERM$, I_ERL)
    IF I_ERR = &H0100000A THEN
        PRINT "0除算の実行時エラーを検知しました。処理を継続します"
    ELSE
        PRINT "エラー : &H" + HEX$(I_ERR) + " の実行時エラーを検知しました。処理を中止します"
    END
END IF
END SUB

```

上のプログラムは、以下のように処理&実行されます。

1. 「ON ERROR CALL」コマンドで、次のパラメータ「CB\_ERR」サブルーチンを、エラーが発生した際にジャンプする命令として宣言します。
2. 次に「ERROR ON」コマンドで、この後以降 エラーが発生したらジャンプする事を「有効」にします。
3. 次に「PRINT 10 / 0」の行で、0で割り算しようとしてエラーとなります、「CB\_ERR」サブルーチンにエラー情報と共にジャンプします。
4. 「CB\_ERR」サブルーチンにジャンプした時、「I\_ERR」変数にエラーコード、「I\_ERM\$」変数にエラーメッセージ、「I\_ERL」変数にエラーの発生した行番号がセットされてジャンプしてきます
5. サブルーチンの処理の中で、「IF」コマンドで「I\_ERR」変数のエラーコード値が、&H0100000Aかどうか判断して、そうであれば メッセージを出力した後、サブルーチンの処理を終えて、エラーが発生した行の次の行へ戻ります。
6. そうでなければ、メッセージを出力した後「END」コマンドでプログラムを終了します。

エラーコードは、どんな原因でエラーが発生したのか、大まかな要因を数値で表します。

この値を用いることで、処理を継続するか、中断終了するか判断する事ができます。

エラーコードの詳細はコマンドリファレンス「標準コマンド編」の「4.1 エラーコードリファレンス」を参照ください。

### 3.2.15 アプリケーション独自のエラーあるいは異常処理を作る(1)

「3.2.14 AJANはエラーが起きても処理を続けられる」で、実行時にエラーが発生した時に、処理を続ける方法を説明しました。

アプリケーションを作成する場合、AJANの使用方法に問題があつて エラーとなる場合とは別に、アプリケーション固有に ユーザー(=使用者)の使用方法に問題があつて、エラーとしたい場合があります。

例えば、おおよそ 3つ程度の選択肢があり、ユーザーに入力選択した後に 選択した処理に分岐するようなアプリケーションを作るとします。

```

PRINT "メニュー"
PRINT "1... 今日の宿題をする"
PRINT "2... 明日の予習をする"
PRINT "3... 今日は寝る"
INPUT "今日のやりたい事の番号を入力してください"; NUM%
IF NUM% = 1 THEN
    PRINT "今日の宿題処理に入る"
ELSEIF NUM% = 2 THEN
    PRINT "明日の予習処理に入る"
ELSEIF NUM% = 3 THEN
    PRINT "今日は寝る処理に入る"
END IF

```

INPUT命令で入力を期待するのは、1, 2, 3のうちのどれかです。それ以外の数値を入力した時、IF～ELSEIF～END IF の判定条件に該当するものが無い為、その数値は「想定外」となります。

つまり、AJANの文法上は問題がありませんが、アプリケーションから見て、ここで 1, 2, 3以外の数値を入力してはいけません。これを、「想定外の入力」とか「アプリケーション固有の入力エラー」などのような言い方をします。

このままでは、選択肢以外の入力をしてしまうと、正常に動作しません。手順書なりで「選択肢以外を入力しないで」と伝える事も出来ますが、基本 人間は間違える生き物なので、選択肢以外の入力をしてしまうこともあります。

ならば、選択肢以外の入力が行われた時、何がしかエラーメッセージを出して、再入力を促すようにするとユーザーに優しいでしょう。この想定外の入力などに対して対処する処理を「異常処理」と呼んだりします。以下に、選択肢以外の入力があった時の異常処理を組んだ例を示します。

```
MENU_LOOP:  
PRINT "メニュー"  
PRINT "1... 今日の宿題をする"  
PRINT "2... 明日の予習をする"  
PRINT "3... 今日は寝る"  
INPUT "今日のやりたい事の番号を入力してください"; NUM%  
  
IF NUM% = 1 THEN  
    PRINT "今日の宿題処理に入る"  
ELSEIF NUM% = 2 THEN  
    PRINT "明日の予習処理に入る"  
ELSEIF NUM% = 3 THEN  
    PRINT "今日は寝る処理に入る"  
ELSE  
    PRINT "エラー!! メニュー以外の入力です"  
    GOTO MENU_LOOP      ' 再入力の為に、MENU_LOOP にジャンプする  
END IF
```

追加した異常処理では、ELSE命令を追加して、選択肢以外の入力があるとメッセージの後に、再度メニュー表示と選択肢入力を促すようにしています。

### 3.2.16 アプリケーション独自のエラーあるいは異常処理を作る(2)

「3.2.15 アプリケーション独自のエラーあるいは異常処理を作る(1)」で、入力選択間違いに対するアプリケーション独自の異常処理を書いてみました。

先の例では、単に PRINT文で エラーメッセージを表示していましたが、アプリケーションの作り方次第では、例えば ON ERROR CALL で定義する エラー処理と組み合わせたいケースも出てくるでしょう。

ON ERROR CALL で定義するエラー処理と組み合わせるには、ERROR命令で 任意にエラーを発生させます。

以下に、ERROR命令で、任意にエラーを発生させる事例を示します。

```

ON ERROR CALL CB_ERR      ' エラー発生時、CB_ERR サブルーチンが呼ばれます
ERROR ON

MENU_LOOP:
PRINT "メニュー"
PRINT "1... 今日の宿題をする"
PRINT "2... 明日の予習をする"
PRINT "3... 今日は寝る"
INPUT "今日のやりたい事の番号を入力してください"; NUM%

IF NUM% = 1 THEN
    PRINT "今日の宿題処理に入る"
ELSEIF NUM% = 2 THEN
    PRINT "明日の予習処理に入る"
ELSEIF NUM% = 3 THEN
    PRINT "今日は寝る処理に入る"
ELSE
    ERROR &H01000004, "エラー!! メニュー以外の入力です", "MAIN"
    GOTO MENU_LOOP          ' 再入力の為に、MENU_LOOP にジャンプする
END IF

' エラー発生時に呼び出されるサブルーチン
SUB CB_ERR(I_ERR, I_ERM$, I_ERL)
    PRINT "エラー発生!! コード:"; HEX$(I_ERR); " 説明:"; I_ERM$; " 行:"; I_ERL
END SUB

```

前の事例では、エラーメッセージは単なる PRINT 命令でしたが、ここでは ERROR 命令としています。ERROR 命令で、任意にエラーを発生させ、ON ERROR CALL 命令で定義した CB\_ERR サブルーチンへとジャンプし、エラーメッセージを出力して戻ってきます。

!	事例では、ERROR命令で指定したエラーコードは、AJANのエラーコードから選択していますが、SETERRTBL命令で、独自のエラーコードを定義して使用することもできます。
---	--

このように、アプリケーション独自の異常処理を、AJANのエラー処理と組み合わせる事が可能です。

### 3.2.17 AJANは割り込みができる

先の説明では、プログラム実行中にエラーが発生したとき、エラーに対処するサブルーチンにジャンプして対処する方法を紹介しました。

このように、プログラム実行中に、何かの要因で「割り込んで別の処理」する事がAJANはできます。これを一般的に「割り込み」処理といいます。

AJANで割り込み処理ができるコマンドと機能が、いくつかあります。(下表は代表的なコマンドです)

ON TIME\$ CALL	指定時間になつたら割り込み
ON TIMER CALL	指定周期になつたら割り込み
ON KEY CALL	特定のファンクションキーが押されたら割り込み
ON WEB CALL	Webのボタンやテキストボックスの変更が行われると割り込み
ON GUEVT CALL	GUIのボタンを押すやウインドウを閉じるなどが行われると割り込み
ON CAN CALL	CAN通信で受信したりバッファが一杯になると割り込み
ON CM CALL	システム固有の機能で、温度異常や電圧異常などを検知したときに割り込み
ON COMMON CALL	グローバル共有を介して、メッセージを受信すると割り込み。

以下のプログラムは、この中で「ON TIMER CALL」コマンドを使って、3秒周期で特定のメッセージを出力します。

```

ON TIMER(3000) CALL CB_TIMER
TIMER ON

DO WHILE TRUE
    PRINT TIME$
    SLEEP 1          ' ON TIMER CALLは、SLEEP中でも割り込みが発生します
LOOP

SUB CB_TIMER()
    PRINT "3秒経過しました"
END SUB

```

上のプログラムは、以下のように処理&実行されます。

1. 「ON TIMER CALL」コマンドで、次のパラメータ「ON\_TIMER」サブルーチンを、指定周期毎にジャンプする命令として宣言します。
2. 指定周期は「ON TIMER CALL」コマンドの「TIMER」の括弧内の数値です。  
ミリ秒単位で数値を指定します。3000 なので 3秒周期で「割り込み」が発生します。
3. 次に「TIMER ON」コマンドで、この後以降 3秒周期でジャンプする事を「有効」にします。
4. 次の処理は、「DO WHILE」コマンド内で、「PRINT」コマンドの「TIME\$」関数で現在時刻を画面に出力した後、「SLEEP」コマンドで 1秒休止するのを、繰り返します。
5. 繰り返す間、3秒経過すると、「ON TIMER CALL」コマンドで宣言した「CB\_TIMER」サブルーチンに「割り込み」ジャンプします。
6. 「PRINT」コマンドで「3秒経過しました」と画面に出力した後、サブルーチンから抜けて、元の処理に戻ります。
7. その後、また3秒経過すると、同じ「CB\_TIMER」サブルーチンにジャンプして処理して抜ける。というのを繰り返し続けます。

このように「割り込み」機能を使うと、何かしらの要因をコンピュータが検知すると、待ったを掛けて「割り込み」処理することができます。

!	ON TIMER CALLなど、いくつかの割り込みルーチンは、SLEEP呼び出し中に割り込みが行われます。 ON GUEVT CALLは、OSの仕様上の制限により、SLEEP呼び出し中の割り込みが行われません。 詳細は、各コマンドの説明を参照ください。
---	--

### 3.2.18 INCLUDEで他のファイルを取り込む

AJANでプログラムを書き続けていると、どんどんプログラムの行数が長くなっています。

プログラムが長くなってくると、エディタ上でプログラムの処理を見渡すにも時間が掛かるようになります。

一人の人間が全体を把握できる能力には限界があるので、一般的には サブルーチンやユーザ定義関数などで共通化や、処理を分割して把握できるようにします。

共通化や処理を分割したとき、一旦 目に見えない所に置いた方が、気にしなくて済みますし、書き間違う事もありません。

AJANでは、共通化や分割した処理を別のファイルに括り出しておいて、後から取り込む事ができます。

「3.2.13 サブルーチン内でローカル変数を使って独立性を高めるべき」で紹介したプログラムを事例に、2つのファイルに分けて管理する事例を紹介します。

まず、以下のプログラムを「COMMON.AJN」とします。これは、共通化となるサブルーチンおよびユーザ定義関数の処理を集めたファイルです。

```
' 文字列を16進数の文字コードの羅列に変換します
FUNCTION MSG2HEX$(S$)
    LOCAL R$, I
    R$ = ""
    FOR I=1 TO LEN(S$)
        R$ = R$ + HEX$(ASC(MID$(S$, I, 1)))
    NEXT I
    MSG2HEX$ = R$
END FUNCTION

' 文字列をカンマ(,)区切りで、16進数の文字コードの羅列を出力します
SUB MSG_LIST(S$)
    LOCAL I
    LOCAL LIST ARY$
    ARY$ = SPLIT$(S$, ",")
    FOR I=0 TO UBOUND(ARY$)
        PRINT I, MSG2HEX$(ARY$(I))
    NEXT I
END SUB
```

次に、「COMMON.AJN」を取り込む、メインの処理部分です。

```
INCLUDE "COMMON.AJN"

CALL MSG_LIST("Hello, AJAN, World")
```

「INCLUDE」コマンドの次のパラメータ「COMMON.AJN」で、共通化の部分を集めた

「COMMON.AJN」ファイルを取り込む事を指示します。これを呼び出すとAJANでコンパイルするとき、この場所に「COMMON.AJN」の内容を取り込んでコンパイルします。

この呼び出しで、「MSG\_LIST」サブルーチンと「MSG2HEX\$」ユーザ定義関数のプログラムが取り込まれているので、後は「CALL MSG\_LIST」でサブルーチンを呼び出すだけです。

このように「INCLUDE」コマンドを使うと、プログラムの見た目が非常にすっきりします。

!	「INCLUDE」で取り込む対象となるファイルは、同じフォルダに置くことをお奨めします。もし、異なる場所に置く場合、ファイルを置いた場所を、ディレクトリ情報付きで指定しなければなりません。
---	--

### 3.2.19 AJANはマルチスレッドで並行処理ができる

「3.2.17 AJANは割り込みができる」で紹介したように、AJANは特定の処理を行っている中で、何かしらの要因で割り込んで処理する事ができます。

これと同じように、特定の処理を行っている中で、並行して別の何かの処理を行いたい要求があつたとします。

AJANは、この並行して処理する機能を持っています。これを、「マルチスレッド」機能といいます。

以下のプログラムでは、0以上1未満の乱数を得る「RND」関数を何度も振って、0.5未満が出た数と全体に対する比率を、およそ1秒おきに出力します。

```

ATTACH THREAD 1, CB_TH          ' スレッドを起動する

SUM& = 0
MIMAN& = 0

' 1秒休止して、現時刻と乱数を振った数、乱数が0.5未満が出た回数と比率を表示します
DO WHILE TRUE
    SLEEP 1
    PRINT TIME$, SUM&, MIMAN&, MIMAN& / SUM&
LOOP

END

' 並行して動くスレッドの処理
DEFINE THREAD CB_TH(NUM%)

' 亂数が0.5未満が出たら MIMAN& 変数を +1 します
' 亂数を降る都度、SUM& 変数を +1 します
DO WHILE TRUE
    IF RND < 0.5 THEN
        MIMAN& = MIMAN& + 1
    END IF
    SUM& = SUM& + 1
LOOP

END THREAD

```

上のプログラムでは、まず「ATTACH THREAD」コマンドで、現在動いているプログラムとは別に並行する処理、「スレッド」を起動します。次のパラメータ「1」で、スレッド番号 1で、更に次のパラメータ「CB\_TH」ルーチンを、別に並行して動く「スレッド」として起動します。

なお、現在動いている「スレッド」の事を一般的に「メインスレッド」と言い、「ATTACH THREAD」コマンドを使って別に起動する「スレッド」のことを「ワーカースレッド」と呼びます。

「ワーカースレッド」の処理は、「DEFINE THREAD」コマンドから「END THREAD」コマンドの間の「CB\_TH」ルーチンの処理です。

パラメータ「NUM%」変数は、「ATTACH THREAD」コマンドで指定したスレッド番号の数値が渡されます。ここでは 1です。

「IF」コマンドの中の条件式「RND < 0.5」で、「RND」関数を振って、0.5未満なら「MIMAN&」変数に対して +1 しています。

その後、「SUM&」変数に対して必ず +1 しています。

「ワーカースレッド」の処理は、「DO WHILE」コマンドで、この処理を永遠に続けています。

「メインスレッド」の処理は、「SLEEP」コマンドで 1秒ほど休止した後、「PRINT」コマンドの「TIME\$」関数で現在時刻を出力し、「SUM&」変数、「MIMAN&」変数、および「MIMAN& / SUM&」で割り算した結果を出力しています。

この演算式により、「RND」関数で 0.5未満が出た比率が outputされることが判ります。

このように、「マルチスレッド」機能を使うと、同時並行で異なる処理する事ができます。

### 3.2.20 複数ファイルを扱う為の、グローバル変数の名前付けの工夫事例

多くのサブルーチンをまとめた複数のAJANファイルを、INCLUDEして扱うとき、個々のAJANファイルのサブルーチン名やグローバル変数名が、衝突してしまう可能性が出てきます。

サブルーチン名およびグローバル変数名が衝突すると、どちらか片方のファイルの名前を変更しなければなりません。そうすると、その名前を参照する部分を、修正して回らなくてはならず、非常に面倒です。

これを避ける為に、例えばファイル名を名前の先頭に付加して、サブルーチン名やグローバル変数名を作ることが考えられます。こうする事で、個々のファイルの名前が衝突しません。

例えば、2つのAJANファイルがあると仮定します。

HOGE.AJN

```
' 連想配列から取り出したキー配列
LIST KEYS$

SUB DICT_LS(ARY AS DICT)
KEYS$ = GET_DICT_KEYS(ARY)
FOR I=0 TO UBOUND(KEYS$)
  PRINT KEYS$(I); "="; ARY(KEYS$(I))
NEXT I
END SUB
```

FUGA.AJN

```
' 連想配列から取り出したキー配列
LIST KEYS$

SUB DICT_LS(ARY AS DICT)
KEYS$ = GET_DICT_KEYS(ARY)
KEYS$ = ONEDIM SORT$(KEYS$)
FOR I=0 TO UBOUND(KEYS$)
  PRINT KEYS$(I); "="; ARY(KEYS$(I))
NEXT I
END SUB
```

それぞれのファイルが、同じ名前のサブルーチンの為、以下のように、2つのファイルをINCLUDEすると、エラーとなります。

```
INCLUDE "HOGE.AJN"
INCLUDE "FUGA.AJN"

DICT ARY
ARY("hoge") = 123
ARY("fuga") = 456
ARY("test") = 789
CALL DICT_LS(ARY)      ' サブルーチン名が衝突してしまう
```

これを避ける為に、関数名およびグローバル変数名の前に、ファイル名を付加してみましょう。

HOGE.AJN

```
' 連想配列から取り出したキー配列
LIST HOGE_KEYS$

SUB HOGE_DICT_LS(ARY AS DICT)
HOGE_KEYS$ = GET_DICT_KEYS(ARY)
FOR I=0 TO UBOUND(HOGE_KEYS$)
PRINT HOGE_KEYS$(I); "="; ARY(HOGE_KEYS$(I))
NEXT I
END SUB
```

FUGA.AJN

```
' 連想配列から取り出したキー配列
LIST FUGA_KEYS$

SUB DICT_LS(ARY AS DICT)
FUGA_KEYS$ = GET_DICT_KEYS(ARY)
FUGA_KEYS$ = ONEDIM SORT$(FUGA_KEYS$)
FOR I=0 TO UBOUND(FUGA_KEYS$)
PRINT FUGA_KEYS$(I); "="; ARY(FUGA_KEYS$(I))
NEXT I
END SUB
```

それぞれのファイルが、別々の名前のサブルーチンの為、以下のように、2つのファイルをINCLUDEしても、問題は発生しません。

```
INCLUDE "HOGE.AJN"
INCLUDE "FUGA.AJN"

DICT ARY
ARY("hoge") = 123
ARY("fuga") = 456
ARY("test") = 789
CALL HOGE_DICT_LS(ARY)    ' HOGE.AJNのサブルーチン名を呼び出している
```

このように、少しの工夫で、より見やすいプログラムを書くことができます。

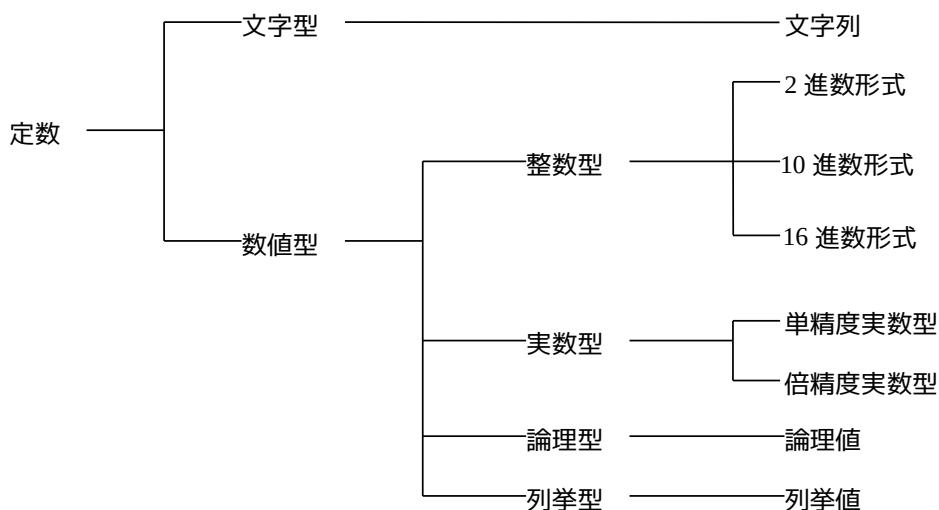
### 3.3 定数 / 変数 / 式 / 演算

#### 3.3.1 定数について

定数とは、それぞれ固有の値を持ったデータのことと、プログラム中に直接表記されるものです。定数は、次のように分類することができます。

定数の表記法は、型によってそれぞれ異なります。

表3-3 定数型一覧



## a) 文字型定数

文字型定数とは、文字をつなぎ、その前後を「"(ダブルクオーテーション)"」で囲んだ文字列データのことです。数値も、ダブルクオーテーションで囲うと、文字型定数になります。

「"(ダブルクオーテーション)"」そのものを文字型定数としたい場合、例のようにCHR\$(&H22)を使う方法またはダブルクオーテーションを2つ続けて書く方法を用います。

文字列同士は、「+」を使ってつなげることができます。

表3-4 文字列出力例

表記	出力例
PRINT "Good Morning"	Good Morning
PRINT "1 2 3 4 5 6 7 8 9"	1 2 3 4 5 6 7 8 9
PRINT "産業用パソコンA"	産業用パソコンA
PRINT "AJAN"	AJAN
PRINT CHR\$(&H22)+ "TEST"+ CHR\$(&H22)	"TEST"
PRINT """TEST""""	"TEST"
PRINT """"""	"

!	文字列は実際には複数のバイトデータから構成されており、見方を変えれば、文字単位に扱うのではなくバイト単位に情報を扱うデータ列として見る事ができます。このようにバイト単位で文字列を扱う時、「バイトデータ」列や「バイナリ文字列」と表現する事があります。
!	文字列の一文字はUTF-8形式で処理されています。 アルファベットの「A」は1文字1バイトですが、日本語の「あ」は1文字3バイトのデータで構成されます。
!	文字列の最大長は、メモリが確保できる範囲までです。

## b) 数値型定数

数値型定数とは、算術演算することのできる数値データを直接表記したものです。

数値型定数は大きく分けて、整数型、実数型、論理型、列挙型があります。

表記の種類	例
数値形式	255 -13965 3.1415
2進数形式	&B01101110
16進数形式	&HFF

### c) 整数型定数

整数型定数は、表記上の観点から、2進数・10進数・16進数の3つの形式に分類されます。

#### ■10進数形式(单精度整数型, 倍精度整数型)

私達が普段の生活で用いる、整数の数値表現です。

1桁を0から9までの10個の数字を用いて表現します。

单精度整数型の範囲は、-2,147,483,648～2,147,483,647です。

倍精度整数型の範囲は、-9,223,372,036,854,775,808～9,223,372,036,854,775,807です。

負の整数の場合、数値の前には必ずマイナス符号を付けなければなりません。

また、同じ範囲内の実数の後に「%」または「&」をつけると、小数点以下は丸め込まれ、整数型の定数とみなされます。

<例>

32767

-123

31100.5% ←整数を表す。(单精度)

31100.5& ←整数を表す。(倍精度)

#### ■2進数形式(单精度整数型)

数値の前に「&B」を付けた、0と1の数字の並びです。

コンピュータでI/O制御するなど、ビット単位に意味づけする際に、この記法がよく使われます。

範囲は、&B0～&B111111111111111111111111111111(32ビット)です。

<例>

&B10101110

#### ■16進数形式(单精度整数型, 倍精度整数型)

数値の前に「&H」を付けた、0からFまでの並びです。

2進数形式で冗長になる桁数をまとめる際に、よく使われます。

最後に「&」を付けると、倍精度整数型になります。

範囲は、&H0～&HFFFFFFF(倍精度整数型では&HFFFFFFFFFFFF&)です。

<例>

&H100

&HCFFF

&HF000000&

<b>!</b>	2進数形式または16進数形式で入力された数値は、PRINT等の出力文では10進数形式で出力されます。 10進数以外の形式で出力する時は、それぞれBIN\$, HEX\$を使って文字列として出力してください。
<b>!</b>	精度範囲外の定数値を与えようとしないでください。エラーもしくは不定の値が得られます。

## d) 実数型定数

実数型定数は、単精度実数型と倍精度実数型に分けられます。

### ■ 単精度実数型定数

有効桁7桁の精度をもつ実数のデータです。出力時は8桁目が四捨五入され、7桁以下で表示されます。単精度実数型の範囲は、-3.402823E+38～3.402823E+38です。

単精度実数型の定数には、表記上の分類により、次の3つの形式があります。

- ・ 7桁以下の実数
- ・ 最後に「!」をともなった数
- ・ Eを使った指数形式

<例>

3525.68

3.14!

-7.09E-06

### ■ 倍精度実数型定数

有効桁が15桁をもつ実数のデータです。出力の時は15桁以下で表示されます。

倍精度実数型の範囲は、-1.79769313486232E+308～1.79769313486232E+308です。

AJANでは、特に型宣言をしない場合、数値はすべて倍精度実数型として扱われます。

倍精度実数型の定数には、表記上の分類により、次の3つの形式があります。

- ・ 8桁以上の実数
- ・ 最後に「#」をともなった数
- ・ Eを使った指数形式

<例>

1234567.89

45678.9#

-1.09432E-58

<b>!</b>	精度範囲外の定数値を与えるようとしてください。エラーもしくは不定の値が得られます。
<b>!</b>	実数型は、コンピュータ内部では、固定ビット長の2進数のデータとして格納されており、整数型と異なり すべての値を正確に表現できず、近似値として表現されます。 このため、計算時に誤差が生じてしまう場合があります。  詳しくは、IEEE754浮動小数点数の規格およびコンピュータ数学に関する書籍を参照ください。 いくつかの事項に関しては、「7.3 Tips」章に事例紹介しています。
<b>!</b>	実数型は、表現可能な範囲に制限があります。 例えば、単精度実数および倍精度実数の正の最大値は、以下により与えることが出来ます。 (最大値を設定するには、値の丸めを勘案して、桁数を多めに指定する必要があります)

	' 単精度実数の正の最大値 <code>FLT_MAX! = 3.40282346638528859812E+38</code> ' 倍精度実数の正の最大値 <code>DBL_MAX# = 1.79769313486231570815E+308</code>
--	---

### e) 論理型定数

真偽を表すデータです。真偽はそれぞれ以下のように分類されます。

真	偽
TRUE	FALSE
0以外の値	0

整数型変数に代入した場合、TRUEは1、FALSEは0になります。

### f) 列挙型定数

整数型定数を任意の名前で扱うことができるデータ集合です。

「ENUM」～「END ENUM」コマンドで定義した列挙名を指示する事で、条件分岐等で実際の値を意識せずに処理できます。

<例>

<b>ENUM</b> SUNDAY MONDAY TUESDAY WEDNESDAY THURSDAY FRIDAY SATURDAY <b>END ENUM</b>
--

<b>PRINT</b> MONDAY
---------------------

### 3.3.2 変数について

変数は、プログラム中に使うデータをしまっておくことができる入れ物のことで、自由に名前(変数名)をつけることができます。

変数は、演算、参照等に使うことができます。

変数に値を入れる前は、数値変数の値は0、文字変数は空の文字列であるとみなされます。

#### a) 変数名

変数名は、英文字で始まる英数文字と「\_(アンダーバー)」で組み合わせた可変長の文字列で表します。

例えば、次の2つの変数は異なった変数名として解釈されます。

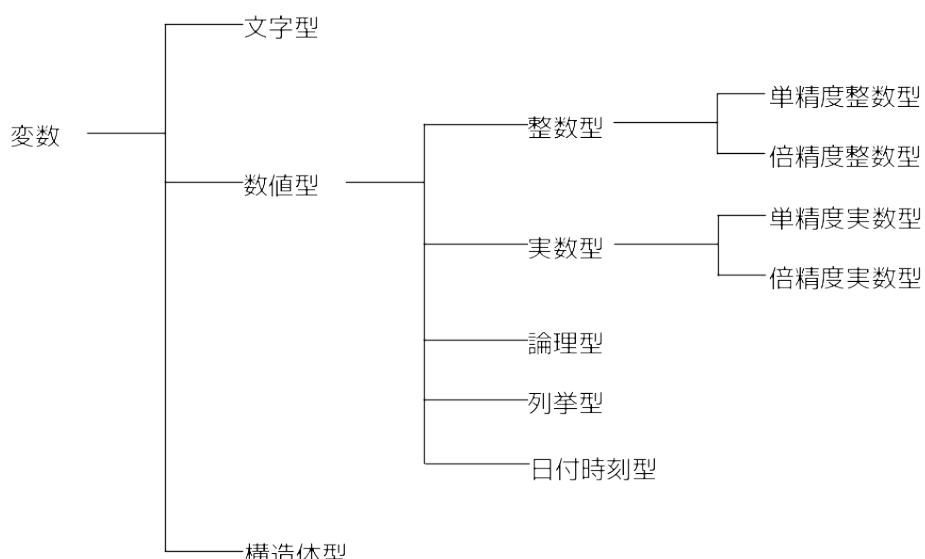
```
COUNTER_OF_TABLE_DATA_999888777666555_01
COUNTER_OF_TABLE_DATA_999888777666555_02
```

AJANのコマンド名や関数名は、変数名として定義できませんが、名前を含んだものは、変数名として定義できます。

#### b) 変数の型

変数の型にも定数と同様、入れるデータに応じた型があります。

変数に値を入れるためにには、代入文等を用いますが、いずれの場合にも変数の型は、入れるデータの型と一致していかなければなりません。



文字型と数値型は、文字型の変数名の最後に「\$」を付けることによって区別します。

接尾辞が省略されている場合は倍精度実数型の変数となります。

型の種類における変数名の接尾辞と例は下記の表のようになります。

型の種類		変数名の接尾辞	例
文字型		\$	X\$
数値型	整数型	% 単精度整数型(-2147483648~2147483647) & 倍精度整数型 (-9223372036854775808~9223372036854775807)	X% X&
	実数型	! 単精度実数型 (有効桁数7桁, -3.402823E+38 ~ 3.402823E+38) # 倍精度実数型(有効桁数15桁, -1.79769313486232E+308 ~ 1.79769313486232E+308)	X! X#
	論理型	BOOLコマンドで変数を宣言します。	X
	列挙型	ENUMコマンドで変数を宣言します。	X
	日付時刻型	DATETIMEコマンドで変数を宣言します。	X
構造体型		STRUCTコマンドで変数を宣言します。	X

<b>!</b>	精度の低い型を精度の高い型に変換すると、変換前の型の有効桁数より小さい部分に無効な数値が現れます。 (変換前の型の有効桁数+1桁目で四捨五入すると、正しい値になります。)
<b>!</b>	精度の高い型を精度の低い型に変換すると、変換後の型の有効桁数に丸められます。 (変換後の型の有効桁数+1桁目で四捨五入されます。)
<b>!</b>	整数型の変数に、NaN値(非数)またはINF値(無限大)は、与えないでください。エラーまたは不正な結果が得られます。

### c) 論理型変数

真偽を表すデータを格納する变数型です。「BOOL」コマンドにより宣言します。

<例>

```
BOOL A = TRUE
```

<b>!</b>	論理型の変数に、NaN値(非数)を、与えないでください。エラーまたは不正な結果が得られます。
----------	--

## d) 日付時刻型変数

日時情報を表すデータを格納する変数型です。「DATETIME」コマンドにより宣言します。

<例>

```
DATETIME A = "2017/12/24 18:00:00"
```

"年/月/日"および"時:分:秒"で表された文字列形式の日付時刻を、この変数型に格納したり、文字列形式で取り出したり、日時の加減算が可能です。

<例>

```
DATETIME A = "2017/12/24"
A = A + 1      ' 1日分を加算
B$ = A
? B$          ' "2017/12/25 00:00:00"と表示されます
```

<b>!</b>	日付時刻型の内部形式は、倍精度実数と同じ形式で、整数部を日付、小数部を時間の組で情報を格納します。
<b>!</b>	日付時刻型の変数に、NaN値(非数)またはINF値(無限大)は、与えないでください。エラーまたは不正な結果が得られます。

## e) 構造体型変数

構造体とは、色々な種類の互いに関連するデータをまとめて格納できるデータ型です。

「DEFINE STRUCT」～「END STRUCT」により構造を定義し、「STRUCT」コマンドにより変数を宣言します。

<例>

```
DEFINE STRUCT MEMBER      ' MEMBERが構造体名
  NAME$                  ' NAME$というメンバ名
  AGE                     ' AGEというメンバ名
END STRUCT

STRUCT MEMBER V           ' 構造体名MEMBERを元に、V変数を作ります
V.NAME$="太郎"
V.AGE=20
```

「DEFINE STRUCT」の後に続けて、構造体名と呼ばれる名前を宣言します。

「DEFINE STRUCT」～「END STRUCT」の間に、構造体の中に格納したい変数名を列挙します。これを、メンバ名と呼びます。

構造体を定義したら、「STRUCT」の後に、構造体名を指定し、続けて変数名を宣言します。

これにより、構造体として定義した各メンバ名が、まとめて使用できるようになります。

このように、構造体を用いると、いくつか関連性のあるデータを、まとめて構造化して扱うことが可能となります。

	構造体のメンバ名に、AJANのコマンド名や関数名を使用することはできません。
---	--

### 3.3.3 配列、連想配列について

#### a) 配列

配列とは、複数の変数(数値や文字列)を格納するためのもので、変数同様、任意に名前を付けることができます。

Xという変数の箱を5個作る場合、以下のような宣言が必要となります。

**DIM X(4)**

X(0)

X(1)

X(2)

X(3)

X(4)

この場合、配列はX(0)からX(4)までの5つの変数の箱となります。

X(0) = 100

X(1) = 5000

:

上記のように代入していきます。

多次元の配列も使用できます。次は、2次元配列の宣言例を示します。

**DIM Y(3, 2)**

Y(0, 0)

Y(1, 0)

Y(2, 0)

Y(3, 0)

Y(0, 1)

Y(1, 1)

Y(2, 1)

Y(3, 1)

Y(0, 2)

Y(1, 2)

Y(2, 2)

Y(3, 2)

配列の情報を得るには、主に以下の関数が使えます。

CDIM関数	配列の次元数を得ます。 例えば、「DIM Y(3, 2)」のような2次元の配列に対して、「CDIM(Y)」の得られる値は「2」です。
LDIM関数	配列の要素数を得ます。 例えば、「DIM X(4)」のような1次元の配列に対して「LDIM(X)」の得られる値は「5」です。
UBOUND関数	配列の指定した次元の添字最大値を得ます。 例えば、「DIM X(4)」のような1次元の配列に対して「LDIM(X)」の得られる値は「4」です。

## b) 範囲配列

配列に対して、添字のある範囲を指定して一括して代入したり、配列を受け取る関数や命令に対して、変数を指定するのではなく値を列挙して渡す事ができます。

これを範囲配列と呼びます。範囲配列は、構造体型の配列以外に使用可能です。

例えば、下記のように記載すると、X(1)～X(3)に11, 12, 13と一緒に値が代入できます。

代入式の右辺や関数のパラメータなど、式として範囲を指定するには、[ ] で指定する範囲を明記する必要があります。

```
DIM X(5)
X( 1 to 3 )=[ 11 to 13 ]
```

他にも、下記のようにセミコロン( ; )やカンマ( , )で値を連続で記載して値を代入することができます(構造体型以外)。

配列の添字内では、セミコロンは値の配列、カンマは次元の区切りを意味します。

範囲配列を使用する場合、右辺の先頭に指定している変数の値の型がすべてに依存してきます。

また、右辺で指定した範囲配列は、内部的には1次元配列として扱われます。

```
DIM X(5)
X( 1;2;3 )=[ 11;12;13 ]
```

構造体も配列にすることができますが、まとめて値などを代入することはできません

```
DEFINE STRUCT OFFICE
  NAME$
  ADDRESS$
END STRUCT

STRUCT OFFICE  OFFICE_LIST(5)
```

また、STRUCT命令やBOOL命令のように、変数名の前に型を指定するものは、DIMは不要です。

```
BOOL  A    ' 論理型の変数定義
BOOL  B(5) ' 論理型の配列変数定義
```

### c) 固定長の配列と可変長の配列

「DIM」コマンドは、固定長の配列を宣言しますが、配列の要素数が不定ないしは可変の場合、「LIST」コマンドで可変長の配列を宣言できます。

例えば、文字列に対して、指定した区切りに従い分割する「SPLIT\$」関数を用いた例を示します。

まず、「DIM」コマンドの事例です。

```
DIM D$(4)
D$ = SPLIT$("1,2,3,4,5", ",")
? D$
D$ = SPLIT$("1,2,3,4,5,6,7,8,9,10", ",")
```

? D\$

実行結果>

```
[ 1, 2, 3, 4, 5 ]
[ 1, 2, 3, 4, 5 ]
```

次に、「LIST」コマンドの事例です。

```
LIST D$
D$ = SPLIT$("1,2,3,4,5", ",")
? D$
D$ = SPLIT$("1,2,3,4,5,6,7,8,9,10", ",")
```

? D\$

実行結果>

```
[ 1, 2, 3, 4, 5 ]
[ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ]
```

「DIM」コマンドの場合、配列の要素数が5に固定される為、大きな要素数の配列を代入しても、結果は宣言時の要素数に切り落とされます。

しかし、「LIST」コマンドの場合、大きな要素数の配列を代入した時、結果は代入元と同じ要素数を受け取る事が可能です。

「LIST」コマンドで、変数を宣言した後、次元数や要素数を変更したい場合は、「REDIM」コマンドを使用します。

また、キュー(待ち行列)のように、配列の要素数を増やしたり削除したい場合は、「ONEDIM INSERT」、「ONEDIM REMOVE」コマンドなどを使用します。

例えば、以下のように。

```
LIST ARY
ARY = [ 1; 2; 3; 4; 5; 6 ]      ' ARY配列に一気に代入
? "代入時:"; ARY
REDIM PRESERVE ARY(1, 1)      ' 元の値をなるべく保持しつつ2次元配列へ
? "REDIM PRESERVE使用時:"; ARY
REDIM ARY(3)                  ' 中身をクリアして1次元配列へ
? "REDIM使用時:"; ARY
```

実行結果>

```
代入時: [ 1; 2; 3; 4; 5; 6 ]
REDIM PRESERVE使用時: [[ 1, 2 ], [ 3, 4 ]]
REDIM使用時: [ 0, 0, 0 ]
```

または、

```
LIST ARY
ONEDIM INSERT ARY, -1, [ 1; 2; 3; 4; 5 ]      ' ARY配列の後尾に追加
? "要素数>"; LDIM(ARY)
? "中身>"; ARY
ONEDIM REMOVE ARY, 0, 3      ' ARY配列の先頭から3要素を削除
? "要素数>"; LDIM(ARY)
? "中身>"; ARY
```

実行結果>

```
要素数=6
中身=[ 0; 1; 2; 3; 4; 5 ]
要素数=3
中身=[ 3; 4; 5 ]
```

可変長の配列は、配列変数への代入により、右辺の配列の次元、要素数に合わせて容易に変化します。  
変化させたくない場合は、添字を使って要素毎にアクセスしてください。

例えば、以下の例では、最初「REDIM」コマンドで 2次元配列変数に可変させていますが、  
次の行で、配列変数自体に別の配列(ここでは範囲配列)を代入した結果、内容が変化しています。

```
LIST ARY
REDIM ARY(2,3)          ' 2次元配列に可変した
? "次元数:"; CDIM(ARY)
? "全要素数:"; LDIM(ARY)
ARY = [ 1; 2; 3; 4; 5; 6 ]   ' ARY配列に範囲配列を一気に代入
? "次元数:"; CDIM(ARY)
? "全要素数:"; LDIM(ARY)
```

実行結果>

次元数:2

全要素数:12

次元数:1

全要素数:6

このように、可変長配列は非常に便利な機能ですが、使い方に注意して使用ください。

## d) 連想配列

### (1) 連想配列とは

連想配列は、文字列で要素の添え字を指定することができる配列です。

但し、連想配列の要素は順番が必ずしも指定した順序になるとは限りません。

連想配列を使うと、配列の要素の添え字指定が分かりやすくなります。

### 一般的な配列

<code>DIM A\$(3)</code>	' 要素数が4の配列を宣言
<code>A\$(0) = "TEST MESSAGE"</code>	' 0番目の要素に文字列を代入

### 連想配列

<code>DICT A\$</code>	' 連想配列を宣言
<code>A\$("NAME") = "INTERFACE"</code>	' 要素「NAME」に文字列を代入

配列の情報を得るには、「CDIM」関数や「LDIM」関数を使用しましたが、

連想配列の情報や操作を行うには、主に以下の関数と命令が使えます。

LDICT関数	連想配列に登録されているキーの総数を取得します。
GET_DICT_KEYS\$関数	連想配列のキー一覧を文字列配列として得ます。
HAS_DICT_KEY関数	連想配列に対して、指定したキーが存在するか否かを得ます。
DEL_DICT_KEY	連想配列に対して、指定したキーに紐付けされたデータを削除します。
CLEAR_DICT	連想配列の内容をクリアします。

### (2) 連想配列と一般的な配列の違い

連想配列と一般的な配列は以下が異なります。

項目	一般的な配列	連想配列
要素数	固定長(可変長配列を除く)	可変長
順序	添え字の順に固定	不定
指定の要素がないときの動作	エラー	新しく要素を生成
範囲配列指定	可	不可
定数をまとめて代入	可	不可
要素の指定	数値	文字列

### (3) 連想配列の配列

複数の連想配列をまとめた連想配列の配列を定義することができます。

これにより、連想配列のリストを表現することができます。

**DICT A\$(2)** '要素数3の配列を宣言

上記を実行すると、以下のような変数が作成されます。

A\$

連想配列0
連想配列1
連想配列2

各要素は独立した連想配列です。連想配列に対して操作をするには、どの連想配列に対して行うかを指定します。連想配列の指定とキーの指定はそれぞれ丸カッコ「( )」でくくります。

<b>A\$(1)(“NAME”)</b> = “INTERFACE” ’連想配列1のNAMEに文字列を代入
--

また、LIST命令と組み合わせることで可変長の連想配列リストを宣言することができます。

例えば以下のように、

<b>LIST DICT A\$</b>
----------------------

連想配列の配列に対して、「LDICT」関数を適用するには、配列の添え字を指定して要素を限定して行ってください。

例えば以下のように。

<b>DICT A\$(2)</b> A\$(0)(“NAME”) = “yamamoto” A\$(0)(“GENDER”) = “man” A\$(1)(“NAME”) = “hirosima” A\$(1)(“GENDER”) = “woman” A\$(1)(“OPT”) = “hello” ? LDICT(A\$(0)) ? LDICT(A\$(1))
---

実行結果>
-------

2
---

3
---

「LIST DICT」コマンドを使用すると、連想配列の可変長配列を使用できます。

その際、「ONEDIM INSERT」、「ONEDIM REMOVE」コマンドなどを使用する事で、配列の要素数を増減できます。

例えば以下のように。

```
LIST DICT ARY$  
PRINT "配列の要素数="; LDIM(ARY$)  
ARY$(0)("hoge") = "fuga"  
PRINT "中身="; ARY$  
  
DICT TMP$  
ONEDIM INSERT ARY$, -1, TMP$  
PRINT "配列の要素数="; LDIM(ARY$)  
ARY$(1)("hello") = "interface"  
ARY$(1)("world") = "corporation"  
PRINT "中身="; ARY$  
' ↑PRINT時の中身の表示順番は、実行時により変化する可能性があります
```

実行結果>

配列の要素数=1

中身=[ (hoge)="fuga" ]

配列の要素数=2

中身=[ (hoge)="fuga", (world)="corporation", (hello)="interface" ]

### 3.3.4 式と演算について

#### a) 算術演算子

演算内容	記号	記号の読み方例	書き方例	書式例と結果
加算	+	プラス	X + Y	5+2は7
減算, 負号	-	マイナス	X - Y	5-2は3
乗算	*	アスタリスク	X * Y	5*2は10
整数の除算	\	バックスラッシュ	X \ Y	5\2は2
実数の除算	/	スラッシュ	X / Y	5/2は2.5
指数演算	^	ハット、キャレット	X ^ Y	5^2は25
剰余算	MOD	モッド、モジュラス	A MOD B	5 MOD 2は1

!	整数の除算記号( \ )は、一部のキーボードでは「¥」キーで入力可能です。(一般に日本語キーボードと呼ばれるものが該当します)
---	---

#### b) 関係演算子

演算内容	記号	記号の読み方例	書き方例	書式例と結果
等しい	=	イコール	X = Y	XとYは等しい
等しくない	◊	大なり小なり、ノットイコール	X ◊ Y	XとYは等しくない
大小比較(小なり)	<	小なり	X < Y	XはYより小さい
大小比較(大なり)	>	大なり	X > Y	XはYより大きい
大小比較(小なりイコール)	<=	小なりイコール	X <= Y	XはY以下
大小比較(大なりイコール)	>=	大なりイコール	X >= Y	XはY以上

### c) 論理演算子

演算内容	記述内容	読み方例	書き方例
否定	NOT	ノット	NOT X
論理積	AND	アンド	X AND Y
論理和	OR	オア	X OR Y
排他的論理和	XOR	エックスオア	X XOR Y
包含	IMP	アイエムピー	X IMP Y
同値	EQV	イーキューブイ	X EQV Y

※ ビット演算とブール演算のどちらでも扱うことができます。

論理演算子の書式例を、以下に示します。

NOTの演算結果例:

式	結果
NOT TRUE	FALSE
NOT FALSE	TRUE

AND、OR、XOR、IMP、EQVの演算結果例(式1 論理演算子 式2):

式1	式2	AND結果	OR結果	XOR結果	IMP結果	EQV結果
TRUE	TRUE	TRUE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
FALSE	TRUE	FALSE	TRUE	TRUE	TRUE	FALSE
FALSE	FALSE	FALSE	FALSE	FALSE	TRUE	TRUE

### d) シフト演算子

ビット演算は論理演算子の他に右シフト演算、左シフト演算を用意しています。

演算内容	記述内容	読み方例	書き方例	書式例と結果
右シフト	>>	右シフト	&H10 >> 1	&H10の1ビット右シフトは&H8
左シフト	<<	左シフト	&H10 << 1	&H10の1ビット左シフトは&H20

シフト演算は左辺をビット列とみなして右辺分、左右に列をずらします。

AJANのシフト演算は論理シフトです。よって、ずれた分のビットは左右限らず、0が入ります。

数値としては、一般的に1ビット右シフトすると1/2,1ビット左シフトすると2倍になります。

!	ビット演算は整数型のみ値が保証されます。実数型は不定な値を得る恐れがあります。。
---	--

## e) 演算の優先順位

各演算には優先順位があり、以下の順番で処理されます。

高い ↑	優先順位	演算子
	1	括弧で括られた式
	2	関数
	3	指数演算 ( ^ )
	4	負号 ( - )
	5	乗算 ( * ) 、実数の除算 ( / )
	6	整数の除算 ( \ )
	7	整数の剰余 (MOD)
	8	加算 ( + ) 、減算 ( - )
	9	シフト演算子( >>, << )
	10	関係演算子 ( =, <, > など)
	11	NOT
	12	AND
	13	OR
	14	XOR
	15	IMP
↓ 低い	16	EQV

### 3.4 特殊記号について

AJANで使用できる文字は次の通りです。

演算子(+, -, \*, /等)の他にも特別な意味を持つ記号があります。

記号	機能
コロン(:)	<ul style="list-style-type: none"> <li>マルチステートメントの区切りとして使います。 ＜例＞ <code>A=B+C:PRINT A</code></li> <li>ラベル名の終端に付けます。 ＜例＞ <code>LABEL:</code></li> </ul>
コンマ または カンマ( , )	パラメータが並ぶ場合、その区切りとして使います。 ＜例＞ <code>BOOL A, B</code>
セミコロン( ; )	PRINT等の出力文中でパラメータが並ぶ場合、その区切りとして使います。 ＜例＞ <code>PRINT "A=";A</code>
アポストロフィ または シングルクオーテーション(' )	REM(コメント文)の代用として使います。 または、 3つ連続で記入すると複数行を1つの文として解釈します。
クエッショニングマーク( ? )	PRINTの代用として使います。
エクスクラメーションマーク( ! )	単精度実数として使います。
シャープ( # )	倍精度実数として使います。
パーセント( % )	単精度整数として使います。
アンド または アンパサンド( & )	倍精度整数として使います。
ダラー(\$)	文字列変数として使います。
スペース( )	日本語入力時の幅広のスペースと区別するために、半角スペース、全角スペースと区別して呼ぶことがあります。 また、原則として文字型定数に含まれているスペース以外のスペースは無視されますが、コマンドの名前の直後には必ず半角スペースを入れなければなりません。
アンダースコア または アンダーバー、アンダーライン( _ )	行の末尾に記述すると、次の行への継続行として認識します。

### 3.5 戻り値について

コマンドの中で、関数と呼ばれるコマンドは戻り値を返します。

戻り値は変数への代入、命令や関数のパラメータとして使用する事ができます。

特別な説明がない限りコマンド名の添字によって戻り値の型を判定する事ができます。

<例>

```
R = ERR
```

関数名に添字が付いてないため数値型の値を返します。

<例>

```
R$ = ERM$
```

関数名に\$の添字が付いているため文字列型の値を返します。

<例>

```
DATETIME A
```

```
A = CDATETIME(B)
```

関数名に添字が付いていませんが、日付時刻型の値を返します。

## 3.6 特殊な構文

AJANでは、プログラムを作成する上で便利な構文があります。

### a) 1つの文を複数行に渡って記述する

行の末尾に「\_」を入力すると、複数行を1つの文として扱うことができます。

行の末尾に「\_」がある限り、その次の行までは1行として解釈されます。

<例>

```
PRINT _  
A, _  
B, _  
C, _  
D, _  
_  
E
```

上記プログラムを実行した結果は、「PRINT A,B,C,D,E」と同じです。

空自行や改行はないものとして処理されます。

コメント文でもこの機能は有効ですので、複数行のコメントも簡単に書くことができます。

<例>

```
REM _  
ここはコメントです。_  
複数行に渡ってコメントを書くことができます。
```

同一行に複数記述することはできません。エラーになります。

<例>

```
PRINT A, B, C, D, E _ _
```

	<p>「_」は行の末尾に無いと解釈されないため、1行に複数回書くことはできません。</p> <p>&lt;違反例&gt;</p> <pre>PRINT A;B;C; _ D; _ E; _ F</pre> <p>&lt;適合例&gt;</p> <pre>PRINT A;B;C; _ D; _ E; _ F</pre>
	<p>「_」の直前は半角スペース、直後は改行が必要です。</p> <p>&lt;違反例1&gt;</p> <pre>S\$ _ = "TEST"</pre> <p>&lt;適合例1&gt;</p> <pre>S\$ _ = "TEST"</pre> <p>&lt;違反例2&gt;</p> <pre>S\$ _ 'コメント = "TEST"</pre> <p>&lt;適合例2&gt;</p> <pre>S\$ /* コメント */ _ = "TEST"</pre>
	<p>INCLUDE文は、「_」の処理が終わってから解釈されます。よって、複数のファイルにまたがって「_」を有効にすることはできません。</p> <p>&lt;違反例&gt;([内はファイル名です)</p> <pre>[ MAIN.AJN ] S\$ _ INCLUDE "A.AJN" PRINT S\$ [ A.AJN] = _ "TEST"</pre> <p>&lt;適合例&gt;</p> <pre>[ MAIN.AJN ] INCLUDE "A.AJN" PRINT S\$ [ A.AJN] '同じファイル中で完結させる S\$ _</pre>

	= _ "TEST"
!	<p>文字列の中に「_」を入れたい場合は、末尾にかかるないように注意してください。</p> <p>&lt;例1&gt;  <code>S\$ = "A _ _ B"</code>        ' S\$に 「A _ _ B」という文字列が入ります。</p> <p>&lt;例2&gt;  <code>S\$ = "A _ _B"</code>        ' S\$に 「A _B」という文字列が入ります。</p>
!	<p>文字列の中で「_」を使って複数行にすると、行間にスペースが入ります。        スペースをいれずに結合するには「+」を使用してください。</p> <p>&lt;例1&gt;  <code>S\$ = "A _ B"</code>        ' S\$に 「A B」という文字列が入ります。</p> <p>&lt;例2&gt;  <code>S\$ = "A" + _ "B"</code>        ' S\$に 「AB」という文字列が入ります。</p>

## b) 1行の一部分だけをコメントにする

通常、コメント(REM命令や「'」)は、該当する命令を書いたところから行末まで有効です。  
「/\*」と「\*/」でくくりだすことで、1行の一部分だけをコメントすることができます。

<例>

```
PRINT A; /* ここはコメントです */ B
```

上記は、「PRINT A; B」と記入したことと同じ動作をします。

この機能を使うと、多くのパラメータを記述する際に、1つの文を複数行に渡って記述する(→「3.6」章の「1つの文を複数行に渡って記述する」)機能と組み合わせることで、説明をパラメータごとに記入できるようになります。

<例>

```
PRINT _  
TO$, /* あて先 */ _  
FROM$, /* 送信元 */ _  
SUBJECT$, /* タイトル */ _  
DATE$, /* 日付 */ _  
TIME$, /* 時刻 */ _  
MAIN$ /* 本文 */
```

	<p>「/* */」は1行に書いてください。行をまたいで記入することはできません。</p> <p>&lt;違反例&gt;</p> <pre>/*  コメント */</pre> <p>&lt;適合例&gt;</p> <pre>/* コメント */</pre>
	<p>「/* */」は文字列の中で使用することはできません。そのまま文字列として認識されます。</p> <p>。</p> <p>&lt;例&gt;</p> <pre>A\$ = " 内容は /* コメント */ ありません。"</pre> <p>この場合、A\$には、「内容は /* コメント */ ありません。」と入ります。</p> <p>コメントとしたい場合は結合式「+」を使ってください。</p> <p>&lt;例&gt;</p> <pre>A\$ = " 内容は" /* コメント */ + " ありません。"</pre>

### c) 改行を含めた文字列をそのまま記述する

**文字型定数に限り**、文字列としたいところの最初と最後に「"(シングルクオーテーション3つ)"」でくくると、改行や空白行も含めて解釈することができます。この機能を「ヒアドキュメント」と呼びます。

<例>

```
S$ = ''' 1,2
      3,4

      5,6 '''
```

S\$には、以下が入ります。

```
「1,2
  3,4
  <空白行>
  5,6」
```

ヒアドキュメントはコメントより優先して動作します。

応用すると、複数行のコメントも記述できます。

<例>

```
REM '''
This is Comment.
ここはコメントです。
```

空白行を入れたコメントもかけます。

```
'''
```

ヒアドキュメントの中に変数や式を入れることはできません。全てそのまま文字列として解釈されます。

<違反例>

```
A$ = "TEST1"
B$ = "TEST2"
PRINT '''A$+
B$'''
```

「A\$+
 B\$」とそのまま出力されます。

<適合例>

```
A$ = "TEST1"
B$ = "TEST2"
PRINT A$ + _
B$
```

「\_」を使って記述すると、式が使用できます。

	<p>この例では、「TEST1TEST2」と表示されます。</p>
!	<p>1行に複数のヒアドキュメントを併記することはできません。また、行をまたがないでヒアドキュメントを終わらせるることはできません。</p> <p>&lt;違反例1&gt;</p> <pre>PRINT ''' This is message. これはメッセージです ''' ; ''' 続きはこちら next message is here. '''</pre> <p>&lt;適合例1&gt;</p> <pre>PRINT ''' This is message. これはメッセージです ''' ; _ ''' 続きはこちら next message is here. '''</pre> <p>「_」を使って記述すると、複数のヒアドキュメントを使用することができます。</p> <p>&lt;違反例2&gt;</p> <pre>PRINT ''' This is message. '''</pre> <p>&lt;適合例2&gt;</p> <pre>PRINT "This is message."</pre> <p>この場合はヒアドキュメント不要です。「"(ダブルクオーテーション)"」でくくると文字列として解釈されます。</p>

### 3.7 制限事項

以降では、現在のAJANの構文に関する、制限事項について列挙します。

#### a) 配列の添え字に、更に配列の添え字を与えることは出来ない

配列の添え字に、更に配列の添え字を与えないでください。

例えば、以下のようなコードは、コンパイルできません。

```
DIM A(5)
DIM B(5)
A = [1 to 6]
B = [1 to 6]
PRINT A(B(1))      ' エラーになります
```

この場合、一旦単純変数に代入した後に、配列の添え字に与えるようにしてください。

```
DIM A(5)
DIM B(5)
A = [1 to 6]
B = [1 to 6]
IDX = B(1)          ' 一旦、添え字先の情報を単純変数に代入
PRINT A(IDX)        ' 配列の添え字を使用
```

#### b) 未初期化のローカル変数へのアクセスは動作不定

ユーザ一定義のサブルーチン内で使用可能なローカル変数は、「LOCAL」コマンドなどで宣言&初期化してから使用してください。(サブルーチンに渡すパラメータは、AJANコンパイラが初期化します)

例えば、以下のように、「LOCAL」コマンドをスキップしてアクセスすると、ローカル変数は初期化されてないため、動作は不定となります。

```
SUB HOGE()
    GOTO SKIP      ' 次行のLOCAL文をスキップしている
    LOCAL VAR1
    SKIP:
    PRINT VAR1    ' LOCAL文を通過せずに、ローカル変数 VAR1 をアクセスしている
END SUB
```

### c) グローバル変数の宣言は、プログラムの先頭にまとめる事を推奨する

「DIM」や「LIST」コマンドなど、グローバル変数を宣言する命令は、ユーザー定義のサブルーチンの中でも使用できます。

サブルーチンの中で、グローバル変数を宣言しておいて、「INCLUDE」コマンドで外部ファイルに括りだすと、判りにくいプログラムが出来てしまう可能性があります。

この為、グローバル変数の宣言および初期化は、プログラムの先頭部分にまとめて定義する事を お奨めします。

お奨めの例:

```
' 先頭部分で変数を宣言 & 初期化
LIST ARY
ARY = 123

SUB TEST()
FOR I=1 TO 10
    ' 配列の後尾に値を追加 - 先頭部分の定義を見ているので ARYが配列と判る
    ONEDIM INSERT ARY, -1, I
NEXT I
END SUB

CALL TEST()
PRINT ARY
```

混乱する例:

```
SUB TEST()
    ' サブルーチンの中で変数を宣言
    LIST ARY

    FOR I=1 TO 10
        ' 配列の後尾に値を追加
        ONEDIM INSERT ARY, -1, I
    NEXT I
END SUB

    ' SUB内の変数定義を見てないと、ARYが配列と判りにくい
ARY = 123
CALL TEST()
PRINT ARY
```

## 3.8 コマンドリファレンスの説明

コマンドリファレンスとはコマンドの辞書のことで、どのようなコマンドがあり、そのコマンドで指示を与えた場合、コンピュータがどのように動くかが記載されています。

コマンドリファレンスはコマンドの種類により、「xxxコマンド編」というように分冊されています。

コマンドの種類	コマンドリファレンス
表示, 入力, 変数, 演算, 文字列操作, ファイル操作, 分岐, ループ, 配列, スレッド	標準コマンド編
拡張コマンド	拡張コマンド編, 拡張コマンド編Vol.2
システム監視制御	システム監視コマンド編
ネットワーク	ネットワークコマンド編
GUI	GUIコマンド編
数学処理	数学統計コマンド編
エッジサーバ	エッジサーバコマンド編
CAN / CANFD	IOコマンド CAN編
シリアル通信	IOコマンド シリアル通信編

### 3.8.1 コマンドリファレンスやマニュアルを表示する

標準コマンドのコマンドリファレンスマニュアルを表示するには、

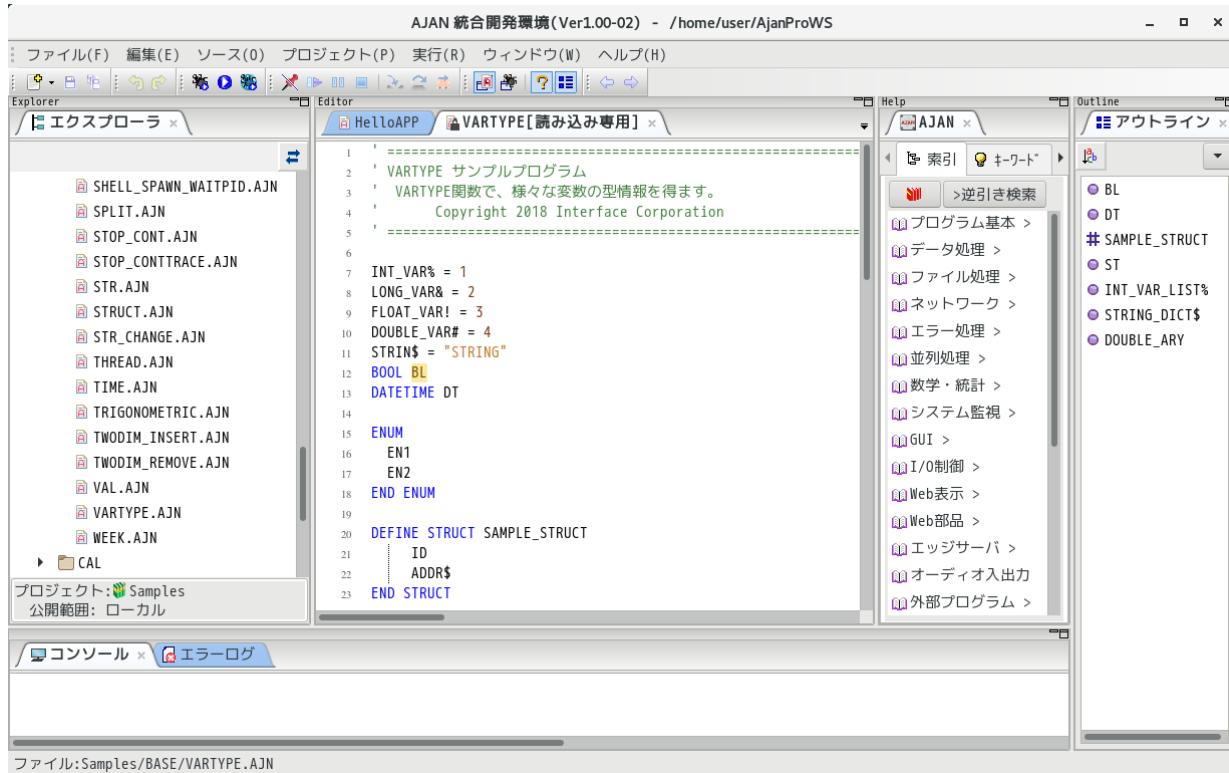
- ・AJAN 統合開発環境のヘルプウインドウを使用するか、
- ・デスクトップから左上の「アプリケーション」 → 「Interface」 → 「AJAN」 → 「AJANヘルプファイル」 → 「1:標準コマンド編」を選択すると、ドキュメントが表示されます。

## 第4章 AJAN 統合開発環境の使い方

### 4.1 AJAN統合開発環境とは

AJAN 統合開発環境は、AJANでプログラムを開発する際に便利な様々な機能を持った開発ツールのことです。一般的にはIDE(Integrated Development Environment)とも呼ばれています。

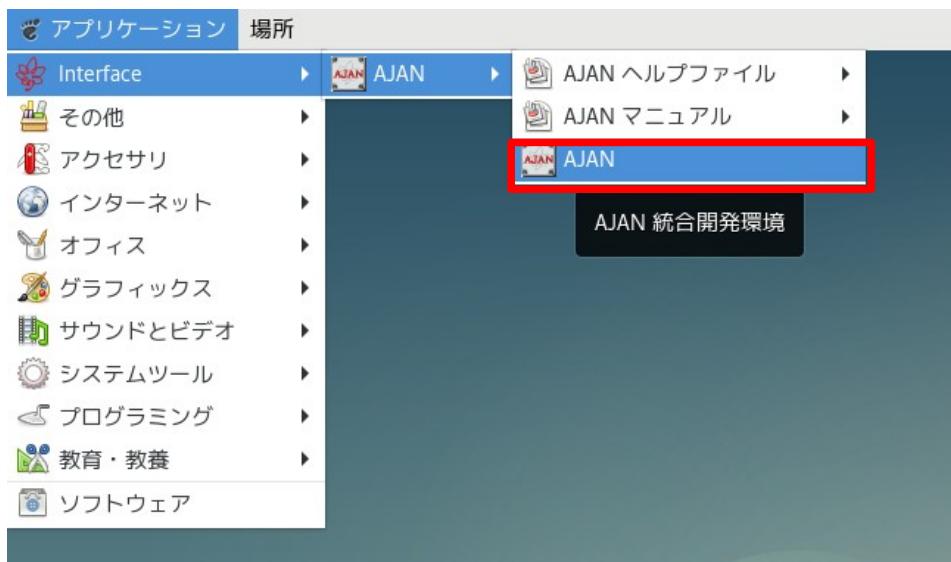
編集や実行や各種設定をほとんどマウスで操作できます。また、AJANプログラムソースが常に表示されているので、実行結果を確認しながら開発しやすいようになっています。



## 4.2 起動と終了方法

### 4.2.1 起動方法

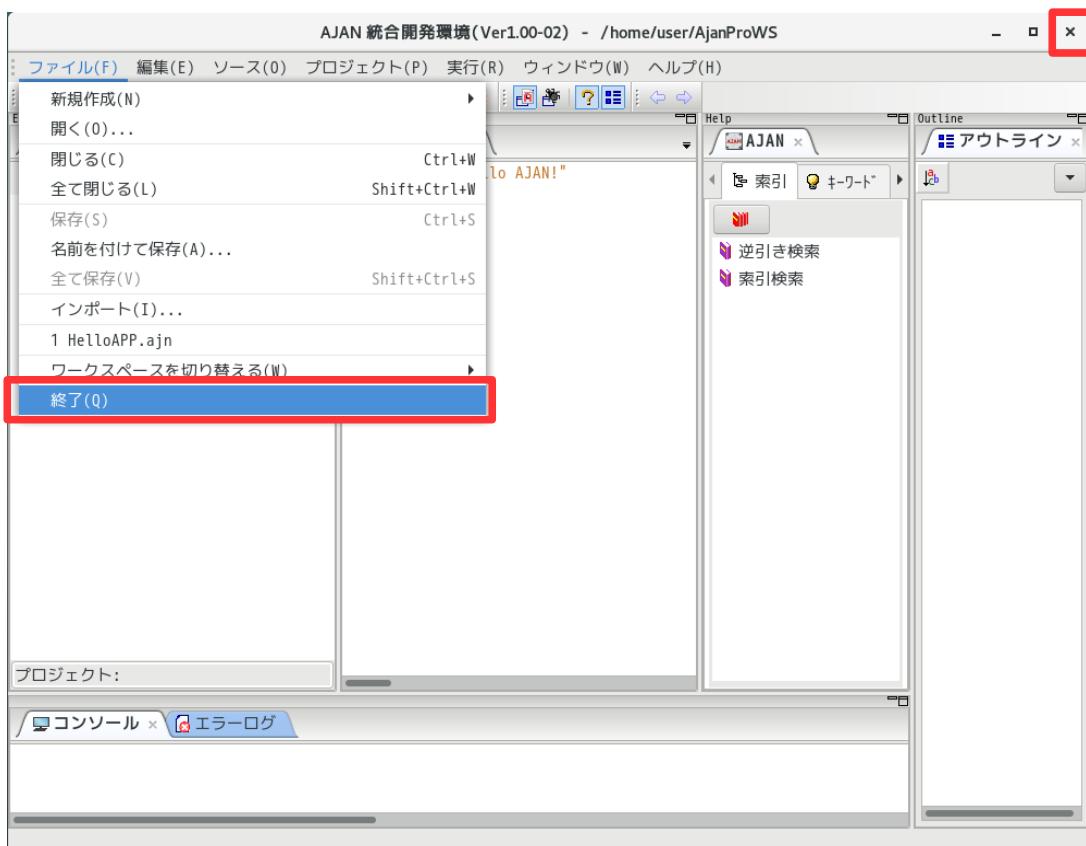
デスクトップから左上の「アプリケーション」 → 「Interface」 → 「AJAN」 → 「AJAN」をクリックします。



### 4.2.2 終了方法

AJANを終了するには、メニューの「ファイル(F)」 → 「終了(Q)」を選びます。

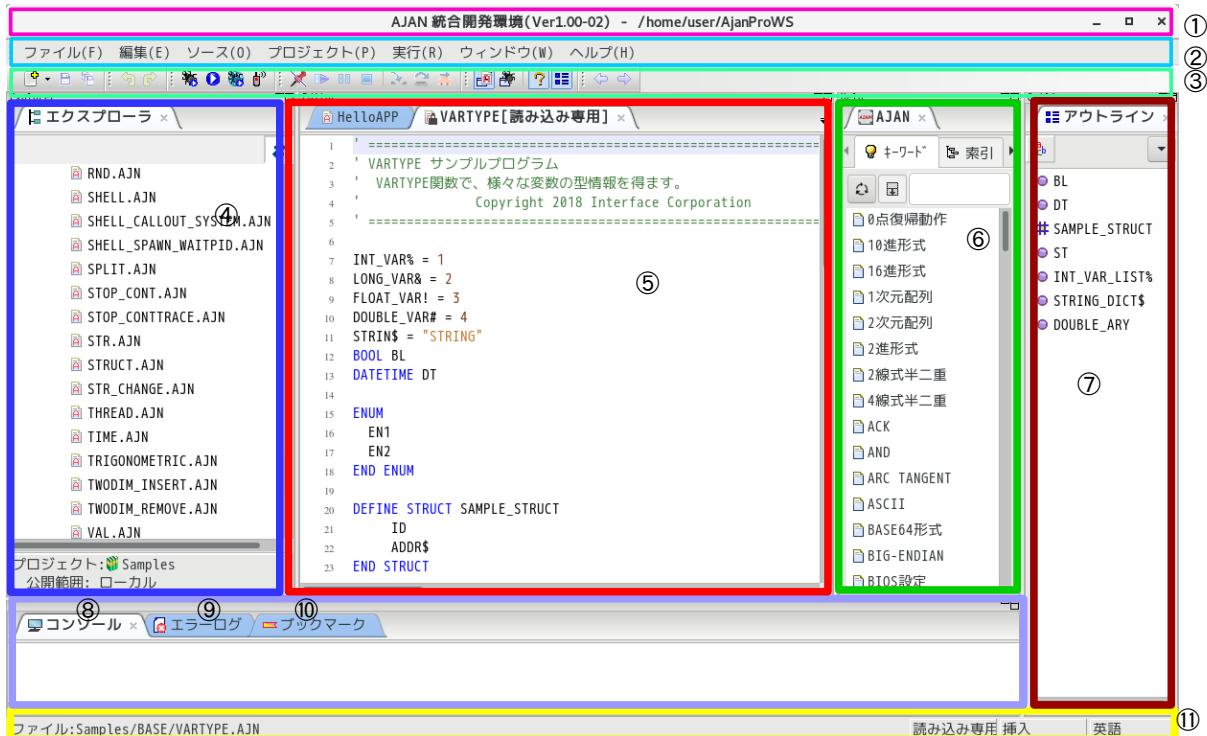
右上の「×」ボタンでも終了できます。



## 4.3 画面説明

### 4.3.1 エディタ画面

プログラムの編集に適したレイアウトです。



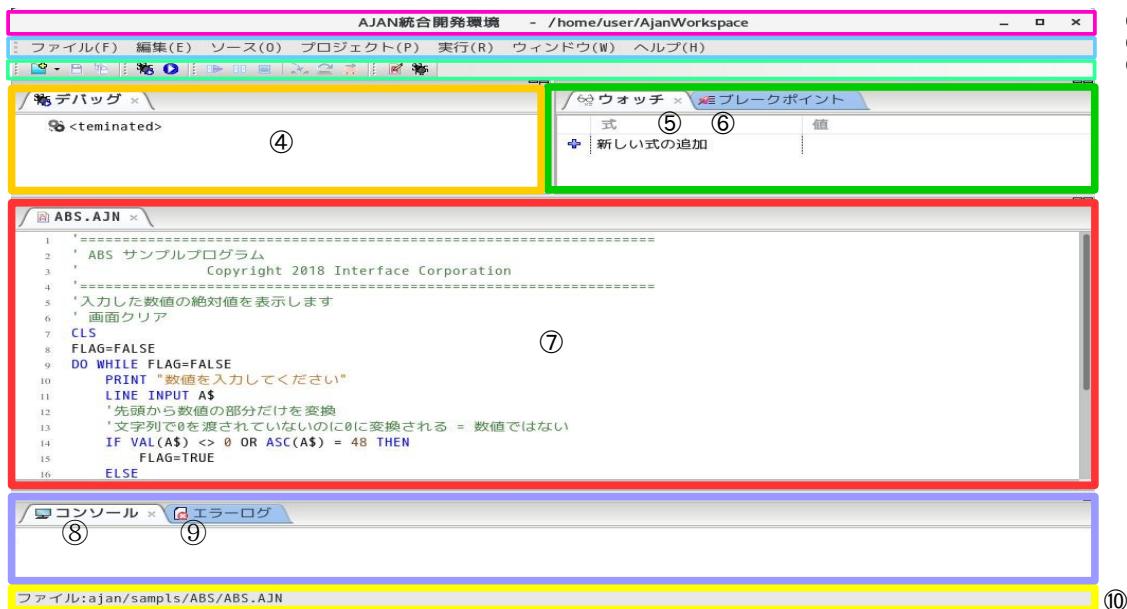
「エクスプローラ」「エディタ」「コンソール」「エラーログ」「ヘルプ」等のウィンドウで構成されています。

番号	名称	説明
①	タイトルバー	タイトルと共に、ワークスペース名が表示されます。 ※ワークスペースについては、『4.4管理フォルダの階層構造』を参照
②	メニューバー	メニュー名をクリックすると、プルダウンで選択可能な項目が表示されます。
③	機能ボタン	主要な機能がボタンとして用意されており、ワンクリックで機能を実行出来ます。
④	エクスプローラウィンドウ	プロジェクトを構成するフォルダ及びファイルがツリー形式で表示され、ファイルの移動等の基本的な操作が行えます。
⑤	エディタウィンドウ	AJANソースファイルの編集が行えます。 エクスプローラウィンドウからファイル名をダブルクリックするか、ファイルをドラッグ & ドロップすると開きます。
⑥	ヘルプウィンドウ	コマンドを検索する事が出来ます。
⑦	アウトラインウィンドウ	エディタウィンドウで現在選択しているファイルの変数、構造体、ファンクション、サブルーチン、スレッドの宣言が一覧で表示されます。クリックすると宣言箇所にジャンプします。
⑧	コンソールウィンドウ	実行プログラムが標準出力に対して出力した内容が表示されます。
⑨	エラーログウィンドウ	コンパイルエラー時のメッセージが表示されます。
⑩	ブックマークウィンドウ	エディタ上のファイルに設定したブックマークの一覧が表示されます。
⑪	ステータスバー	現在開いているファイルの情報やエディタの状態が表示されます。  ファイルパス: プロジェクト内ファイル: プロジェクト以降のパスが表示されます。 プロジェクト外ファイル: フルパスが表示されます。 ※プロジェクトについては、『4.4.1 プロジェクト管理』を参照 書き込み可否: 現在開いているファイルが書き込みできるかを表します。 挿入/上書き: エディタで編集する際にカーソル位置の文字への動作を表します。 英語/日本語: 現在のエディタの入力モードが英語か日本語かを表します。

### 4.3.2 デバッグ画面

プログラムのデバッグに適したレイアウトです。

「デバッグ」「ウォッチ」「ブレークポイント」「コンソール」「エラーログ」等のウィンドウで構成されています。



番号	名称	説明
①	タイトルバー	タイトルと共に、ワークスペース名が表示されます。 ※ワークスペースについては、『4.4管理フォルダの階層構造』を参照
②	メニューバー	メニュー名をクリックすると、プルダウンで選択可能な項目が表示されます。
③	機能ボタン	主要な機能がボタンとして用意されており、ワンクリックで機能を実行出来ます。
④	デバッグウィンドウ	稼動スレッドと実行ファイル、実行行が表示されます。
⑤	ウォッチウィンドウ	監視する変数の登録と、値の参照が行えます。
⑥	ブレークポイントウィンドウ	ブレークポイントの一覧が表示されます。
⑦	エディタウィンドウ	AJANソースファイルの編集が行えます。 エクスプローラーウィンドウからファイル名をダブルクリックするか、ファイルをドラッグ & ドロップすると開きます。
⑧	コンソールウィンドウ	実行プログラムが標準出力に対して出力した内容が表示されます。
⑨	エラーログウィンドウ	コンパイルエラー時のメッセージが表示されます。 その他、デバッグ時に発生したメッセージ等が表示されます。
⑩	ステータスバー	現在開いているファイルの情報やエディタの状態が表示されます。 ファイルパス: プロジェクト内ファイル: プロジェクト以降のパスが表示されます。 プロジェクト外ファイル: フルパスが表示されます。 ※プロジェクトについては、『4.4.1 プロジェクト管理』を参照 書き込み可否: 現在開いているファイルが書き込みできるかを表します。 挿入/上書き: エディタで編集する際にカーソル位置の文字への動作を表します。 英語/日本語: 現在のエディタの入力モードが英語か日本語かを表します。

### 4.3.3 画面レイアウトの切り替え

AJANの画面レイアウトは、プログラムを編集するのに適した「エディタ」と、プログラムをデバッグするのに適した「デバッグ」の2つのレイアウトが用意されています。

メニューバーの「ウインドウ(W)」→「レイアウト(L)」→「レイアウトの切り替え(C)」→

「エディタ(E)」:エディタレイアウトが開きます。

「デバッグ(D)」:デバッグレイアウトが開きます。

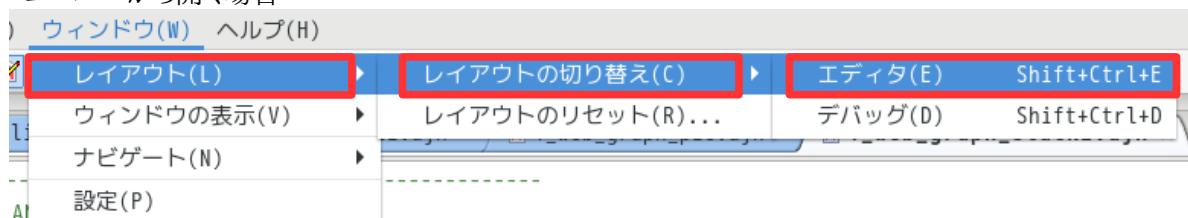
	<p>「エディタ」、「デバッグ」の2種類のレイアウトは、それぞれユーザーが変更したレイアウト状態を記憶します。</p>
---	---

#### ■エディタ画面への切り替え

機能ボタンを使用する場合

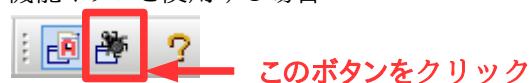


メニューバーから開く場合

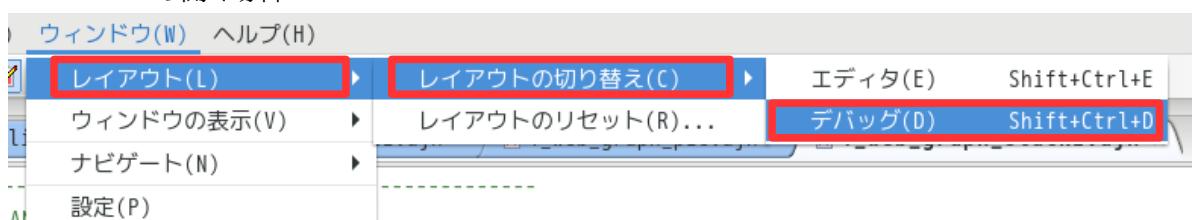


#### ■デバッグ画面への切り替え

機能ボタンを使用する場合



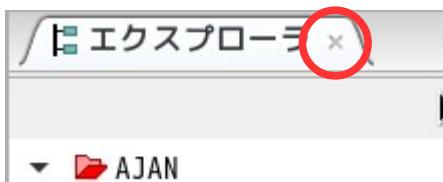
メニューバーから開く場合



#### 4.3.4 各種ウィンドウの開き方/閉じ方

##### ■閉じる

基本的なウィンドウの閉じ方は、各ウィンドウの「×」ボタンをクリックして下さい。



##### ■開く

メニューバーの「ウィンドウ(W)」→「ウィンドウの表示(V)」から、希望のウィンドウを選択して下さい。

##### ■トグル操作

一部のウィンドウは、機能ボタンをクリックする事で、開く←→閉じるの動作をトグルで実行できます。

トグル操作が可能なウィンドウは、以下となります。



アイコン	項目名	説明
?	ヘルプウィンドウ	ヘルプウィンドウを表示/非表示(トグルボタン)します。
≡	アウトライン	アウトラインウィンドウを表示/非表示(トグルボタン)します。
CS	CSデバイスマネージャ	CSデバイスマネージャを起動します。(EtherCAT搭載モデルのみ)

### 4.3.5 メニューバー

メニューバーでは、各機能をメニュー形式で呼び出せます。



メニュー名	項目名	説明
ファイル	新規作成	プロジェクト、フォルダ、ファイルの新規作成が行えます。
	開く	AJANファイル及び、全ての拡張子のテキストファイルが開けます。エディタウィンドウにファイルをドロップしても開けます。
	閉じる	エディタウィンドウで選択しているファイルを閉じます。
	全て閉じる	エディタウィンドウで開いている全てのファイルを閉じます。
	保存	エディタウィンドウで選択しているファイルを保存します。
	名前を付けて保存	エディタウィンドウで選択しているファイルを別名で保存します。
	全て保存	編集されたファイルを全て保存します。
	インポート	AJANファイルをプロジェクト内に取り込みます。
	最近開いたファイル	最近開いたファイルが、最大5件まで一覧で表示されます。クリックすると、エディタで開く事ができます。
	ワークスペースを切り替える	他のワークスペースに切り替えてIDEを再起動します。
編集	終了	IDEを終了します。
	元に戻す	編集箇所を元に戻します。
	やり直し	「元に戻す」をキャンセルします。
	切り取り	選択範囲を切り取ってクリップボードに格納します。
	コピー	選択範囲をクリップボードにコピーします。
	貼り付け	クリップボードのデータをカーソル位置に貼り付けます。
	削除	選択範囲を削除します。
	全て選択	テキストを全て選択状態にします。
	検索／置換	文字列の検索、置換のダイアログが開きます。
	指定行へジャンプ	指定した行へカーソルを移動させます。
ソース	コンテンツアシスト	AJANコマンドの選択メニューが開きます。
	コメントの切り替え	エディタウィンドウで選択している行のコメントアウト／コメント解除を切り替えます。
プロジェクト	インデント/逆インデント	カーソル位置もしくは選択行のインデントを追加したり削除できます。
	プロジェクトを開く	既存プロジェクトを開きます。
	プロジェクトを閉じる	エクスプローラウィンドウで選択しているプロジェクトを閉じます。
	ファイルをコンパイル	手動でコンパイルを実行します。 実行メニューの「実行」及び「デバッグ実行」を行うと、自動的にコンパイルも実施されます。

メニュー名	項目名	説明
実行	スーパーユーザーで実行	プログラムの通常実行とデバッグ実行に対してroot権限を付与します。
	再開	デバッグ実行の中断時に、実行状態にします。
	中断	デバッグ実行中に停止状態にします。
	終了	実行中やデバッグ実行中にプログラムを終了します。
	ステップ・イン	デバッグ実行の中断時にステップ動作をします。 (サブルーチン内もステップ動作を行います)
	ステップ・オーバー	デバッグ実行の中断時にステップ動作をします。 (サブルーチンはスキップします)
	自動トレース	ステップ・インを自動で繰り返します。
	実行	プログラムを通常実行します。
	デバッグ実行	プログラムをデバッグ実行します。
	Webのデバッグ/Webデバッガの停止	Webデバッグモードを有効/無効にします。通常は操作不要です。 詳細は、『4.11.7 WebブラウザにURLを直接指定した場合にデバッグする方法』を参照して下さい。
	コマンドライン引数指定	プログラムの実行/デバッグ実行時の引数を指定します。
ウィンドウ	レイアウト→ レイアウトの切り替え	画面レイアウト(ウィンドウ配置)を予め設定されている状態に切り替えます。 ・エディタ ・デバッグ
	レイアウト→ レイアウトのリセット	現在選択されている画面レイアウトを、出荷時状態にリセットします。
	ウィンドウの表示	指定したウィンドウを開きます。 ・コンソール ・エラーログ ・デバッグ ・ウォッチ/ブレークポイント ・ヘルプ ・エクスプローラ ・アウトライン
	ナビゲート	エディタの選択ファイルを逐次切り替えます。 ・次のエディタ ・前のエディタ
	設定	ウィンドウの色とフォントの設定が行えます。
ヘルプ	ヘルプを開く	閉じたヘルプウィンドウを表示します。 以下の操作と同様です。 ウィンドウ→ウィンドウの表示→ヘルプ
	コマンドヘルプを開く	カーソル位置のAJANコマンドのリファレンスを表示します。
	AJANについて	AJANのバージョンを表示します。

### 4.3.6 機能ボタン

機能ボタンは、ボタンで機能を呼び出せます。

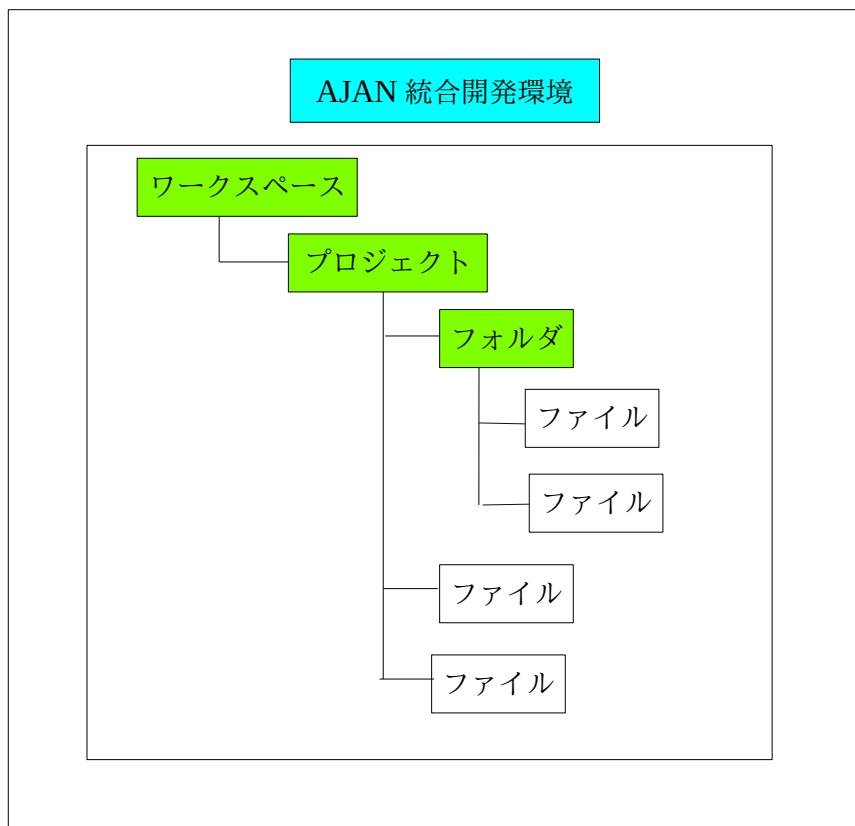


アイコン	項目名	説明
	新規作成	プロジェクト、フォルダ、ファイルの新規作成が行えます。
	保存	エディタウィンドウで選択しているファイルを保存します。
	全て保存	エディタウィンドウで開いている全てのファイルを閉じます。
	元に戻す	編集箇所を元に戻す
	やり直し	元に戻した編集箇所をやり直す
	デバッグ実行	プログラムをデバッグ実行します。
	実行	プログラムを通常実行します。
	Webデバッグモード	Webデバッグモードの有効/無効を切り替えます。通常は操作不要です。 詳細は『4.11.7 WebブラウザにURLを直接指定した場合にデバッグする方法』を参照して下さい。
	全てのブレークポイントをスキップ	トグルボタンになっており、オンの状態では全てのブレークポイントをスキップします。ブレークポイントは個別に有効/無効を切り替えられますが、その設定に影響を与えずに一斉にスキップさせる事が出来ます。再度、オフに切り替える事で、ブレークポイントで停止させる事ができます。
	再開	デバッグ実行の中断時に、実行状態にします。
	中断	デバッグ実行中に停止状態にします。
	終了	実行中やデバッグ実行中にプログラムを終了します。
	ステップ・イン	デバッグ実行の中断時にステップ動作をします。 (サブルーチン内もステップ動作を行います)
	ステップ・オーバー	デバッグ実行の中断時にステップ動作をします。 (サブルーチンはスキップします)
	自動トレース	ステップ・インを自動で行います。
	エディタ画面	画面をエディタレイアウトに切り替えます。
	デバッグ画面	画面をデバッグレイアウトを切り替えます。
	ヘルプウィンドウ	ヘルプウィンドウを表示/非表示(トグルボタン)します。
	アウトライン	アウトラインウィンドウを表示/非表示(トグルボタン)します。
	CSデバイスマネージャ	CSデバイスマネージャを起動します。(EtherCAT搭載モデルのみ)
	前回のカーソル位置に戻る	アウトライン機能でカーソル移動した際に、カーソルを前回の位置に戻します (最大5箇所)。
	次のカーソル位置に進む	カーソル位置を「前回のカーソル位置に戻る」を実行する前の位置に戻します (最大5箇所)

## 4.4 管理フォルダの階層構造

AJAN 統合開発環境が管理するフォルダやファイルは以下の階層構造となっています。

「ワークスペース」は、AJAN 統合開発環境管理下における作業領域で、下位層にはプロジェクトやフォルダ、更にプログラムファイルが管理されます。



#### 4.4.1 プロジェクト管理

プロジェクトとは、特別なフォルダで、ソースファイルを実行する際の動作に影響を与えます。

##### a) プロジェクトの種類

アイコン	基本プロジェクト名	説明
	AJAN (読み書き可)	プログラム作成用プロジェクトです。 自由にファイルやフォルダを作成する事が出来ます。 この種類のプロジェクトは複数作成したり、削除が可能です。
	Samples (読み込み専用)	サンプルプログラム用プロジェクトです。 <ul style="list-style-type: none"> <li>サンプルプログラムが多数格納されており、このプロジェクトからソースコードを直接開いて実行する事が出来ます。</li> <li>サンプルプログラムは、製品をバージョンアップした際に自動的に最新の内容に更新されます。</li> <li>このプロジェクトは読み込み専用となっており、プロジェクトの削除を含めて、全ての改変が行えません。ソースコードを修正して実行したい場合、プログラム作成用プロジェクト内にコピーしてから修正して下さい。</li> <li>このプロジェクトは削除できません。不要な場合、『4.4.1 c) プロジェクトを閉じる』を実行して下さい。</li> </ul>

## b) プロジェクト新規作成

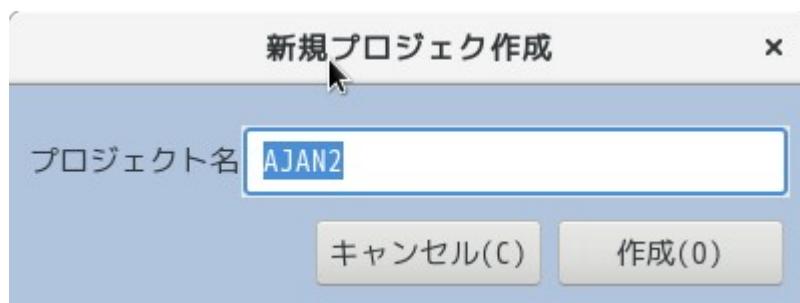
新たなプログラムを作る際に、プロジェクトを新規作成する方法を説明します。

プロジェクトを作成するには、メニューバーから「ファイル(F)」 → 「新規作成(N)」 → 「プロジェクト(P)...」をクリックします。

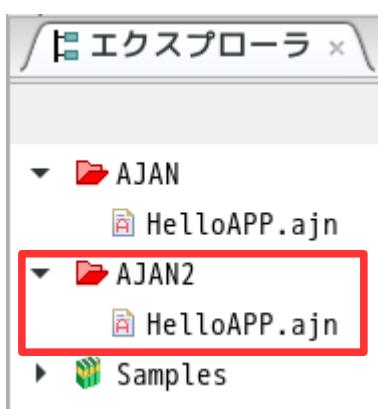
下記ダイアログボックスが開いて、プロジェクト名の入力を求められます。

デフォルトで、プロジェクト名の末尾に連番が付加されて重複しない名称が設定されています。

名前は後からでも変更出来ますので、デフォルトで良い場合は、そのまま作成ボタンをクリックしてください。



エクスプローラウィンドウに、新たに作成したプロジェクトが登録されます。



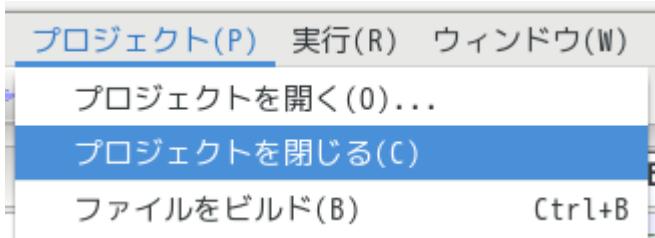
- ・プロジェクトを複数作成できます。
- ・プロジェクト名を変更するには、プロジェクトを右クリック→「名前の変更」で任意の名前に変更できます。
- ・プロジェクトを作成すると、テンプレートのHelloAPP.ajnも合わせて作成されます。不要であれば、削除して下さい。

### c) プロジェクトを閉じる

- 1) 閉じたいプロジェクトをエクスプローラで選択します。



- 2) メニューバーから「プロジェクト(P)」→「プロジェクトを閉じる(C)」をクリックします。



- 3) メッセージボックス「プロジェクト[ XXX ]を閉じても良いですか?」と表示されますので、問題なれば「はい」をクリックしてください。

### d) プロジェクトを開く

- 1) メニューバーから「プロジェクト(P)」→「プロジェクトを開く(O)」をクリックします。



- 2) プロジェクトファイル一覧が表示されるので、開きたいプロジェクトを選択して、「開く(O)」をクリックしてください。

<span data-bbox="203 1949 282 2030">!</span>	プロジェクトファイル一覧は、「プロジェクトを閉じる(C)」で閉じたプロジェクトが無いと表示されません。
--	---

## e) プロジェクトを外部公開する

各プロジェクトはAJAN内蔵Webサーバによって公開されており、PageGeneratorで作成したWebページをWebブラウザで表示させる事が出来ます。

デフォルトでは、公開範囲はローカル(このコンピュータの中だけ)になっており、このコンピュータが属するネットワークの他のコンピュータからWebアプリを実行する事ができません。

Webアプリを、他のコンピュータから使用できるように許可するには、以下の手順で設定を変更してください。

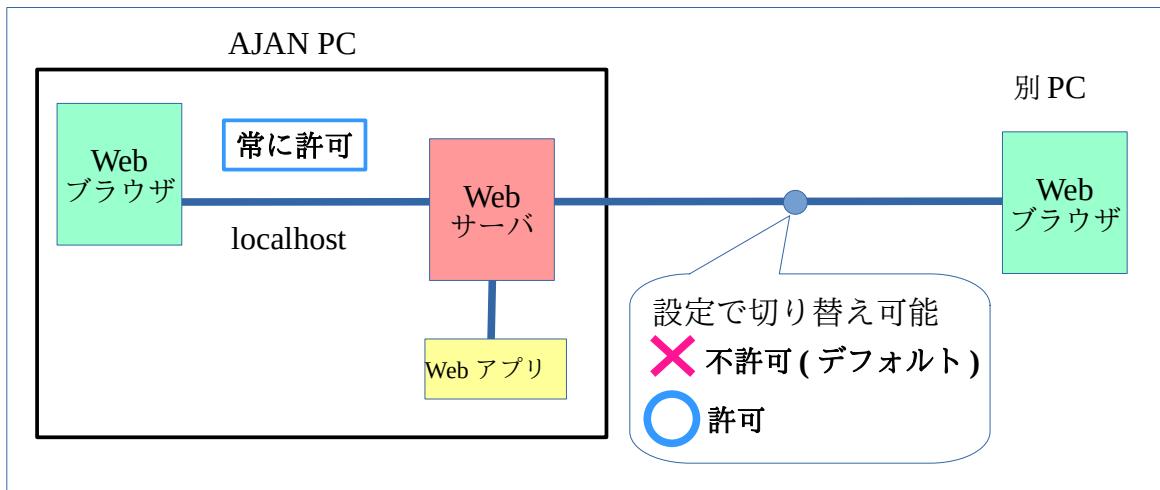


図4-2 Webアプリ公開範囲

プロジェクトのアイコン状態で、公開範囲が確認できます。

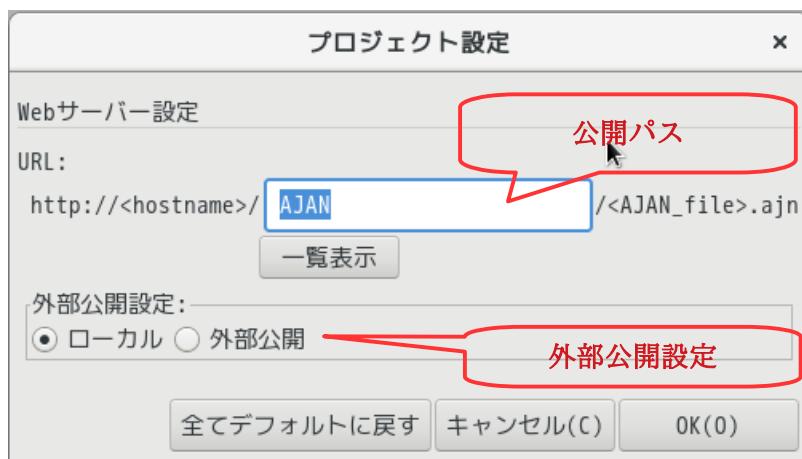
アイコン	状態
	公開範囲がローカルになっており、このPC内でのみアクセスできます（デフォルト）。
	公開範囲が外部公開になっており、別PCからのアクセスが可能。

### ■公開範囲設定

1. エクスプローラウィンドウのプロジェクト上で右クリックするとポップアップメニューが開きますので、「設定」をクリックしてください。



2. プロジェクト設定ダイアログが開きますので、設定を変更してください。



公開パス	URLの一部を任意に変更する事が出来ます。 判りやすいURLに変更したい場合に設定してください。 ※漢字等のマルチバイト文字は使用できません。  例)URLの波線の箇所を変更できます http://localhost/AJAN/1_web_graph_bar.ajn ↓ http://localhost/sales/1_web_graph_bar.ajn
外部公開設定	Web アプリケーションの使用できる範囲を設定します。 <ul style="list-style-type: none"> <li>• <b>ローカル</b>（デフォルト）の場合： Web アプリケーションをローカルPC内でのみ使用できます。 アプリケーションの開発時などに設定してください。 URLの例：http://localhost:8080/ ~/1_web_graph_bar.ajn ※ホスト名は localhost:8080 になります。</li> <li>• <b>外部公開</b>の場合： Web アプリケーションをネットワークで接続された別のコンピュータから実行出来るようになります。 作成したアプリケーションを外部のPCから使用する場合に設定してください。 URLの例：http://192.168.100.100/ ~/1_web_graph_bar.ajn ※ホスト名は IPアドレス になります。</li> </ul>
一覧表示	公開パスは、全てのワークスペースのプロジェクトに一意の名前が割り当てられています。それらを一覧で確認したい場合に、このボタンをクリックして下さい。一覧表示の「Cleanup」ボタンは、何らかの要因で存在しないプロジェクトに公開パスが割り当てられたままになっている場合に公開パスを削除します。

3. OKボタンをクリックすると適用されます。

URLは大文字/小文字を判別しますので、設定された公開パス、AJANファイル名に合わせて入力してください。

<b>!</b>	公開パスは、同一コンピュータの複数のプロジェクトで重複しないように設定してください。
----------	--

	Webサーバを停止させる必要がある場合、7.5AJAN Webサーバの停止方法をご参照ください。
---	--

#### 4.4.2 ワークスペースの切り替え

初期のワークスペースAjanProWSを切り替えるには、以下の操作を行います。

1. 「ファイル(F)」 → 「ワークスペースを切り替える(W)...」  
※ワークスペースを1度でも作成している場合、「ファイル(F)」→「ワークスペースを切り替える(W)」→「その他(O)...」
2. 以下のダイアログが起動しますので、新しいワークスペースを入力してOKをクリックしてください。



※ ログインした時のアカウントにより、user部分が異なります。

※ 起動時に毎回このダイアログを表示させる場合、チェックボックスのチェックを外してください。

「ワークスペース:」のデフォルトのパスは、「/home/user/AjanProWS」となっています。

ワークスペース名やパス名は変更できます。

3. 「参照」ボタンを押すとホームディレクトリ(~/)内のフォルダ(ディレクトリ)が表示されます。ワークスペース以外を選択した場合、パスの最後に「AjanProWS」が自動的に付加されます。
4. OKボタンをクリックすると、IDEが一旦終了して、新しいワークスペースで起動します。

	現在設定されているワークスペースは、タイトルバーで確認できます。
	AJAN Pro統合開発環境(Ver1.00-02) - /home/user/AjanProWS

## 4.5 プログラムの編集/検索/置換

「エディタウインドウ」で、プログラムの編集/検索/置換ができます。



ステータスバーに現在編集中のファイルのパスが表示されます。



プロジェクト内のファイルは、プロジェクトからのパスが表示されます。

プロジェクトが属するフォルダ(ワークスペースのパスは、タイトルバーに表示されますので、合わせることで、フルパスになります。

### 4.5.1 プログラムの編集

エディタウインドウは、Linuxで言うgeditテキストエディタの編集と同じ機能を持っています。

主なキー操作を、以下に示します。

キー	動作
Enter	カーソル位置で、改行します。
Back Space	カーソル位置の左側の1文字を削除します。
Delete	カーソル位置の右側の1文字を削除します。
Home	行の先頭に移動します。 「Ctrl」キーと併用すると、プログラム全体の先頭に移動します。
End	行の終端に移動します。 「Ctrl」キーと併用すると、プログラム全体の終端に移動します。
Page Up	1ページ分 スクロールアップします。
Page Down	1ページ分 スクロールダウンします。
Insert	編集動作を挿入モード/上書きモードに切り替えます。

メニューバーの「編集」から、いくつかの編集操作が呼び出せます。

項目 (ショートカットキー)	説明
元に戻す (Ctrl+Z)	直前の動作を元に戻します。
やり直し (Shift+Ctrl+Z)	上記で元に戻した処理をやり直します。
切り取り (Ctrl+X)	選択した文字を切り取ります。
コピー (Ctrl+C)	選択した文字をコピーします。
貼り付け (Ctrl+V)	カーソル位置に文字を貼り付けます。
削除	選択した文字を削除します。
全て選択 (Ctrl+A)	全ての行を選択します。
検索/置換 (Ctrl+F)	文字列の検索、置換のダイアログが開きます。
指定行ヘジャンプ (Ctrl+J)	指定した行へカーソルを移動させます。
コンテンツアシスト(Alt+/)	AJANコマンドの選択メニューが開きます。

## 4.5.2 自動補完機能

エディタ上で文字を入力すると、入力内容に応じてAJANコマンドの候補が表示されます。

入力内容に応じて絞り込まれ、候補が無くなると自動的に閉じます。

1. エディタウインドウ上で、入力したいコマンドの入力を開始して下さい。先頭一致するコマンドがリストで表示されます。
2. キーボードの上下キーかマウスで、入力したいコマンドをリストから選択、EnterキーもしくはTABキーを押すか、マウスでダブルクリックしてください。



選択コマンドがエディタ上に入力され、メニューが自動的に閉じます。

本メニューをキャンセルしたい場合、ESCキーを押してください。

<span style="font-size: 2em;">!</span>	<ul style="list-style-type: none"> <li>• リストの先頭行は、入力途中のワードがそのまま表示されており、コマンド候補は2行目から表示されています。</li> <li>• 先頭行でEnterキーを押すと、リストの内容が確定入力されるのと同時に、エディタ上で改行が発生します。2行目以降でEnterキーを押すと、リストの内容が確定入力され、改行は発生しません。</li> <li>• 候補は小文字で入力すると小文字で表示され、大文字で入力すると大文字で表示されます。</li> </ul>
--	---

## 4.5.3 コンテンツアシスト機能

本機能は、自動補完機能と同等の機能になります。

自動補完機能はアルファベットを入力する事で起動しますが、本機能はメニューもしくはショートカットキー(Alt+ '/')で起動します。

カーソル位置にコマンドが途中まで入力されている場合、その文字列と一致するコマンドが絞り込まれて表示されます。

1. エディタウインドウで、コマンドを入力したい箇所にカーソルを移動します。
2. メニューバーの「編集(E)」→「コンテンツアシスト(T)」を選択するとコマンドメニューが表示されます。



3. キーボードの上下キーかマウスで、入力したいコマンドをリストから選択、EnterキーもしくはTABキーを押すか、マウスでダブルクリックしてください。
- 選択コマンドがエディタ上に入力され、メニューが自動的に閉じます。
- 本メニューをキャンセルしたい場合、ESCキーを押してください。

#### 4.5.4 ポップアップメニュー

エディタ上で右クリックすると、以下のポップアップメニューが起動します。



「コマンドヘルプを開く」「ウォッチに追加」「行を保護する」以外は、『4.5.1 プログラムの編集』メニューの機能と同等ですので、そちらをご参照下さい。

「コマンドヘルプを開く」は、AJANコマンド上で右クリックした場合のみ有効になります。

選択すると、そのコマンドのリファレンスマニュアルが開きます。

詳細は、『4.9.6 コマンドヘルプジャンプ機能』をご参照下さい。

「ウォッチに追加」は、AJANコマンド以外のワード上で右クリックした場合のみ有効になります。

選択すると、デバッグ機能のウォッチに追加されます。詳細は、「4.11.4 変数ウォッチ」をご参照下さい。

「行を保護する」は、ソースコードの行単位に書き込み保護や非表示にする等の保護が行えます。詳細は、「4.5.10 ソースコード保護」をご参照下さい。

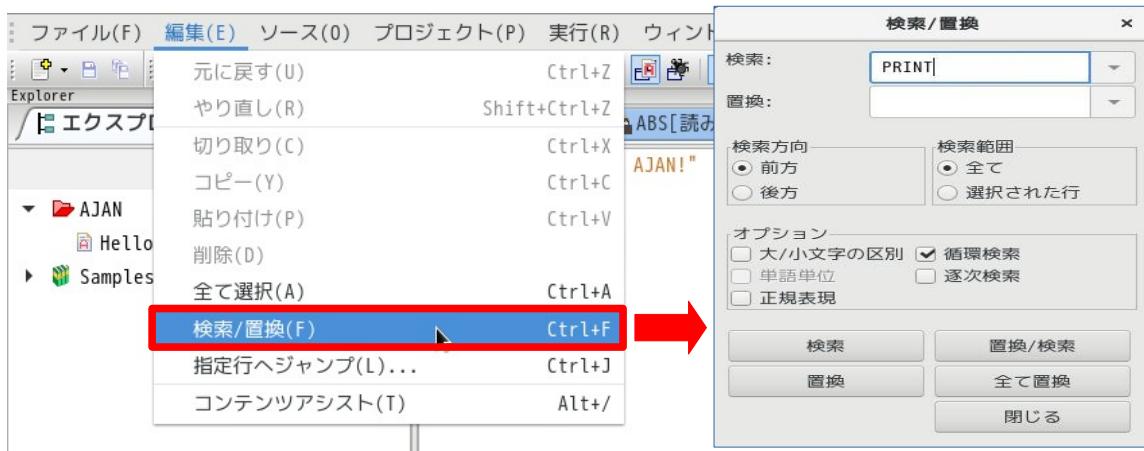
#### 4.5.5 指定行へジャンプ

メニューバーの「編集(E)」 → 「指定行へジャンプ(L)」を選択して、行番号を入力すると指定した行番号へジャンプします。



#### 4.5.6 プログラムの検索と置換

メニューバーの「編集(E)」 → 「検索/置換(F)」を選択すると、「検索/置換」ダイアログが表示されます。



項目名	説明	
検索:	検索したい文字列を入力します。	
置換:	置換する文字列を入力します。	
検索方向	前方	カーソルの前方に向かって検索します。
	後方	カーソルの後方に向かって検索します。
検索範囲	全て	プログラム全てを検索します。
	選択された行	指定した選択範囲を検索します。
オプション	大/小文字の区別	大文字/小文字の区別を行います。
	単語単位	単語単位で検索します。
	正規表現	正規表現で検索します。 (特殊な記号と組合せにより文字列の集合をパターンマッチングさせる機能です。正規表現はPythonライブラリを使用しています。正規表現の構文に付きましては、インターネット検索サイトで「Python 正規表現」で検索して頂ければ、ドキュメントが見つかります。)
	循環検索	循環検索します。 (最後の行まで検索すると、先頭行に戻って検索を続けます)
	逐次検索	逐次検索します。 (有効にしたカーソル位置から、検索文字列を1文字づつ入力する度に検索が実行されます)
ボタン	検索	検索します。
	置換	置換します。
	置換/検索	置換後に検索します。
	全て置換	全て置換します。
	閉じる	ダイアログを閉じます。

## 4.5.7 ブックマーク機能

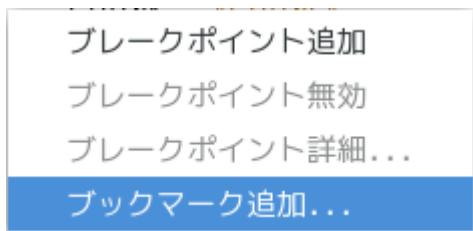
本機能を使用すると、エディタ上の任意のファイルの指定した行に、ブックマーク(しおり)を設定できます。設定したブックマークは、ブックマークウィンドウに一覧で表示され、それをクリックすることで、素早く指定ファイルのブックマーク位置にアクセスすることができます。

ファイルの行数が多い場合や、複数のファイルを編集する場合に便利です。

### ■手順

#### <ブックマーク登録>

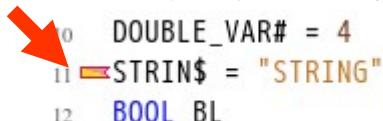
1. エディタウインドウで、ブックマークを設定したい行の行番号上で右クリックします。
2. 以下のポップアップメニューが開きます。



3. 「ブックマーク追加...」をクリックします。
4. 以下のダイアログが開きますので、ブックマーク名を入力して下さい。



5. OKをクリックすると、エディタの行番号にブックマークのアイコンが表示されます。



#### <ブックマークを開く>

1. ブックマークウィンドウのタブをクリックします。



2. 以下の様にブックマークの一覧が表示されますので、表示されている情報を参考に、目的のブックマークを見つけて、その行をダブルクリックすると、エディタ上にブックマークの箇所が表示されます。

説明	ファイル名	パス	場所
STRIN\$ = "STRING"	VARTYPE.AJN	/home/user/AjanProWS/Sam	11

## &lt;ブックマーク削除&gt;

1. エディタウインドウで、ブックマークを削除したい行の行番号上で右クリックします。

2. ポップアップメニューから、「ブックマーク削除」をクリックして下さい。

もしくは、ブックマークウインドウの削除したい行上で右クリックして、「削除」をクリックして下さい。

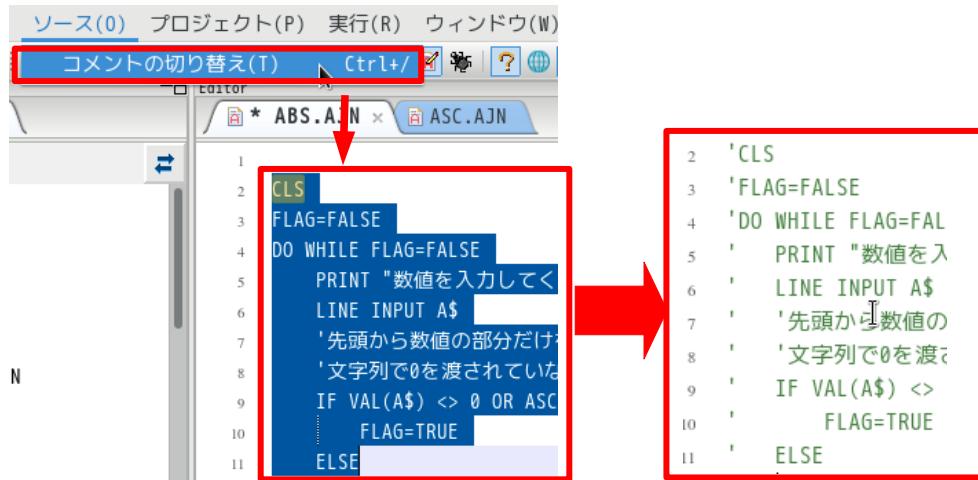
#### 4.5.8 コメントの切り替え

本機能を使用すると、指定した行をコメント行に切り替えができます。

##### ■手順

3. エディタウインドウで、変更したい行を選択します。

4. メニューバーの「ソース(O)」 → 「コメントの切り替え(T)」を選択すると指定された行がコメント行に切り替わります。再度同様に操作すると、コメント行を解除します。



#### 4.5.9 インデントガイド線

エディタウインドウのインデント(TABの位置)にガイド線が表示され、FORコマンド等のインデントを付けて表記する場合に、NEXTコマンドとの対応が判りやすくなります。

```

1 FOR I = 0 TO 100
2   FOR J = 0 TO 100
3     FOR K = 0 TO 100
4     ...
5     NEXT
6   NEXT
7 NEXT

```

	インデントガイド線は、「ウィンドウ(W)」→「設定(P)」→「エディタ」→「インデントガイド線を有効にする」のチェックを外すと、非表示にできます。
--	---

#### 4.5.10 ソースコード保護

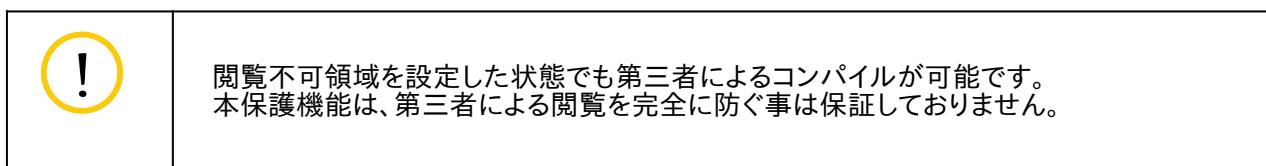
本機能は、AJANソースコードに対して、不用意な書き換えや、見せたくない部分を隠す等のソースコードレベルの保護機能を提供します。

```

2 ↓DEFINE STRUCT ABC
3   PARA1
4   PARA2
5 ↑END STRUCT
6
7 ←*この領域は非表示に設定されています*
8 *この領域は非表示に設定されています*
9 *この領域は非表示に設定されています*
10 *この領域は非表示に設定されています*
11 ←*この領域は非表示に設定されています*

```

種類	説明
書き換え不可領域	<ul style="list-style-type: none"> <li>この領域の書き込み操作は無効となります。</li> <li>コピー操作可能</li> </ul>
閲覧不可領域	<ul style="list-style-type: none"> <li>この領域の書き込み操作は無効となります。</li> <li>コピー操作は可能ですが、保護領域メッセージが、そのままコピーされますので、ソースコード自体はコピーされません。</li> </ul>



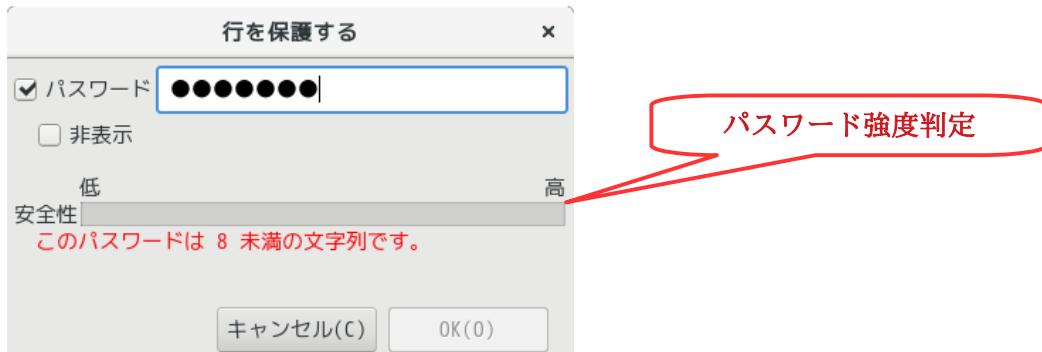
##### ■手順

- エディタウィンドウで、保護したい行を選択します。
- エディタ上で右クリックしてポップアップメニューを開きます。



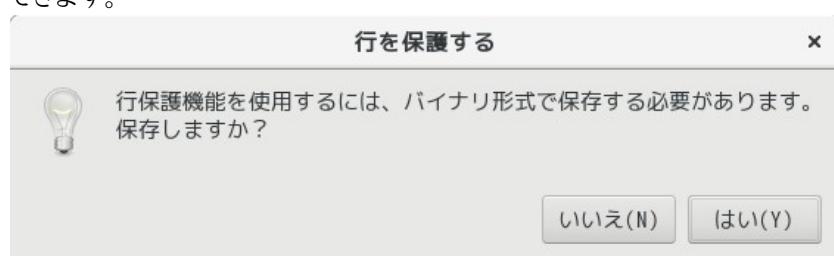
- 「行を保護する...」をクリックします。

4. 以下のダイアログが開くので、適宜、必要な項目を設定して下さい。



項目	説明
パスワード	<p>保護領域を解除するのに必要なパスワードとなります。パスワードを設定しない場合は、誰でも解除可能な保護領域を設定する事が出来ます。</p> <p>パスワードを設定する場合、チェックボックスにチェックを付けて、テキストボックスにパスワードを入力して下さい。</p> <p>パスワード強度の最低条件を満たしていない場合、赤字で理由が表示され、OKボタンは無効になります。最低条件を満たすと、OKボタンが有効になり、安全性のバーが強度に応じて増減します。</p> <p>極力「高」に近づけるようにパスワードを設定する事を推奨します。</p> <p>&lt;強度を上げるポイント&gt;</p> <ul style="list-style-type: none"> <li>・なるべく文字数を多くする。</li> <li>・アルファベットに記号や数字を組み合わせる。</li> </ul> <p>パスワード強度の設定は、/etc/security/pwquality.confを編集する事で変更出来ます。</p> <ul style="list-style-type: none"> <li>・編集には、root権限が必要です。</li> <li>・設定項目の詳細は、端末を起動して下記コマンドを実行して下さい。 man pwquality.conf</li> </ul>
非表示	チェックを付けると、閲覧不可の保護領域を設定出来ます。チェックを付けない場合、書き換え不可領域となります。

5. OKをクリックします。パスワードを掛けた場合は、パスワードの確認を求められますので、先程入力したパスワードを再度入力して下さい。
6. 初めて保護領域を設定する場合、ファイルをテキスト形式からバイナリ形式に変換する事の承認を求めています。



即座にファイルを保存して良い場合は、「はい」をクリックして下さい。

「いいえ」をクリックした場合、次回、保存操作を実行したタイミングでバイナリ形式に変換されます。

バイナリ形式に変換されたファイルは、ファイル名の先頭に以下のアイコンが表示されます。

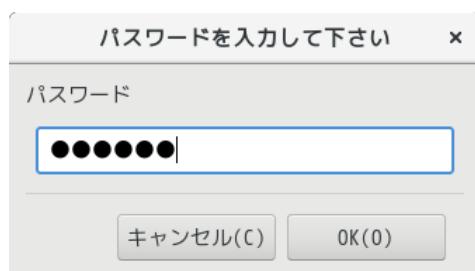


## ■解除手順

- エディタウインドウで、保護を解除したい領域上で右クリックしてポップアップメニューを開きます。



- 「保護を解除する」をクリックして下さい。
- パスワードを設定している場合、パスワード入力のダイアログボックスが表示されますので、保護領域を掛けた際のパスワードを入力して下さい。  
パスワードを掛けていない場合は、即座に保護領域が解除されます。



	解除する領域を選択する際に、範囲選択は行わないで下さい。範囲選択を行うと、「保護を解除する」は表示されません。
	保護領域が全て解除された場合、次回のファイル保存時にバイナリ形式からテキスト形式に自動的に変換されます。

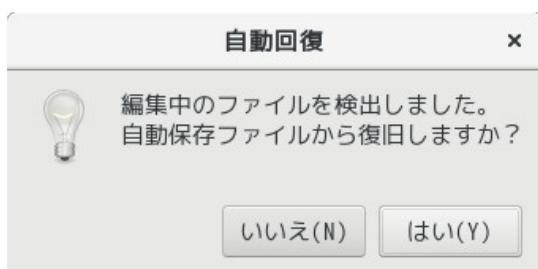
#### 4.5.11 ファイル自動回復機能

エディタウィンドウでファイルの編集中に、IDEが動作異常等で強制終了した場合に、自動回復する機能を搭載しています。  
編集操作を行う度に、デフォルト設定で3分後にファイルの自動回復情報を保存します。  
これにより、IDEが強制終了しても、前回、自動回復情報を保存した時点に復元する事が出来ます。

自動回復機能が適用される条件は、以下を全て満たした時です。

- ・エディタでファイル(拡張子に制限はありません)に変更を行う。
- ・プロジェクト内のファイル(エクスプローラに表示されているファイル)であること。

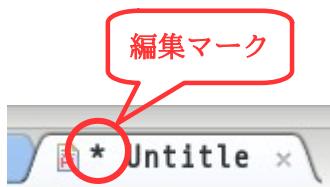
1) 強制終了した後にIDEを起動させると、以下のメッセージボックスが表示されます。



最後の状態に復旧させるには、「はい」を選択して下さい。

「いいえ」を選択すると、自動回復情報を破棄してIDEを起動します。

2) 自動回復を行ったファイルは以下の様に編集マークが付加されます。



回復した内容を確定させるには、 (保存)ボタンをクリックする等でファイルを保存して下さい。  
保存せずに破棄すると、自動復旧した情報は失われ、前回、保存操作を行った時点の内容が維持されます。



自動回復情報を保存する間隔は、設定メニューで変更出来ます。  
設定メニューに付きましては、『4.12 各種設定』の「エディタ」→「自動回復情報の保存」を参照して下さい。

## 4.6 ファイル操作

新規フォルダまたは、新規ファイルを作成します。

### 4.6.1 プログラムの新規作成

#### ■手順

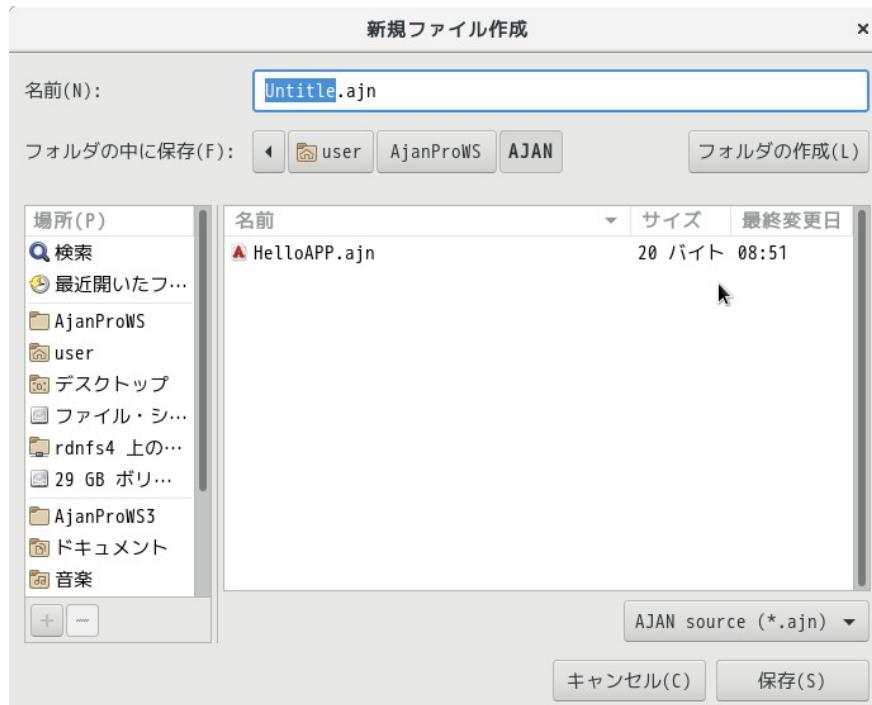
1. メニューバーから以下の手順を実行します。  
「ファイル(F)」 → 「新規作成(N)」 → 「ファイル(F)」
2. 作成出来る2種類のファイルが表示されます。



種別	説明
AJANファイル	拡張子.ajnのファイルを作成します。AJANアプリを作成する場合は、こちらを選択して下さい。
PageGenerator	PageGeneratorを使用してWebページを作成する場合に選択して下さい。詳細は、「第5章 PageGenerator」を参照して下さい。

3. ここでは「AJANファイル」を選択した場合を説明します。

新規ファイル作成用ダイアログが起動します。



4. デフォルトファイル名Untitled.ajnとなっていますので任意に変更して保存ボタンをクリックすると、エクスプローラーウィンドウにファイルが追加され、エディタウィンドウで開かれます。

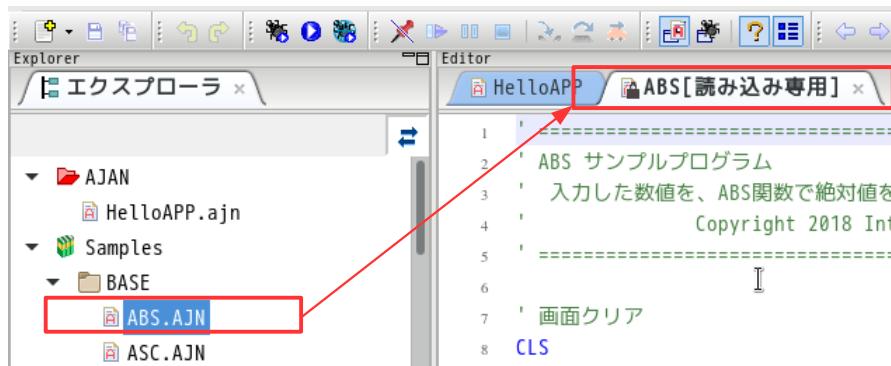
## 4.6.2 プログラムを開く

メニューバーもしくは、エクスプローラーウィンドウから既存のプログラムファイルを開きます。

### <プロジェクト内のプログラムファイル>

エクスプローラーウィンドウから以下の手順を実行します。

1. ウィンドウに表示されている既存フォルダの左側の「▶」をクリックすると「▼」となり、ファイルが存在する場合はファイルが表示されます。
2. ファイルを選択して、ダブルクリックしてください。
3. エディターウィンドウにファイルが開かれ、編集や実行ができるようになります。

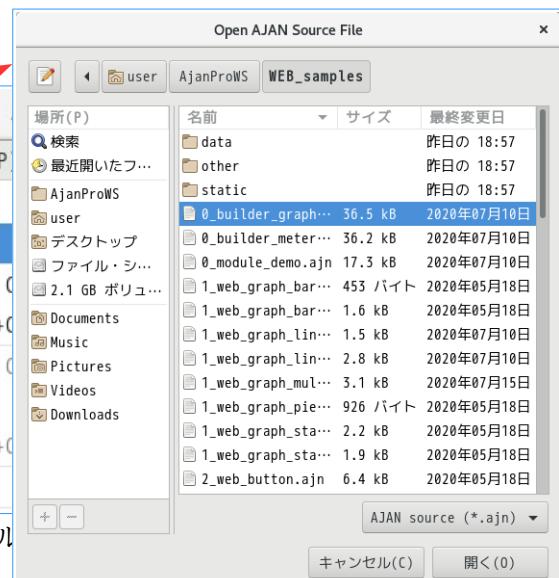
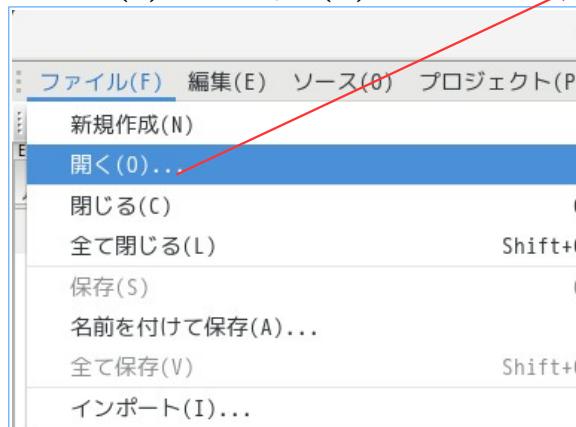


上記手順を繰り返すことで、複数のファイルを開く事ができます。

### <プロジェクト外のプログラムファイル>

メニューバーから以下の手順を実行します。

1. 「ファイル(F)」 → 「開く(O)」

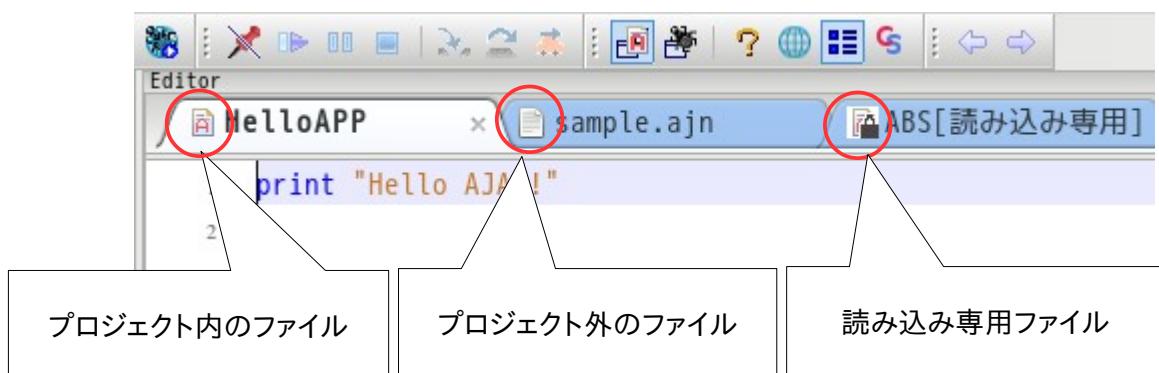


2. ファイル一覧が表示されます。フォルダ上でダブルクリックが表示されます。
3. ファイルを選択して、タブルクリックもしくは、「開く(O)」をクリックしてください。
4. エディターウィンドウにファイルが開かれ、編集や実行ができるようになります。

	<ul style="list-style-type: none"> <li>・ファイルをエディターウィンドウに直接ドラッグ&amp;ドロップすることでも開くことが出来ます。</li> <li>・プロジェクト外ファイルは実行はできますがデバッグはできません。</li> </ul>
--	--

## &lt;アイコン種類&gt;

開いたファイルの所属や状態でアイコンが変化します。



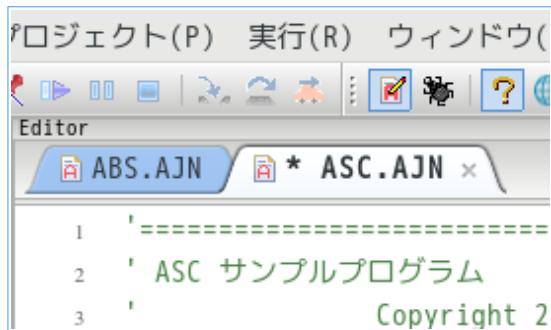
### 4.6.3 プログラムを保存

エディタウインドウで選択されているプログラムを保存します。

#### ■手順

以下の手順を実行します。

1. エディタウインドウで保存したいファイルを選択します。



編集されて保存が可能なファイルは、タブのファイル名の左横に(\*)が付加されます。

2. メニューバーから「ファイル(F)」 → 「保存(S)」もしくは、機能ボタン「保存」をクリックしてください。
3. 編集中のファイル名でプログラムが上書き保存されます。



#### 4.6.4 プログラムを別名で保存

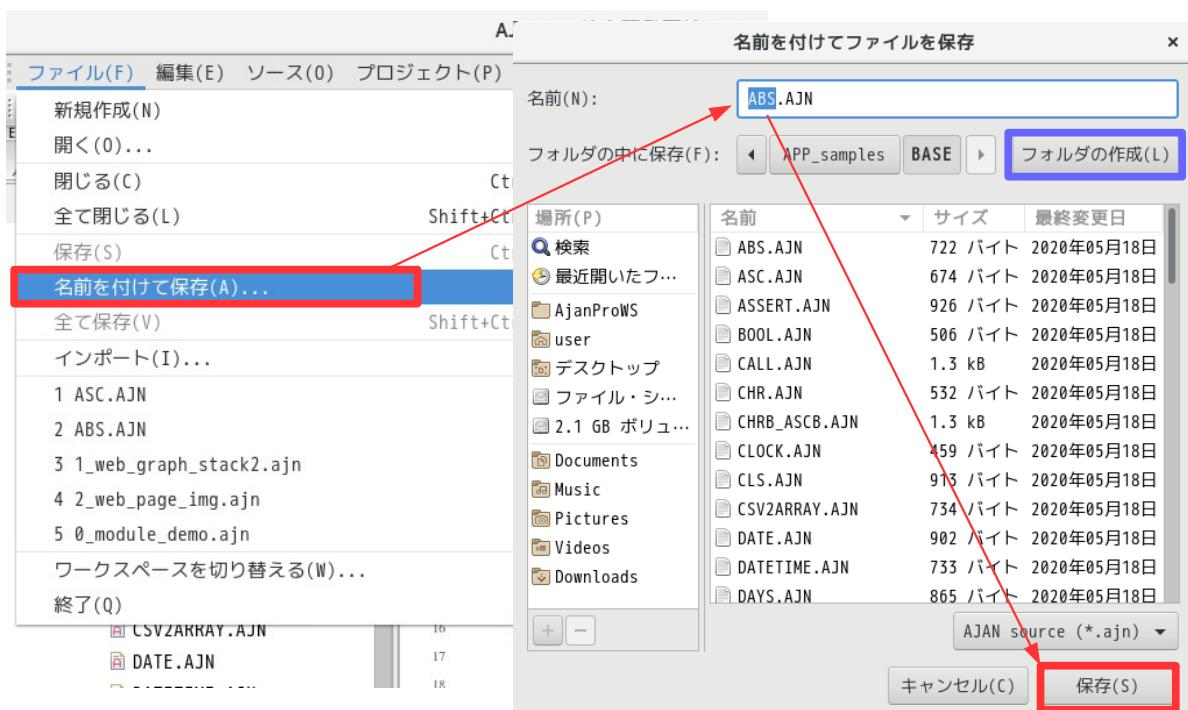
メニューバーからエディタウンドウに表示されているプログラムを別名で保存します。

##### ■手順

以下の手順を実行します。

1. メニューバーから「ファイル(F)」 → 「名前を付けて保存(A)」を選択してください。
2. 保存用のダイアログが表示されるので、名前に新しいファイル名を入力して、「保存(S)」を押して下さい。

※「フォルダ作成」を押すと新しいフォルダも作成できます。



#### 4.6.5 プログラムのインポート

指定したフォルダ直下のAJANファイルを、プロジェクト内の指定フォルダに取り込みます。

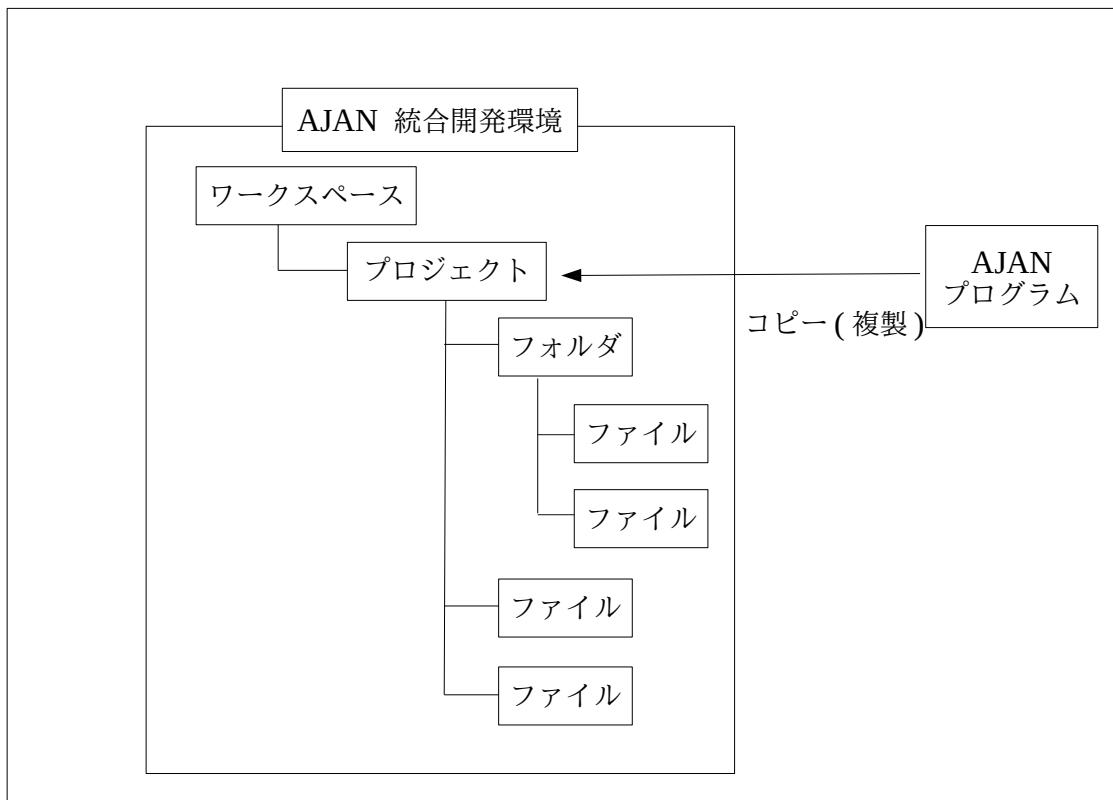


図4-3 インポート概念図

<span style="font-size: 2em; border: 1px solid yellow; border-radius: 50%; padding: 5px;">!</span>	<ul style="list-style-type: none"> <li>プロジェクトを作成した後で、インポート機能を実行してください。</li> <li>ファイルは複製されて取り込まれます。</li> <li>取り込み元のファイルは、削除されません。</li> <li>指定フォルダ直下のファイルがフォルダ構成を含めて全て取り込まれます。 ※指定フォルダ自体は、プロジェクト内に取り込まれません。</li> <li>ファイルを指定して取り込む事は出来ません。</li> <li>取りめるファイルの拡張子は、以下となります。 (.AJN .HTM .CSV .ODG .DAT .XML .PY .GIF .PNG .WEBM .JPG .MP4 .WAV)</li> <li>既にプロジェクト内に同じ名前のファイルがある場合、上書きされません。 上書きしたい場合、インポート前にファイルを削除してください。</li> </ul>
--	---

##### ■手順

- プロジェクトが複数ある場合、取り込み先のプロジェクトをエクスプローラウインドウで選択してください。
- 任意でプロジェクトの下にフォルダを作成してください。  
「ファイル(F)」 → 「新規作成(N)」 → 「フォルダ(D)」  
新規フォルダ作成用のダイアログが起動した後、フォルダ名を入力してください。
- フォルダの下にファイルをインポートします。  
「ファイル(F)」 → 「インポート(I)」  
インポート用ダイアログが起動します。  
※取り込み元フォルダは、事前にエクスプローラで選択したプロジェクトになります。

4. 「参照」ボタンをクリックすると、取り込み元フォルダを選択するダイアログが起動します。

取り込みたいAJANファイルがあるフォルダを選択して、「開く(O)」ボタンをクリックしてください。

以下は、サンプルプログラムのフォルダ(/usr/share/interface/AJANPro/samples)を指定した例です。

5. 取り込み元フォルダのテキストボックスにパスが入力されます。



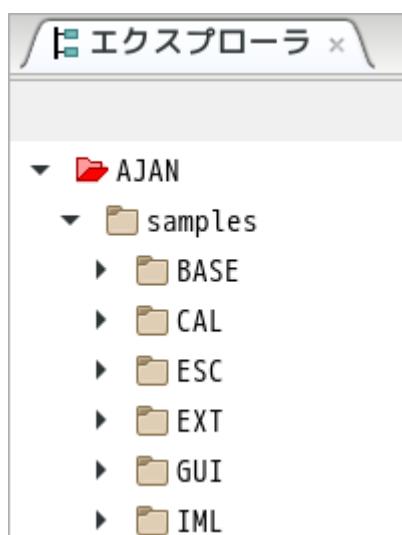
サンプルプログラムをインポートする場合、参照ボタンの直下にある「サンプルプログラム」の文字をクリックすると、取り込み元フォルダのパスとして取り込むことができます。

6. 取り込み先フォルダのツリー選択ボックスから、取り込み先フォルダを選択します。

新たなフォルダに取り込みたい場合、インポート前に「ファイル(F)」 → 「新規作成(N)」 → 「フォルダ(D)」で予めフォルダを作成しておいてください。

7. 「Finish」ボタンをクリックすると、指定された取り込み先フォルダにAJANファイルが取り込まれます。

その他、取り込みが必要なファイルがある場合は、別途手動でコピーを行ってください。



## 4.7 プログラムの実行/中断/終了/コンパイル

AJANプログラムの実行、中断、終了、コンパイルの手順を説明します。

### 4.7.1 プログラムの実行

メニューバーもしくは、機能ボタンでエディタウインドウで選択しているプログラムを実行します。

#### ■手順

1. メニューバーもしくは、機能ボタンから以下の手順を実行します。

- ・メニューバーの場合

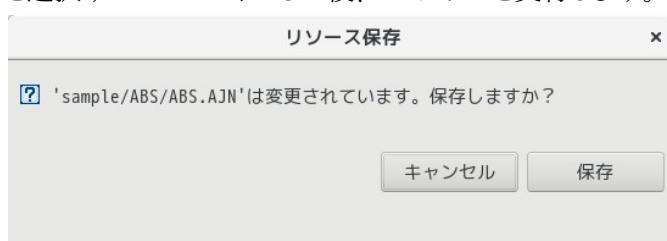
「実行(R)」 → 「実行(R)」を選択すると、プログラムを実行します。

- ・機能ボタン

 (実行)ボタンをクリックすると、プログラムを実行します。

※ プログラムを変更した場合は、リソース保存の確認のダイアログが表示されます。

「保存」を選択するとコンパイルした後、プログラムを実行します。



編集を行って初めて実行する場合、自動的にコンパイルが行われて実行ファイルが生成されます。

#### <コンパイルが正常終了の場合>

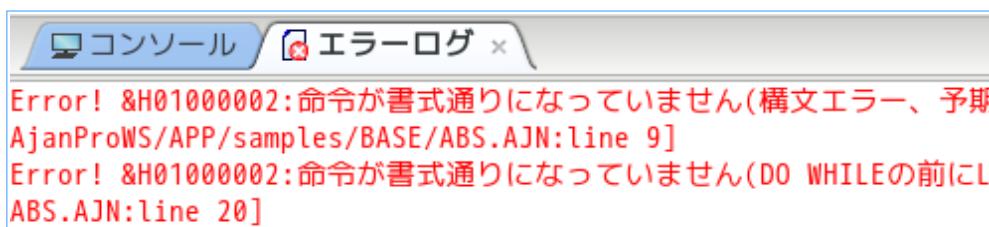
コンソールウインドウに、「コンパイル中...」と表示され、正常終了したら、「コンパイル終了」と表示され、即座にプログラムが実行されます。



※ プログラムでCLSコマンド(画面クリア)を実行する等、コンソールウインドウの表示に影響を与えるコマンドを実行した場合、コンパイルに関するメッセージが消去され表示されない場合があります。

#### <コンパイルが異常終了の場合>

- ・エラーログウインドウ: エラー内容とエラーが発生した行番号を表示します。



	エラーメッセージをダブルクリックする事で、エディタウィンドウの該当箇所にジャンプできます。
---	---

・エディタウィンドウ:ソースコードのエラー箇所に マークが表示されます。

マークにマウスカーソルを重ねると、エラー内容が表示されます。

```

8 FLAG=FALSE
9 DOWHILE FLAG=FALSE
10 構文エラー、予期しない uncertain variable です。予期されるの(
11 LINE INPUT A$
12 '先頭から数値の部分だけを変換

```

#### 4.7.2 プログラムの中断

メニューバーもしくは、機能ボタンで実行中のプログラムを一時的に中断します。

■手順

1.メニューバーもしくは、機能ボタンから以下の手順を実行します。

・メニューバーの場合;

「実行(R)」 → 「中断(S)」を選択すると、プログラムを中断します。

・機能ボタン

 (中断)ボタンをクリックすると、プログラムを中断します。

#### 4.7.3 プログラムの終了

メニューバーもしくは、機能ボタンで実行中のプログラムを終了します。

■手順

1.メニューバーもしくは、機能ボタンから以下の手順を実行します。

・メニューバーの場合;

「実行(R)」 → 「終了(T)」を選択すると、プログラムを終了します。

・機能ボタン

 (終了)ボタンをクリックすると、プログラムを終了します。

	デバッグ中に終了ボタンをクリックした際に、  ボタンが、  ボタン（終了ボタンに砂時計）に変化して、コンソールウィンドウにテキストが表示し続けている状態で即座にプログラムを終了したい場合、  ボタンをクリックして下さい。
---	---

#### 4.7.4 プログラムのコンパイル

プログラムを実行すること無く、実行形式のファイルにするために、コンパイルのみを実行します。

メニューの「プロジェクト(P)」 → 「ファイルのコンパイル(B)」を選択すると、エディタウィンドウで現在選択されているプログラムファイルをコンパイルして実行形式のファイルを作成します。

<コンパイルが正常終了の場合>

コンソールウィンドウに、「コンパイル中...」→「コンパイル終了」→「OK」と表示します。

## &lt;コンパイルが異常終了の場合&gt;

前述の『4.7.1 プログラムの実行』と同様になります。

#### 4.7.5 コマンドライン引数を指定してプログラムを実行

IDEからAJANプログラムを実行する際に引数を渡す事が出来ます。

渡された引数は、AJANコマンドGETCOMMANDLINEARGS\$()で取得する事が出来ます。

##### ■手順

1. 引数を設定したいプログラム(AJAN組込みアプリ)をエディタウィンドウで選択します。

以下の例でHelloAPP.ajnを選択しています。



2. メニューバーから以下の項目を実行します。

「実行(R)」 → 「コマンドライン引数指定(C)...」を選択。

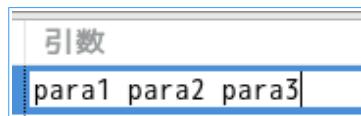
3. コマンドライン引数指定ダイアログが開き、エディタウィンドウで選択したプログラムが選択された状態で開きます。



4. 引数指定領域をクリックすると、入力モードになりますので、任意の文字列を入力して下さい。

※複数の引数を入力する場合、半角スペースで区切って入力して下さい。

※引数に空白を含めたい場合、引数をダブルクオーテーション(")で囲って下さい。



5. 入力を完了するにはEnterキーを押した後、「閉じる」ボタンかESCキーを押して下さい。

6. プログラムを実行、またはデバッグ実行すると、プログラムに指定した引数が渡されます。

<b>!</b>	Webアプリに引数を指定する事は出来ません。エディタウィンドウでAJAN Webアプリ用プロジェクトのプログラムを選択した状態では、「実行(R)」 → 「コマンドライン引数指定...」は無効になり選択出来ません。
----------	--

#### 4.7.6 スーパーユーザーモードでプログラムを実行

弊社のシリアル通信のコマンドを使用してプログラムを作成する場合や、root権限が必要なファイルに書き込みする場合、スーパーユーザーモードで実行する必要があります。

**⚠ 注意**



- ・ **S** マークがコマンドリファレンスのコマンド一覧にある場合、「スーパーユーザーで実行」のチェックボックスにチェックを入れてください。
- ・ スーパーユーザーモードでは強力な権限が付与され、操作を誤ると、システムが起動しなくなる恐れがあります。特に必要な場合は、**スーパーユーザーモードで起動しない事を強く推奨します。**

■手順

1. メニューバーから以下の手順を実行します。

「実行(R)」 → 「スーパーユーザーで実行(U)」のチェックボックスにチェックを付けます。



2. パスワード入力を求められますので、パスワードを入力して、OKをクリックしてください。
3. アプリケーション下部のステータスバーが以下の表示になり、スーパーユーザーモードであることを通知します。



以降のプログラムの実行は、root権限が付与されます。

(パスワードはチェックボックスのチェックを外すか、IDEを終了するまで保持されますので、都度、パスワードの入力を求められることはありません。)



- ・工場出荷時のパスワードは、以下の様に設定されています。  
userアカウントの場合: user  
※アカウントを追加された場合、後述の「sudoにアカウントを登録する」を参照してください。

### sudoにアカウントを登録する

実行するプログラムをroot権限に昇格させるのにsudoコマンドを使用するため、sudoが使えるように、ユーザー アカウントを追加する必要があります。以下の手順で、アカウントを登録してください。  
※下記作業について管理者にご確認ください。

1. デスクトップから左上の「アプリケーション」→「システムツール」→「端末」を起動する。
2. suコマンドで、スーパーユーザーになる。  
su[Enter]  
パスワードを入力[出荷時は、"root"と入力する]

\$ su

パスワード:新しいパスワードを入力(入力しても表示されません)

3. visudoを実行する。

# visudo

4. 以下の記述の箇所まで、スクロールする。

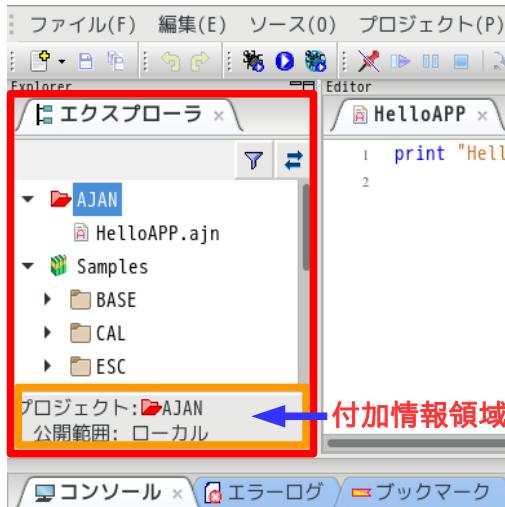
```
#User privilege specification  
root  ALL=(ALL:ALL) ALL  
user  ALL=(ALL:ALL) ALL
```

5. 記述を参考にして、自身のアカウントを次行に追加する。
6. 追加したら、設定を書き込むためにキーボードでCtrl+0(オー)を押す。
7. 「書き込むファイル: /etc/sudoers.tmp」と表示されるので、Enterキーを押すと、設定が書き込まれる。
8. キーボードでCtrl+Xを押して、visudoを終了する。



## 4.8 エクスプローラウィンドウ

ファイルメニューの新規作成(プロジェクト、フォルダ、ファイル)や、インポート機能にてファイルをAJAN 統合開発環境に取り込むと、ファイル構成がツリー状に表示されます。



付加情報領域

プロジェクト: AJAN  
公開範囲: ローカル

エクスプローラで選択したファイルが属するプロジェクトが表示されます。  
又、Webアプリが、本PC内でのみ実行出来る(ローカル)のか、外部から実行出来る(外部公開)のかの付加情報が表示されます。

### 4.8.1 ポップアップメニュー

本ウィンドウ上でマウスを右クリックするとポップアップメニューが開きます。



メニューにて以下の操作が行えます。

機能	説明
開く	エクスプローラウィンドウで選択した対象に応じて、開く動作を行います。 ・ファイル: エディタウィンドウで開きます。 ・フォルダ: フォルダを展開します。  Altキーを押しながら実行すると、ファイル(nautilus)で開きます。
コピー	エクスプローラウィンドウで選択したファイルのパスをクリックボードにコピーします。 アイテムは複数選択が可能です。
貼り付け	クリップボードから文字列を取得して、有効なパスならエクスプローラウィンドウの選択した場所にファイルをコピーします。  まず、上記のコピーを実行して、ファイルのパスをクリップボードにコピーして下さい。
削除	エクスプローラウィンドウで選択したファイルを削除します。
名前の変更...	エクスプローラウィンドウで選択したファイルの名前を変更します。
更新	プロジェクト内のファイル構成をエクスプローラウィンドウに反映します。 プロジェクトに対して、ファイルを直接追加したり、ファイル名を変更したり削除を行った場合に実行してください。
変換	AJANファイルに対して、「Webアプリへ」か「組込みアプリへ」のいずれかが選択できます。AJANファイルを実行した際にWebアプリとしてWebブラウザ上で実行する

	か、通常実行するかを切り替える事が出来ます。
設定...	プロジェクトに関する設定が行えます。
保存履歴と比較...	現在のファイルと、過去に保存したファイルとの差異を確認する事が出来ます。

	<ul style="list-style-type: none"> <li>「コピー」と「貼り付け」を同じ場所で行った場合、ファイルの複製が可能です。 フォルダに対しては複製とはならず、コピーが出来ない旨のメッセージが表示されます。</li> <li>「Samples」は読み込み専用となっており、「貼り付け」「削除」「名前の変更」の操作は、項目が無効になり実行できません。</li> </ul>
---	---

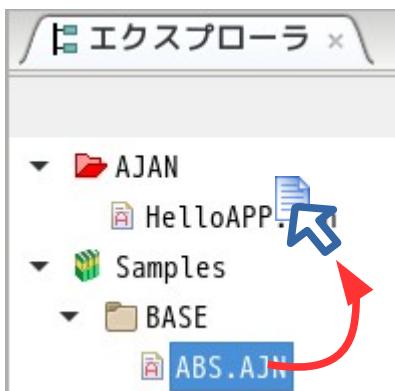
#### 4.8.2 機能ボタン



アイコン	項目名	説明
	フィルタ	エクスプローラで非表示にするファイルの拡張子を選択します。このボタンをクリックすると、IDE起動時に収集された拡張子の一覧が表示されますので、 <b>非表示</b> にしたいファイルの拡張子にチェックを付けて、OKボタンを押して下さい。 フィルタが有効だと、下のアイコンのように赤点が点灯します。
	エディタとリンクする	このボタンはトグルボタンになっており、押し込まれた状態だと、エクスプローラとエディタのリンクが有効になります。リンク状態だと、エクスプローラで選択したファイルをエディタで開いている場合、そのファイルが選択状態になります。又、エディタのタブを別のファイルに切り替えると、エクスプローラの該当ファイルが選択状態になります。

### 4.8.3 ドラッグ&ドロップ

本ウィンドウ上でファイル(フォルダ)のドラッグ&ドロップすることで、以下の操作が可能です。



機能	説明
ファイル / フォルダ 移動	<p>1. エクスプローラーウィンドウ上でファイル / フォルダをクリックして選択します。 複数ファイル / フォルダを選択するには、連続している場合はShiftキーを押しながらクリック、不連続ならCtrlキーを押しながらクリックしてください。</p> <p>2. 選択したファイル / フォルダをドラッグして、同一プロジェクト内の移動したいフォルダにドロップしてください。</p> <p>3. 移動確認のメッセージボックスが表示されますので、「はい」「いいえ」「キャンセル」を選択してください。 「いいえ」は、個々のファイルに適用され、「キャンセル」は複数ファイル選択時に以降の操作全てがキャンセルされます。 移動するファイルを開いている場合、移動確認の前に、ファイルクローズのメッセージボックスが表示されますので、「はい」か「いいえ」を選択してください。 「いいえ」を選択した場合、全てのファイル移動がキャンセルされます。</p>
ファイル / フォルダ コピー	基本的な操作は上記のファイル / フォルダ移動と同様ですが、2. のドラッグする操作をCtrlキーを押しながら行って下さい。

	<ul style="list-style-type: none"> <li>「ファイル」を同じ場所にドロップした場合、そのファイルの複製が可能です。 フォルダを同じ場所にドロップしても複製とはならず、コピーが出来ない旨のメッセージが表示されます。</li> <li>「Samples」は読み込み専用になっており、その中にファイルやフォルダをコピーしたり、改変する事はできません。</li> </ul>
--	---

#### 4.8.4 保存履歴と比較

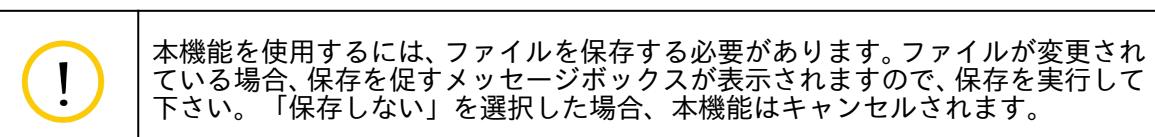
本機能は現在のファイルと、過去に保存したファイルとの差異を確認する事が出来ます。

##### ■手順

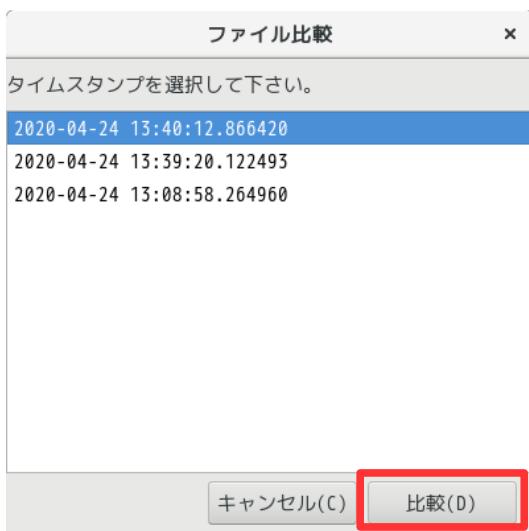
- エクスプローラーウィンドウ上で、過去の編集内容と比較したいファイルを選択して、右クリックしてポップアップメニューを開きます。



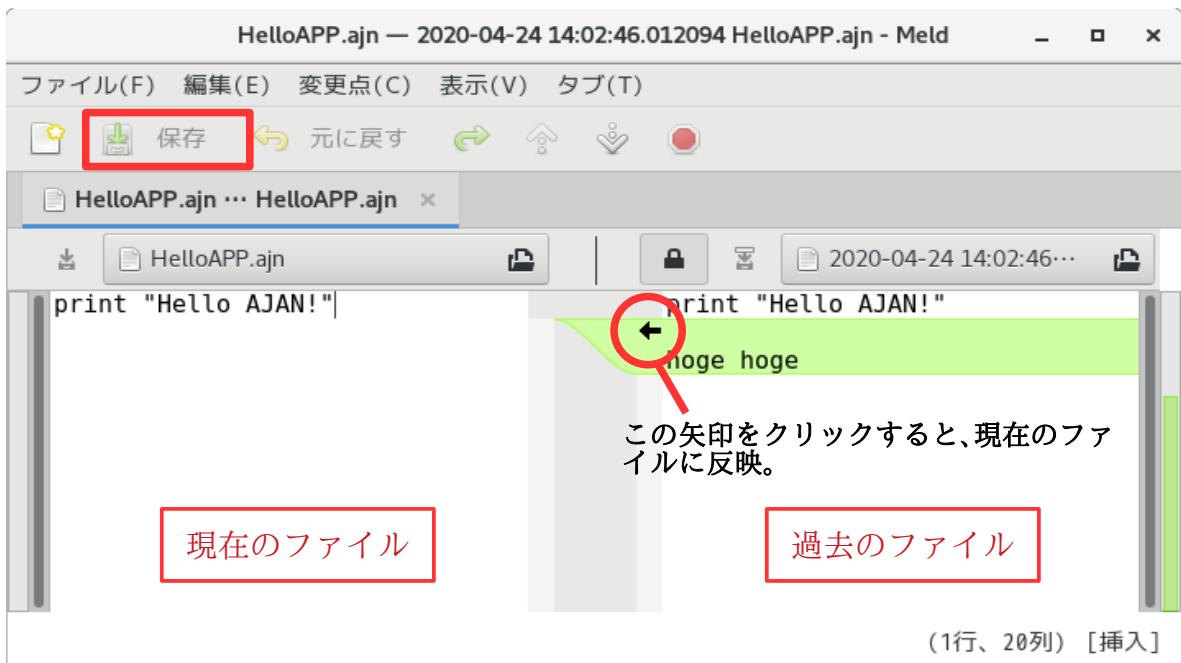
- 「保存履歴と比較...」をクリックします。



- 以下の様に過去に保存した日時が表示されますので、比較したい日時を選択して、「比較(D)」をクリックします。



4. 以下の様にファイル比較ツールが起動して、差異のある箇所がハイライト表示されます。



5. 差異のある箇所で、過去のファイルの内容を現在のファイルに反映したい場合、ハイライト箇所にある矢印をクリックすると、現在のファイルに反映する事が出来ます。  
反映すると、保存ボタンが有効になりますので、保存ボタンをクリックして下さい。
6. 終了したい場合、ファイル比較ツールの右上の×印をクリックして終了させて下さい。
7. 3.の日時選択メニュー画面に戻りますので、続けて他の日時と比較したい場合は、同様の操作を繰り返して下さい。
8. ファイル比較を終了するには、日時選択メニュー画面のキャンセルボタンをクリックして下さい。

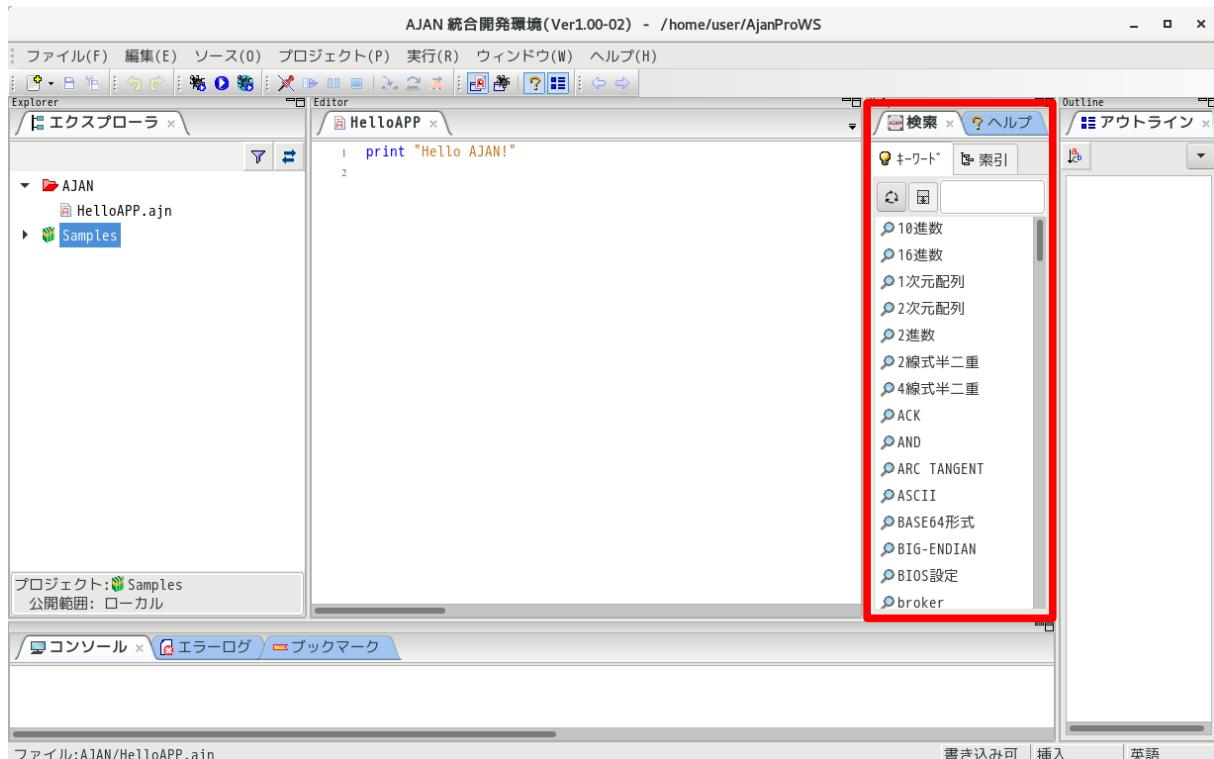
<span data-bbox="250 1417 330 1495">!</span>	<p>ソースコードの差分表示ツールとして、デフォルトでGNOME Meldを使用しています。</p> <p>詳細な使用方法に付きましては、<a href="https://meldmerge.org">https://meldmerge.org</a>をご参照下さい。</p>
--	--

## 4.9 ヘルプ機能

AJANコマンド等の検索や情報を表示する機能を提供します。

### 4.9.1 ヘルプウィンドウ

画面レイアウトがエディタの場合、下記の赤線の箇所(デフォルト時)にヘルプウィンドウが表示されます。



※ヘルプ機能を使用する際は、左右の枠をドラッグして適切な大きさに変更してご使用ください。

ヘルプウィンドウは、以下のタブで構成されています。



主項目	副項目	説明
検索	キーワード	AJANコマンドをキーワードから検索することができます。
	索引	AJANコマンドを分類から検索することができます。
ヘルプ	なし	オンラインヘルプ機能を提供します。索引やキーワードで絞り込まれたコマンドをクリックすると、このタブに切り替わって説明が表示されます。

ヘルプウィンドウが表示されていない場合、以下の方法でヘルプウィンドウを表示してください。

#### ■機能ボタンを使用する場合



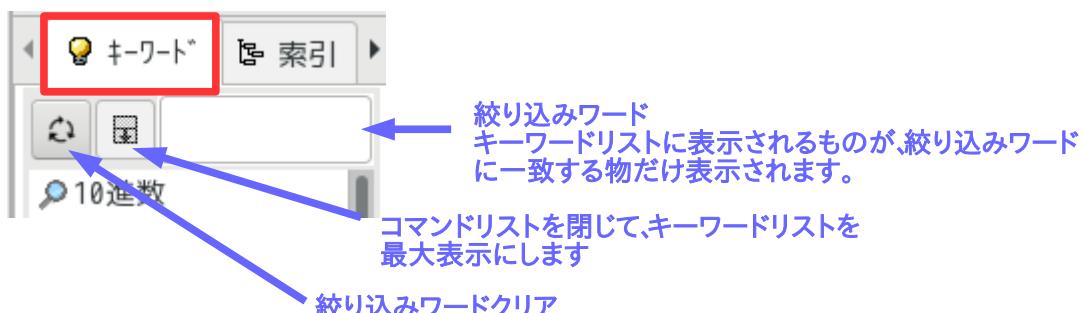
#### ■メニューバーから開く場合

「ヘルプ(H)」→「ヘルプを開く(H)」を選択してください。

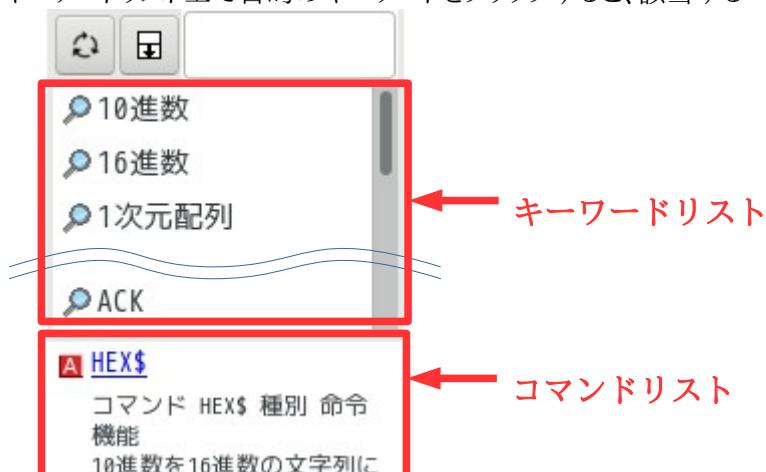
## 4.9.2 キーワード検索

AJANコマンドを予め用意されたキーワードから検索する機能を提供します。

「キーワード」タブを選択して下さい。



キーワードリスト上で目的のキーワードをクリックすると、該当するコマンドがコマンドリストに表示されます。



コマンド名の青字の箇所をクリックすると、ヘルプタブに切り替わり、コマンド説明が表示されます。



元のキーワード検索に戻るには、➡をクリックするか、「検索」タブをクリックして下さい。



先頭に@(アットマーク)を付けて絞り込みワード（最低文字数2文字）を入力すると、AJANコマンドを直接検索することができます。

### 4.9.3 索引検索

AJANコマンドを分類から絞り込んで検索する機能を提供します。

「索引」タブを選択して下さい。

目的からコマンドを検索する逆引き検索と、索引から検索する2つの手段を提供します。



#### 機能ボタンの説明



アイコン	項目名	説明
barcode	トップに戻る	検索階層をリセットして、最初の選択画面に戻します。
>GUI	検索階層ボタン	検索の現在の階層を表します。ボタンの表示名は、選択した項目名が表示されます。 一つ上の階層のボタンを押すと、その階層に戻る事ができます。

例題として、数値を16進数の数字文字列に変換するAJANコマンドを検索してみましょう。



キーワードを選択すると、次のキーワードが表示されますので、関連ありそうなキーワードを次々と選択していきます。



最終選択肢に到達すると、以下の様にコマンド候補が表示されます。

[10進数を2進数の文字列に変換する](#)  
10進数を2進数の文字列に変換する。コマンド BIN\$ 種別 命令機能  
10進数を2進数の文字列に変換します。書式 = BTN\$() 注意

[10進数を16進数の文字列に変換する](#)  
10進数を16進数の文字列に変換する。コマンド HEX\$ 種別 命令機能  
10進数を16進数の文字列に変換します。書式 = HEX\$() 注意

説明から16進数に変換するコマンドがこれと確認出来ましたので、青字の部分をクリックします。

ヘルプタブに切り替わり、コマンド説明が表示されます。



**10進数を16進数の文字列に変換する。**

### コマンド

HEX\$

元のキーワード検索画面に戻るには、をクリックするか、「検索」タブをクリックして下さい。



#### 4.9.4 ヘルプタブ

オンラインヘルプ閲覧機能を提供します。



##### 機能ボタンの説明

<見出し領域>



<コンテンツ領域>



アイコン	項目名	説明
	開く	見出し領域で選択している項目を開きます。
	目次を展開	見出し領域で選択している項目を全て展開します。
	目次を折りたたむ	見出し領域で選択している項目を全て折りたたみます。
	コンテンツ領域表示	オフの場合、状況に応じて見出し領域かコンテンツ領域のいずれかが表示されます。 オンの場合、コンテンツ領域が常に表示されます。
	戻る	検索タブの検索結果からヘルプを開いた場合、クリックすると、検索タブに戻ります。 ヘルプ閲覧時は、前回のページに戻ります。
	進む	戻るボタンで戻る前のページを表示します。
	見出しとリンクする	見出し領域が非表示で、コンテンツ領域のみ表示されている時にボタンが出現します。 このボタンを押すと、見出し領域に切り替わり、該当する見出しが選択状態になります。
	フォントを大きく	コンテンツ領域のフォントサイズを拡大します。
	フォントを小さく	コンテンツ領域のフォントサイズを縮小します。

ヘルプウィンドウの横幅が狭い場合、閲覧しやすいようにコンテンツ領域は非表示になっています。

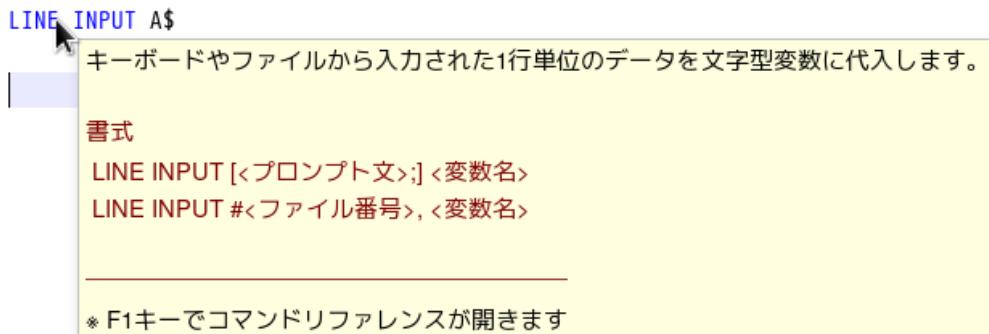
見出し領域で項目を選択して ボタンを押すか、ダブルクリックすると、見出し領域が非表示になり、コンテンツ領域が代わりに表示されます。

ヘルプウィンドウの横幅を一定以上に広げると、コンテンツ領域が常時表示されるようになります。

強制的にコンテンツ領域を常に表示させたい場合、 ボタンをオンにします。

## 4.9.5 コマンドツールチップ機能

本機能は、エディタ上のAJANコマンドにマウスカーソルを重ねると、吹き出し表示でコマンドの簡易ヘルプが確認出来ます。



## 4.9.6 コマンドヘルプジャンプ機能

本機能を使用すると、コマンドツールチップに表示されているコマンドもしくは、エディタ上のキーボードカーソル位置のコマンドのリファレンスを瞬時に開く事ができます。

### ■ 使用方法

1. マウスカーソルをAJANコマンド上に移動して前述のコマンドツールチップが表示された状態にします。もしくは、ツールチップが表示されていない状態でキーボードカーソルをコマンドに接するように移動させます。
2. 以下の3種類のいずれかの操作を行う。
  - ・メニューバーから、「ヘルプ(H)」 → 「コマンドヘルプを開く(J)」をクリック。
  - ・右クリックでポップアップメニューを開いて、「コマンドヘルプを開く」をクリック。
  - ・ショートカットキー(F1)を押す。
3. 以下の専用PDFビューワーが起動して、AJANコマンドの該当ページが表示されます。

**命令**

**機能** 文字列や数値等のデータを画面、またはファイルに出力します。

**書式1** PRINT <(1)文字列/数値> [; | , <(1)文字列/数値> …] [; | ,]  
文字列や数値等のデータを画面に出力します。

**書式2** ? <(1)文字列/数値> [; | , <(1)文字列/数値> …] [; | ,]  
書式1と同じです。「?(クエスチョンマーク)」で「PRINT」の代替えとします。

**書式3** PRINT #<②ファイル番号>, <(1)文字列/数値> [; | , <(1)文字列/数値> …] [; | ,]  
文字列や数値等のデータをファイルに出力します。

**書式4** ? #<②ファイル番号>, <(1)文字列/数値> [; | , <(1)文字列/数値> …] [; | ,]  
書式3と同じです。「?(クエスチョンマーク)」で「PRINT」の代替えとします。

**パラメータ**

(1) [文字列/数値] [; | ,] [数値]  
出力先の文字列/数値を指定します。セミコロン(;)で区切ると、各文字列/数値を連続して出力します。カンマ(,)で区切ると、各文字列/数値間にタブを出力します。最後にセミコロンやカンマを指定すると、改行が起ります。

(2) [ファイル番号] [; | ,] [数値]  
出力先のファイル番号を指定します。ファイル番号に変数を与える場合、他の書式の区別に、「#<変数名>」の記述としてください。

**備考** ファイルに出力する場合、OPENコマンドでFOR OUTPUT/APPENDを指定してください。

**注意** 構造体の表示はできません。

**使用例1** A = 50  
PRINT "A=%"; A  
A=50 と表示します。

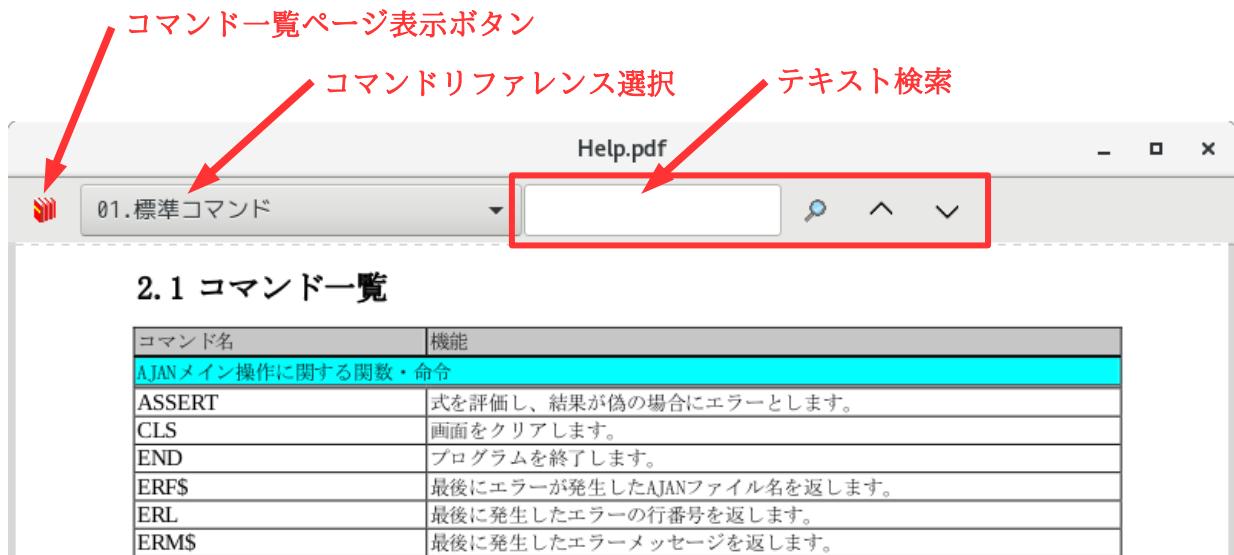
**使用例2** A = 50  
B = 60  
? A, B  
50 60 と表示します。

**使用例3** PRINT #1, "TEST OK"  
ファイル番号1で開いているファイルへ TEST OK という文字列を書き込みます。

**使用例4** NUM% = 1  
PRINT #NUM%, "TEST OK"  
使用例3のファイル番号の指定を、NUM%変数に置き換えた例です。

**使用例5** A = 50  
B = 60

#### 4.9.7 専用PDFビューワ



項目名	説明
コマンド一覧ページ表示ボタン	現在開いているコマンドリファレンスの「コマンド一覧」ページが表示されます。
コマンドリファレンス選択	現在開いているコマンドリファレンスの名称が表示されます。 クリックすると、コマンドリファレンス一覧が表示されて切り替える事が出来ます。
テキスト検索	現在開いているコマンドリファレンスの内容をテキスト検索できます。 テキストボックスに文字を入力して、Enterキーを押すか、ボタンをクリックして下さい。 次の検索結果に進んだり戻るには   ボタンをクリックして下さい。

## 4.10 アウトライン機能

エディタウィンドウで現在選択しているファイルの変数、構造体、ファンクション、サブルーチン、スレッドの宣言が一覧で表示されます。



### 4.10.1 シンボル一覧

アイコン	型
●	<b>グローバル変数宣言</b> 以下の宣言が該当します。 DIM, BOOL, DATETIME, STRUCT, CONST, CONST BOOL, CONST DATETIME, DICT, DICT BOOL, DICT STRUCT, DICT DATETIME, LIST, LIST BOOL, LIST STRUCT, LIST DICT, LIST DICT BOOL, LIST DICT STRUCT, LIST DICT DATETIME
◎	<b>ローカル変数宣言</b> 以下の宣言が該当します。 LOCAL, LOCAL BOOL, LOCAL CONST, LOCAL CONST BOOL, LOCAL CONST DATETIME, LOCAL DICT, LOCAL DICT STRUCT, LOCAL LIST, LOCAL LIST BOOL, LOCAL LIST DICT, LOCAL LIST DICT BOOL, LOCAL LIST DICT DATETIME, LOCAL LIST DICT STRUCT, LOCAL LIST STRUCT, LOCAL STRUCT
#	<b>構造体定義の宣言</b> 以下の宣言が該当します。 DEFINE STRUCT 構造体定義で表示されます。 構造体の実体の宣言では、グローバル変数、もしくはローカル変数のアイコンが表示されます。
S	<b>サブルーチン宣言</b> 以下の宣言が該当します。 SUB
F	<b>ファンクション宣言</b> 以下の宣言が該当します。 FUNCTION
T	<b>スレッド宣言</b> 以下の宣言が該当します。 DEFINE THREAD

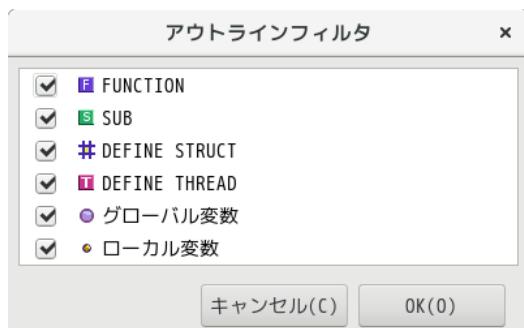
#### 4.10.2 シンボル並び替え

並び替えボタン  (トグル) をクリックすることで、シンボルの並びを切り替える事が出来ます。

状態	説明
オフ(  )	ソースコードに定義された順に表示されます。
オン(  )	アルファベット順に並びます。

#### 4.10.3 フィルタメニュー

フィルタメニュー(▼)をクリックすると、下記のダイアログボックスが開きます。



チェックが付いているシンボルが表示されますので、非表示にしたい場合、そのシンボルのチェックボックスをクリックして、チェックを外した状態にしてOKボタンをクリックして下さい。

#### 4.10.4 カーソル移動

一覧をクリックすると、その宣言箇所にジャンプします。

元のカーソル位置に戻るには、以下のボタンをクリックして下さい。

	アウトライン機能でカーソル移動した際に、クリックする毎にカーソルを以前の位置に戻します(最大5箇所)。
	カーソル位置を  をクリックする前の位置に戻します(最大5箇所)。



## 4.11 プログラムのデバッグ

### 4.11.1 デバッグの実行

メニューバーもしくは、機能ボタンでデバッグを実行します。

#### ■手順

1. メニューバーもしくは、機能ボタンから以下の手順を実行します。

- ・メニューバーの場合；

「実行(R)」 → 「デバッグ実行(D)」を選択すると、デバッグを実行します。

- ・機能ボタン

 (デバッグ実行)ボタンをクリックすると、デバッグを実行します。

※ エディタモードになっている場合、自動的にデバッグレイアウトに切り替わります。

2. プログラムのコンパイルが完了後に実行されます。

	<p>ブレークポイントが設定されていない場合、通常実行と同じ動作になります。 確認したいプログラムの場所に、後述のブレークポイントを設定して下さい。</p>
---	--

### 4.11.2 画面レイアウトの切り替え

デバッグ実行すると、画面レイアウトがデバッグに適したレイアウトに自動的に切り替わります。

デバッグが終了すると、プログラム編集に適したエディタレイアウトに戻ります。

エディタレイアウトに手動で戻すには、以下の操作を行います。

- ・メニューバーの場合；

「ウインドウ(W)」 → 「レイアウト(L)」 → 「レイアウトの切り替え(C)」 → 「エディタ(E)」  
を選択します。

- ・機能ボタン

 (エディタ画面)ボタンをクリックします。

『4.3.3 画面レイアウトの切り替え』も併せて参照して下さい。

### 4.11.3 ブレークポイント設定/解除

ブレークポイントを設定すると、実行中のプログラムを指定行で停止させる事ができます。

ブレークポイント設定/解除は以下の手順で行います。

#### ■手順

- エディタウィンドウ上で、ブレークポイントを設定したい行の行番号をダブルクリックすると、ブレークポイント  が設定されます。

ブレークポイントウィンドウに設定箇所が表示されます。



ブレークポイントを解除する場合は、ブレークポイントをダブルクリックすると解除されます。

ブレークポイントウィンドウで右クリック→削除でも可能です。

	<p>全てのブレークポイントを一時的に無効にしたい場合、下記のアイコンをクリックしてオンにしている間、無効にできます。</p> 				
!	<table border="1"> <tbody> <tr> <td></td><td>ブレークポイントをスキップしない</td></tr> <tr> <td></td><td>ブレークポイントをスキップする</td></tr> </tbody> </table>		ブレークポイントをスキップしない		ブレークポイントをスキップする
	ブレークポイントをスキップしない				
	ブレークポイントをスキップする				
!	ブレークポイントは、空白行やコメント行にも設定出来ますが、ブレークポイントとしては無効となります。プログラムが書かれた行に設定して下さい。				
!	ブレークポイントを設定する以外に、プログラム中にSTOPコマンドを挿入することで、STOPコマンドに到達した時点で停止させる事ができます。				

2. ブレークポイントを設定した行番号で右クリックしてポップアップメニューを表示することで、メニューによる設定が可能です。

```

8
9 FLAG=FALSE
10 DO WHILE ERASE=0
11   ブレークポイント削除
12   ブレークポイント無効
13   ブレークポイント詳細...
14   文字列で受け渡されていないのに@に
15   IF VAL(A$) <> 0 OR ASC(A$) = 48

```

「ブレークポイント無効」: 設定したブレークポイントに到達しても止まらずスキップされます。

「ブレークポイント削除」: 設定したブレークポイントを削除します。

「ブレークポイント詳細」: ブレークポイントに条件式を設定できます。

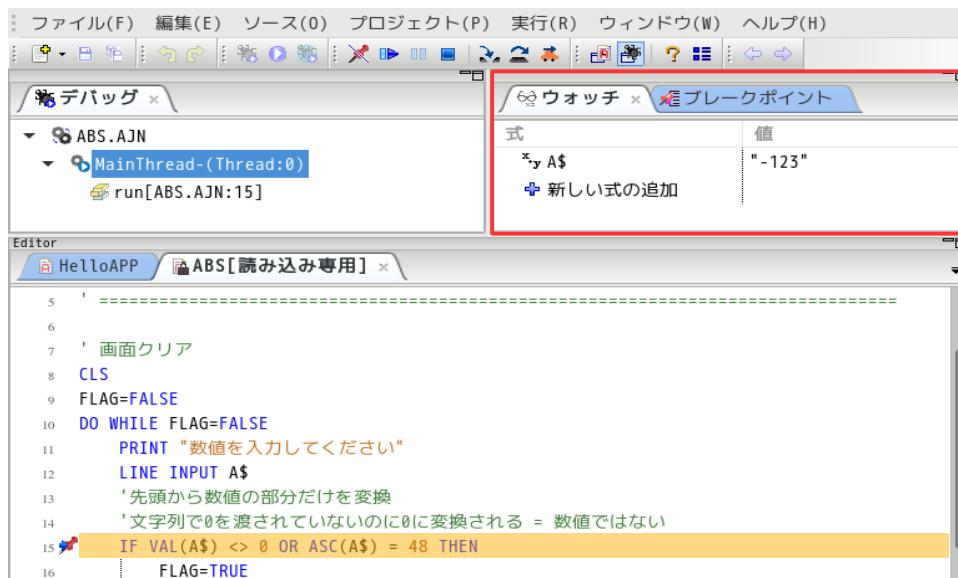
以下は、「i=100」になると停止する例です。



#### 4.11.4 変数ウォッチ

変数ウォッチとは、実行中のプログラムの変数値を表示する機能です。

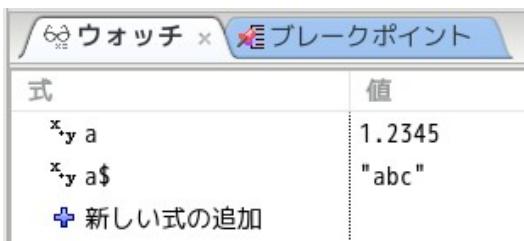
PRINTコマンドで変数値を表示させるデバッグ方法とは異なり、プログラムを変更する必要がなく、手軽にデバッグできます。



変数の型によって、指定した変数名と値が以下の様に表示されます。

#### 変数

例) 数値型変数=a、文字型変数=a\$



<span style="font-size: 2em;">!</span>	<p>文字型変数の値にUTF-8で表現出来ない文字が含まれる場合、以下の様なバイトデータで表現されます。</p> <table border="1"> <thead> <tr> <th>式</th><th>値</th></tr> </thead> <tbody> <tr> <td>x,y STR_DTA\$</td><td>"[bytedata]&amp;H0 &amp;H1 &amp;H2 &amp;H3 &amp;H4 &amp;H5 &amp;H6 &amp;H7 &amp;H8</td></tr> </tbody> </table>	式	値	x,y STR_DTA\$	"[bytedata]&H0 &H1 &H2 &H3 &H4 &H5 &H6 &H7 &H8
式	値				
x,y STR_DTA\$	"[bytedata]&H0 &H1 &H2 &H3 &H4 &H5 &H6 &H7 &H8				

### 配列変数

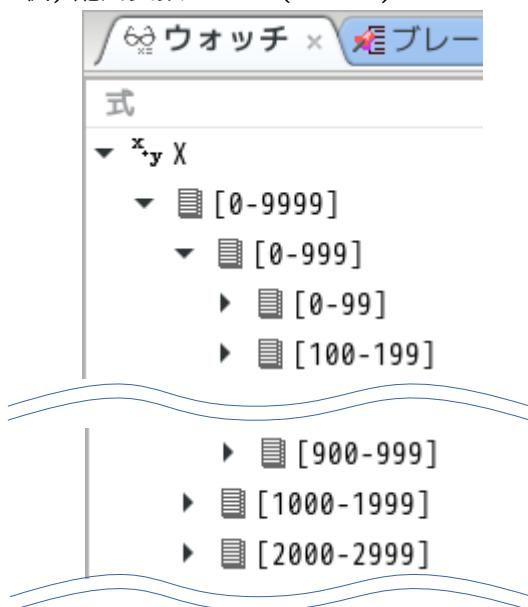
例) 配列変数 DIM x(3,10)

式	1次元目	値
▼ x,y X		[[0,0,0,0,0,
▶ ⚡ [0]		0
▼ ⚡ [1]	2次元目	0
▶ ⚡ [0]		0
▶ ⚡ [1]		0
▶ ⚡ [2]		0

多次元配列の場合、次元毎に上記の図のようにツリー形式で表示されます。

要素数が100個を超えると、100単位(以降、10の倍数単位)で折りたたまれます。

例) 配列変数 DIM x(10000)



要素数1000未満は100単位で折りたたまれ、1000以上～9999は1000単位で折りたたまれます。

## 構造体変数

例) 構造体変数

```
DEFINE STRUCT ABC
```

```
    X
```

```
    Y
```

```
    Z
```

```
END STRUCT
```

```
STRUCT ABC PARAM
```

式	値
▼ * PARAM	{ X : 0, Y : 0, Z : 0 }
● X	0
● Y	0
● Z	0

構造体変数には、変数名の左横にアイコン#が表示され、▶をクリックすると、構造体のメンバ変数がツリー形式で表示されます。

!	表示が重たいと感じる場合、不要な箇所のツリーを閉じると、表示速度が改善される場合があります。
---	--

## 連想配列

例) 連想配列

```
DICT PARAM
```

```
PARAM("abc") = 123
```

```
PARAM("def") = 456
```

式	値
▼ DICT PARAM	(abc)=123, (def)=456
● "abc"	123
● "def"	456

連想配列は変数名の横にアイコン☰が表示され、▶をクリックすると、キーの名前がツリーの枝として表示されます。

## a) ウオッヂ変数を追加する

1. ウオッヂウインドウ内にある「新しい式の追加」をクリックします。



2. 「新しい式の追加」の文字が消えて、カーソルが点滅状態になります



3. 変数名を入力したらEnterキーを押して下さい。



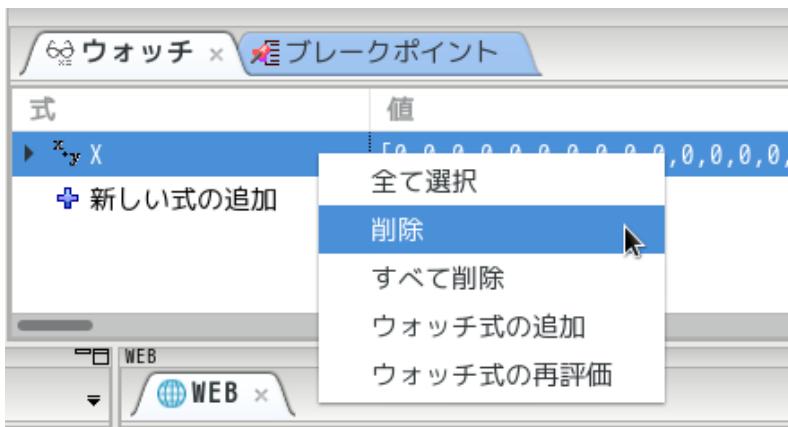
4. デバッグを実行すると、変数の値が表示されます。



<span style="font-size: 2em;">!</span>	変数名として評価できない名前(例:コマンド、関数名、変数でない名前等)は、値の欄に"????"が表示されます。
<span style="font-size: 2em;">!</span>	値に改行文字が含まれる場合、記号に変換されて1行で表示されます。 LF(Line Feed)=↓、 CR(Carriage Return)=↵
<span style="font-size: 2em;">!</span>	文字型変数の値にUTF-8で表現出来ない文字が含まれる場合、以下の様なバイトデータで表現されます。
	<p>The screenshot shows the Watch window interface. The title bar says 'ウォッヂ' and 'ブレークポイント'. The main area has two columns: '式' (Expression) and '値' (Value). The expression column shows 'x,y STR_DTA\$'. The value column shows the binary data as a series of hex values: "[bytedata]&amp;H0 &amp;H1 &amp;H2 &amp;H3 &amp;H4 &amp;H5 &amp;H6 &amp;H7 &amp;H8".</p>

## b) ウオッヂ変数を削除する

ウォッヂ変数を変数エリアから削除するときは、削除したい変数名を右クリックして、「削除」を選択します。



## c) 変数の値を変更する

デバッグ中に任意の変数の値を書き換える事が出来ます。

書き換えたい変数はウォッヂに追加されている必要があります。

追加されていない場合は、『a) ウォッヂ変数を追加する』を参照して、予め追加して下さい。

ウォッヂに追加されている状態で、「値」のセルをクリックすると、以下の様に編集状態になりますので、

書き換え後にEnterキーを押して確定して下さい。キャンセルする場合はESCキーを押して下さい。



#### 4.11.5 トレースを実行する

トレース実行とは、プログラムを1行ずつ実行できる機能です。1行ずつ実行させることで、プログラムの流れや変数の状態を確認しやすくなります。トレース実行は  (中断)ボタンやブレークポイント等で停止した状態で実行できます。

メニューバーもしくは、機能ボタンでトレースを実行します。

##### ■手順

1. メニューバーもしくは、機能ボタンから以下の手順を実行します。

- ・メニューバーの場合;  
「実行(R)」から選択します。
- ・機能ボタンの場合:  
メニューバーと機能ボタンの対応は以下となります。

メニューバー	機能ボタン	機能
ステップ・イン		サブルーチンや関数内部もトレースします。
ステップ・オーバー		サブルーチンや関数内部はトレースされません。
自動トレース		自動的にステップ・インを連続実行します。
中断		プログラムの実行を中断します。
再開		プログラムの実行を再開します。

## 4.11.6 スタックトレース機能

スタックトレースとは、関数やサブルーチンがブレークポイント等で停止した際に、それらが、どこから実行されたか履歴を表示する機能です。

CALL.AJNというソースファイルに、以下のコードを書いて確認します。。

```

5   ' Copyright 2018 Interface
6   =====
7
8
9   SUB ROUTINE3
10  SLEEP 1
11  END SUB
12
13  SUB ROUTINE2
14    CALL ROUTINE3
15  END SUB
16
17  SUB ROUTINE1
18    CALL ROUTINE2
19  END SUB
20
21  CALL ROUTINE1
22
23  END
24

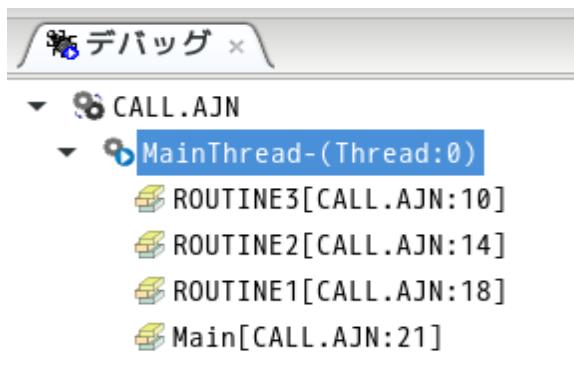
```

このコードは、メインルーチンからサブルーチンROUTINE1がコールされ、ROUTINE1からROUTINE2がコールされ、ROUTINE2からROUTINE3がコールされ、ROUTINE3でSLEEP命令(10行目)が実行されていることが確認できます。

この10行目にブレークポイントを設定します。

この条件でデバッグ実行すると、10行目でブレークが発生します。

この状態でデバッグウィンドウを確認すると、以下の状態が確認できます。



このプログラムはシングルスレッドのプログラムのため、MainThread(Thread:0)に4個のスタックが表示されていることが確認できます。  
上に行くほど直近に実行された処理になります。

一番下のスタックを確認すると、CALL.AJNの21行目からROUTINE1が呼び出されたことが確認できます。ROUTINE1の18行目からROUTINE2が呼び出されたことが確認できます。

ROUTINE2の14行目からROUTINE3が呼び出されたことが確認できます。

ROUTINE3の10行目で停止していることが確認できます。

それぞれのスタックをマウスでクリックすると、ソースコードのその行にジャンプすることができます。

このように、スタックトレースを確認すると、どの順番に実行されたか判断できます。

この例ではROUTINE3は1箇所からしかコールされていませんが、もし複数の箇所からコールされている場合、ROUTINE3で問題が発生した際に、どの経路から呼ばれた場合に問題が発生するかを辿ることができます。

#### 4.11.7 WebブラウザにURLを直接指定した場合にデバッグする方法

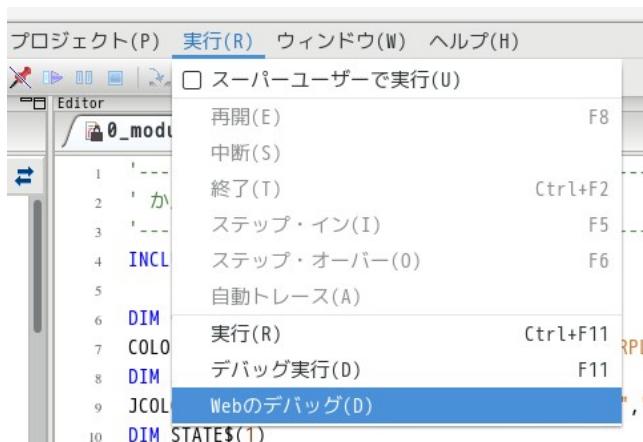
WebアプリをIDEから実行せずに、chromium/Firefox等に直接URLを指定してデバッグしたい場合、URLを入力する前に以下の操作が必要となります。

	<p>WebブラウザにURLを直接指定して実行する場合、WebアプリはWebサーバーから不定期に実行されるため、AJAN組込アプリのデバッグ手順と異なり、Webデバッグモードに切り替えてAJANプログラムの起動を待機させる必要があります。</p>
---	---

1. メニューバーもしくは、機能ボタンから以下の手順でデバッグ待機状態にします。

##### <メニューバーの場合>

Webアプリをデバッグする場合、メニューバーから「実行(R)」→「Webデバッグ(D)」を実行してWebデバッグモードに切り替えます。



メニュー項目が「Webデバッグ(D)」から「Webデバッグの停止(D)」に変化して、Webデバッグモードに切り替わったことが確認出来ます。



##### <機能ボタンの場合>

 (Webデバッグモード)ボタンをクリックすると、Webデバッグモードを開始します。ボタンが引っ込んだ状態がONとなります。

2. WebブラウザにURLを入力します。

※URLの確認は、一度IDEから実行して、WebブラウザのURL表示欄を確認して下さい。

3. 必要に応じて、ブレークポイントによるプログラムの停止や、ウォッチ機能で変数の確認等を行って下さい。
4. デバッグを終了します。

##### <メニューバーの場合>

メニューバーから「実行(R)」→「Webデバッグの停止(D)」を実行します。

メニュー項目が「Webデバッグの停止(D)」から「Webデバッグ(D)」に変化して、Webデバッグモードが終了した事が確認出来ます。

##### <機能ボタンの場合>

 (Webデバッグモード)ボタンを再度クリックすると、Webデバッグモードを終了します。

## 4.12 各種設定

メニューバーから、「ウインドウ(W)」 → 「設定(P)」を選択すると、各種設定を行うダイアログが開きます。

分類	設定項目	説明
全般	起動時にワークスペースの選択をスキップする	有効にすると、IDE起動時にワークスペースを選択するダイアログを表示しません(デフォルト:有効)
外観	色とフォント	ウインドウの色やフォントをカスタマイズします。 『4.12.1 ウインドウの色とフォント』を参照して下さい。
エディタ	タブ幅	エディタのTABキーを押した際の文字幅を指定します。(デフォルト:8)
	タブを空白に変換	有効にすると、エディタのTABキーを押した際にタブ文字を挿入する動作を、空白を挿入する動作に変更します。(デフォルト:無効)
	自動インデントを有効にする	有効にすると、エディタでFOR、SUB等を入力してEnterキーを押した際に、自動的にインデントを付加します。(デフォルト:有効)
	自動補完を有効にする	有効にすると、エディタでキー入力した際に、対応するAJANコマンド候補が自動的に表示されます。(デフォルト:有効)
	インデントガイド線を有効にする	有効にすると、エディタでタブや空白でインデントを付加した箇所に縦線が表示されます。(デフォルト:有効)
	テキストを折り返す	行がエディタの右端を超える場合に、自動的に折り返して表示されます。(デフォルト:無効)
	自動回復情報の保存	IDE不正終了時にエディタで編集中のファイルを復元する情報を保存する間隔を分単位で指定します。(デフォルト:3分)
Webブラウザ	デフォルトブラウザを指定する。	デフォルトはchromiumで、Firefoxに変更できます。前述の2種類のブラウザ以外は対応していません。
デバッガ	デバッグ開始時にレイアウトを変更する	有効にすると、デバッグ実行した際に、レイアウトが自動的に「デバッガ」に切り替わります。(デフォルト:有効)
	停止時に手前に表示	デバッガ時にブレークポイントで停止する等でトレースバーが表示される際に、アプリケーションやWebブラウザの手前にIDEが表示されます。
	実行中のF1ヘルプを無効にする	プログラム実行中にF1キーによるヘルプが開かないようにします。
エクスプローラ	ファイル比較ツール選択	エクスプローラウインドウで「保存履歴と比較...」を実行した際に起動されるファイル比較ツールを指定できます。 デフォルトはmeldが指定されており、引数に2つのファイルのパスを指定して実行するツールであれば、変更することができます。

#### 4.12.1 ウィンドウの色とフォント

各ウィンドウの色とフォントを自由に変更できます。

色は、色相、彩度、明度、赤緑青の割合、フォントは、ファミリ、スタイル、フォントの設定可能です。

項目	ウィンドウ				
	エディタ	コンソール	デバッグ	ウォッチ	エクスプローラ
テキストフォント	○	○	○	○	○
行番号の前景色	○	-	-	-	-
現在行ハイライト背景色	○	-	-	-	-
選択範囲 前景色	○	-	-	-	-
選択範囲 背景色	○	-	-	-	-
検索範囲 背景色	○	-	-	-	-
一致ハイライト背景色	○	-	-	-	-
前景色	-	○	○	-	○
背景色	○	○	○	-	○

○:変更可能 -:変更不可

#### ■手順

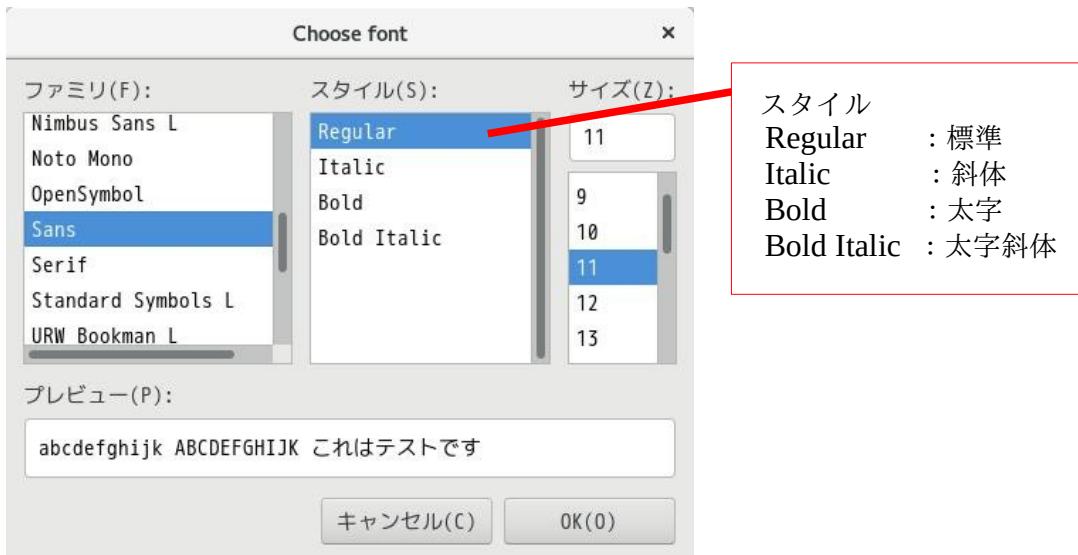
1. メニューバーから、「ウィンドウ(W)」 → 「設定(P)」を選択すると以下のダイアログが開きます。



各ウィンドウの左側の「▶」をクリックすると「▼」になり変更できる項目が表示されます。



- 2.「項目」のテキストフォントを選択して、「編集」ボタンをクリックすると以下の画面を表示します。  
ファミリ(字体)、スタイル(様式)、サイズを設定して、「OK」ボタンをクリックすれば設定できます。



- 3.「項目」の景色を選択して、「編集」ボタンをクリックすると、以下の画面を表示します。  
色相、彩度、明度、色合を設定して、「OK」をクリックすれば設定できます。



- 4.上記を設定すると、以下の画面に戻ります。  
「適用」を押すと設定値に変更されます。



## 第5章 PageGenerator

PageGenerator は、AJAN を使って Web ページを、視覚的かつ直感的な操作で作成できるツールです。

- Web ページの作成には、HTML、CSS の知識は必要ありません
- 主にマウス操作による直感的な操作で Web ページを作成できます
- グラフ、テーブル(表)、画像、ボタンなど、様々な部品を使うことができます
- AJAN プログラムを利用した、動的な Web ページを作成できます
- Web ページ内では、豊富な AJAN コマンド (\*1) を利用できます
- イベント処理を簡単に作成できるイベントアクションエディタを内蔵しています
- 他のコンピュータ上でも Web ページの表示や操作ができます

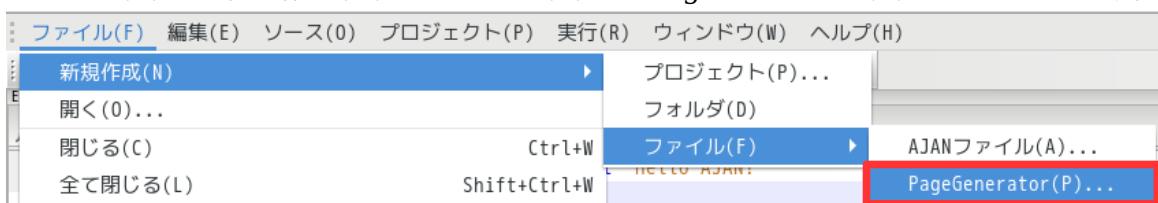
\*1: AJAN では、一般的な Web ページでは高度な実現手段を必要とする機能を簡単に実現できるコマンドが、多数提供されています。(ファイル操作、スレッド処理、シェル制御、エッジサーバ、ネットワーク通信(TCP/IP, UDP), 数学・統計演算, IO 制御ほか)

より詳しい説明に付きましては、別冊『PageGenerator 使用方法』をご参照下さい。

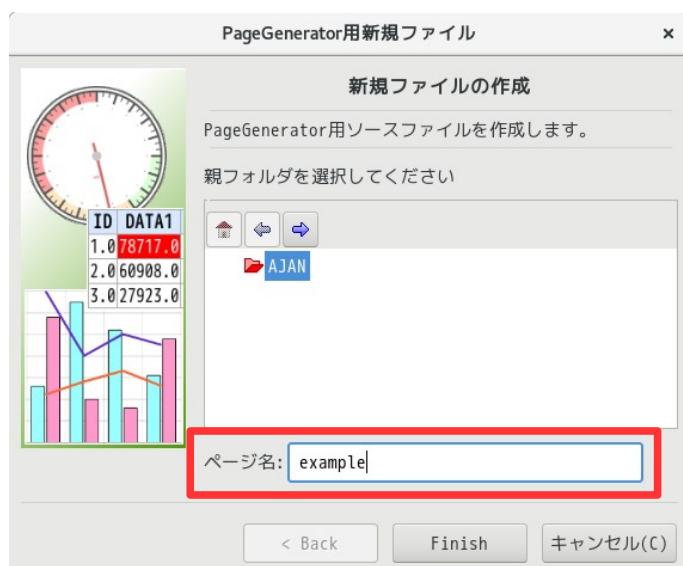
### 5.1 新規プログラムの作成

新たにプログラムを作成する場合はPageGenerator用のファイルを新規に作成します。

- 1) エクスプローラーウィンドウでファイルを作成するプロジェクトをクリックします。  
ここでは「AJAN」をクリックします。
- 2) 「ファイル(F)」→「新規作成(N)」→「ファイル(F)」→「PageGenerator(P)...」をクリックします。



- 3) PageGenerator用新規ファイル作成ダイアログが開きますので、ページ名に名前を入力して、Finish ボタンをクリックして下さい。



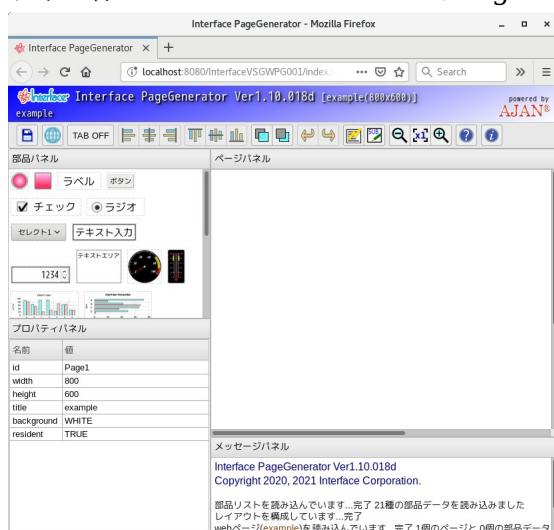
	<ul style="list-style-type: none"> <li>ページ名に全角文字および空白文字は使えません。ページ名はファイル名や実行時のURLの一部として使われます。</li> <li>ページ名は後から変更することは出来ません。</li> </ul>
---	--

- 4) エクスプローラに作成したページ名が表示されれば作成完了です。



## 5.2 PageGeneratorの起動

- エクスプローラウィンドウでPageGeneratorファイルをダブルクリックします。  
ここでは、先程作成した「example」をクリックします。
- 以下の様にWebブラウザが立ち上がりPageGeneratorが起動します。



※PageGeneratorの操作方法に付きましては、別冊『PageGenerator 使用方法』をご参照下さい。

## 5.3 サブルーチンの編集

イベントアクションエディタで作成したアクションで使用するサブルーチンやグローバル変数を作成するにはソースファイルを開いて編集を行います。

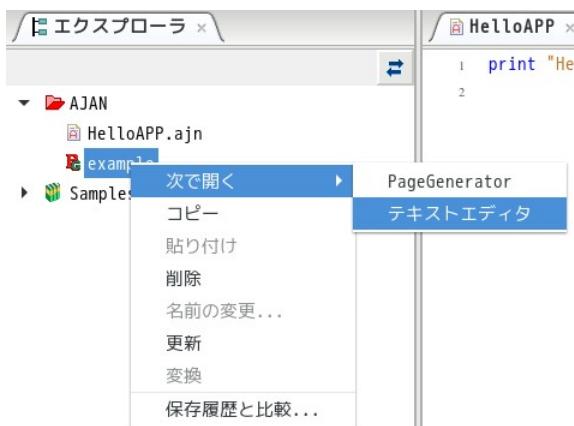
ソースファイルを開く方法として、2種類の方法を、ご紹介します。

<PageGeneratorから開く>

- 1)  (サブルーチン編集)ボタンをクリックします。
- 2) IDEに切り替わって、IDE上でソースファイルが開きます。

<IDEから開く>

- 1) エクスプローラーウィンドウのPageGeneratorファイル上で右クリックするとポップアップメニューが開きますので、「次で開く」→「テキストエディタ」をクリックしてください。



- 2) エディタウィンドウでソースファイルが開きます。

## 5.4 デバッグを行う

PageGeneratorで作成したアプリケーションも、IDEのデバッグ機能を使用してデバッグが行えます。

IDEからデバッグ実行を行うには、IDEでソースファイルを開く必要があります。

- 1) 前述の「5.3 サブルーチンの編集」の手順で、ソースファイルを開いて下さい。
- 2)  (デバッグ実行)ボタンをクリックしてください。



デバッグ機能の詳細は、「4.11 プログラムのデバッグ」を参照して下さい。

## 第6章 マルチスレッドプログラム

AJANでは、簡単に複数のプログラムやサブルーチンを同時に並行して動かすことができます。

この章では、サンプルプログラムとして用意されているTHREAD.AJNを題材にマルチスレッドプログラムの動作と、デバッグ操作の概要を紹介します。

マルチスレッドの詳細は、AJANコマンドリファレンス「標準コマンド」を参照してください。

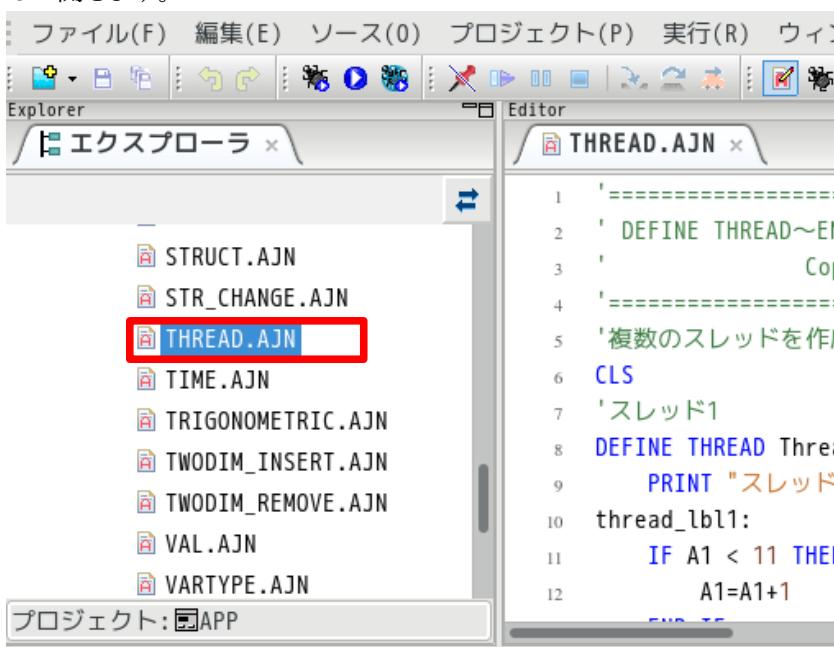
### 6.1 動かしてみよう

#### 6.1.1 動作確認

THREAD.AJNは、スレッド1,スレッド2の実行結果をコンソールウィンドウに表示します。

実際に動かしてみましょう。

1. エクスプローラーウィンドウから Samples→BASEフォルダ下にあるTHREAD.AJNをダブルクリックして開きます。



2. 「実行」ボタンをクリックしてください。



3. スレッド1,スレッド2の実行結果がコンソールウィンドウに表示されます。

```

[コンソール]
ATTACH THREAD
スレッド番号1
スレッド番号2
DETACH THREAD
A1=11 A2=21
Ok
    
```

メインスレッドは、非同期に動作しているスレッド1とスレッド2が更新する変数A1とA2のカウントアップを監視して、一定の値を超えたたら、A1とA2の値を表示して終了します。

## 6.2 マルチスレッドのデバッグ

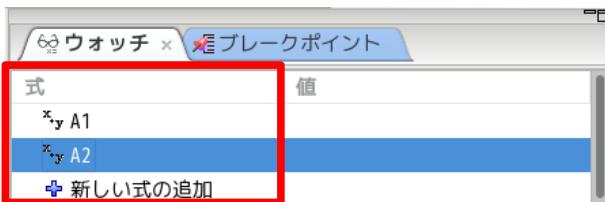
『6.1 動かしてみよう』に引き続き、THREAD.AJNを題材にデバッグの操作手順を見ていきましょう。

### 6.2.1 デバッグ開始

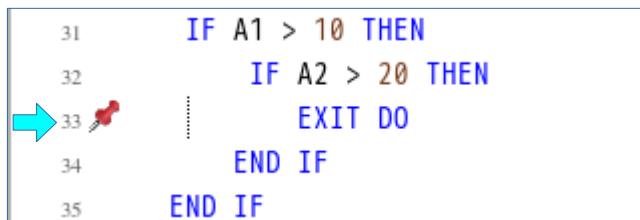
下記のボタンをクリックして、エディタ画面からデバッグ画面に切り替えます。



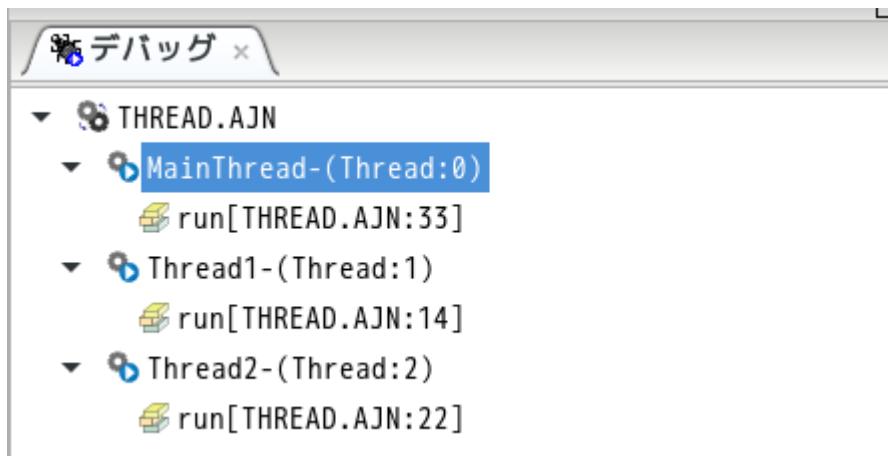
ウォッチウィンドウに監視する変数(A1,A2)を入力してください。



次に、メインスレッドの行番号33をダブルクリックして、ブレークポイントの設定を行ってください。



⑤(デバッグ実行)ボタンをクリックして、メインスレッドの行番号33で、プログラムが停止したら、デバッグウィンドウを広げてください。



デバッグウィンドウには、実行中のスレッド一覧が表示されます。

デバッグ実行が停止されると以下のように停止したスレッドの行番号が表示されます。

メインスレッドのスレッド番号は0固定で、スレッド番号は1から15までの任意の番号(※)です。

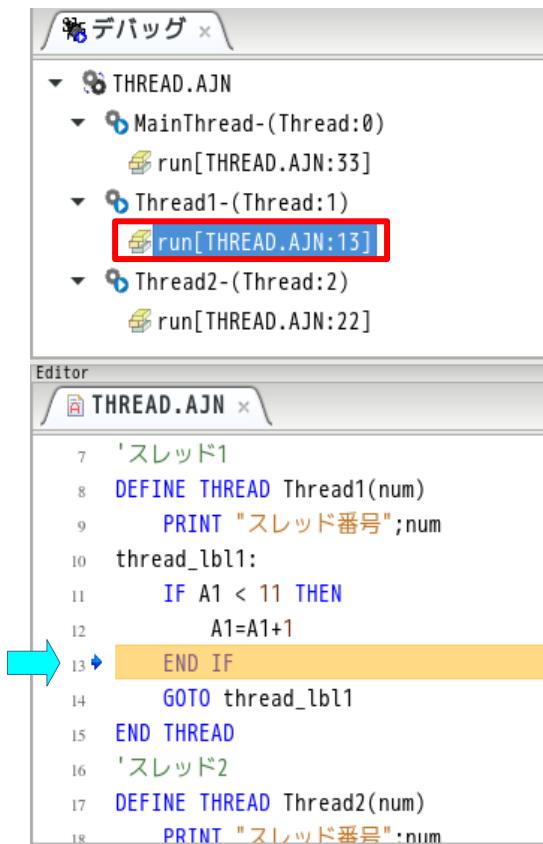
※「ATTACH THREAD」コマンドでワーカースレッドを生成する時に指定したスレッド番号

ワーカースレッド内にブレークポイントを設定し、変数の内容をウォッチするなど、マルチスレッドのデバッグも、通常のプログラムと同様に行うことが出来ます。

但し、マルチスレッド内のローカル変数を表示するには、デバッグウィンドウで、そのスレッドを選択する必要があります。

次にデバッグウィンド内のThread1をクリックすると、プログラムが停止した時に実行されていた行番号13にトレースバーが表示されます。

※ プログラムが停止した時に実行されていたThread1の実行行は不定です。



マルチスレッド時のトレースバーは、デバッグウィンドウで選択したスレッドのものが表示されます。スレッド内のローカル変数をウォッチしている際も同じ名前の場合、デバッグウィンドウで選択したスレッドのローカル変数を優先表示します。  
他のスレッドが選択されている場合は、グローバル変数として扱いますので必ずデバッグしたいスレッドをデバッグウィンドウで選択してください。

(再開)ボタンをクリックすると、プログラムが最後まで実行されて終了します。プログラムが終了した後に、 (エディタ画面)ボタンをクリックしてエディタレイアウトに切り替えてみましょう。

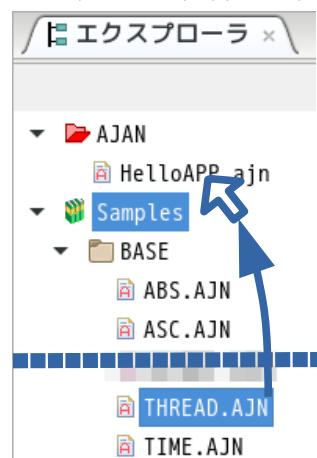
## 6.2.2 デバッグ操作

この節では、自動トレース機能を使って、ウォッチウィンドウ内の変数と3つのスレッドが停止した行番号を確認して行きます。

### <事前準備>

ここではサンプルプログラムを修正しますが、サンプルプログラムのプロジェクトは書込み不可に設定されていますので、THRAD.AJNを編集可能領域にコピーします。SamplesプロジェクトのBASE/THRAD.AJNをマウスでドラッグして、編集可能領域のAJANプロジェクト上にドロップします。

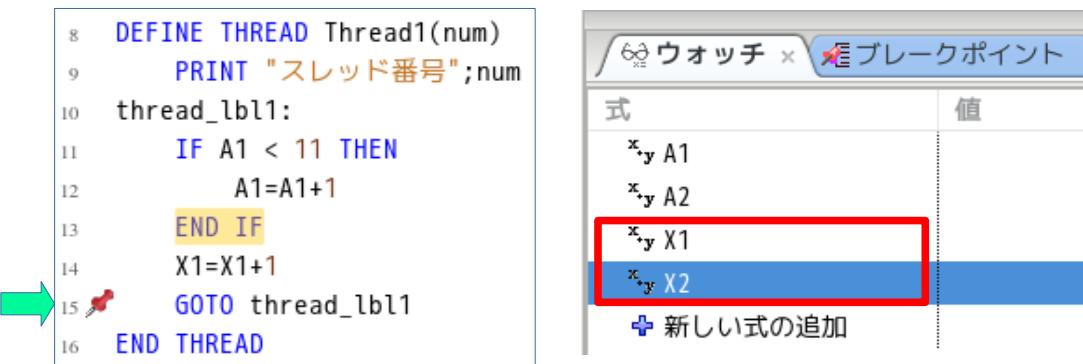
「ファイルをコピーしますか?」と表示されるので、「はい」をクリックします。



では、コピーしたTHREAD.AJNにエディタウィンドウ上で、新たな変数X1とX2の処理を追加します。

```
'=====
'===== DEFINE THREAD~END THREAD,ATTACH THREAD,DETACH THREAD サンプルプロ
グラム
'
Copyright 2018 Interface Corporation
'===== =====
'複数のスレッドを作成し、それぞれが別の変数に対して並列に演算します
CLS
'スレッド1
DEFINE THREAD Thread1(num)
  PRINT "スレッド番号";num
thread_lbl1:
  IF A1 < 11 THEN
    A1=A1+1
  END IF
  X1=X1+1
  GOTO thread_lbl1
END THREAD
'スレッド2
DEFINE THREAD Thread2(num)
  PRINT "スレッド番号";num
thread_lbl2:
  IF A2 < 21 THEN
    A2=A2+1
  END IF
  X2=X2+1
  GOTO thread_lbl2
END THREAD
A1=0
A2=0
X1=0
X2=0
PRINT "ATTACH THREAD"
ATTACH THREAD 1, Thread1      'スレッド1を作成してTHREAD1を実行
ATTACH THREAD 2, Thread2      'スレッド2を作成してTHREAD2を実行
DO WHILE 1
  IF A1 > 10 THEN
    IF A2 > 20 THEN
      EXIT DO
    END IF
  END IF
LOOP
PRINT "DETACH THREAD"
DETACH THREAD 1                'スレッド1を終了
DETACH THREAD 2                'スレッド2を終了
PRINT "A1="; A1; " A2="; A2
END
```

 (デバッグ画面)ボタンをクリックしてデバッグレイアウトに切り替え、15行にブレークポイントの設定とウォッッチ変数の追加を行います。



The screenshot shows the AJAN development environment. On the left is the code editor with the following pseudocode:

```

8  DEFINE THREAD Thread1(num)
9    PRINT "スレッド番号";num
10   thread_lbl1:
11     IF A1 < 11 THEN
12       A1=A1+1
13     END IF
14     X1=X1+1
15     GOTO thread_lbl1
16 END THREAD

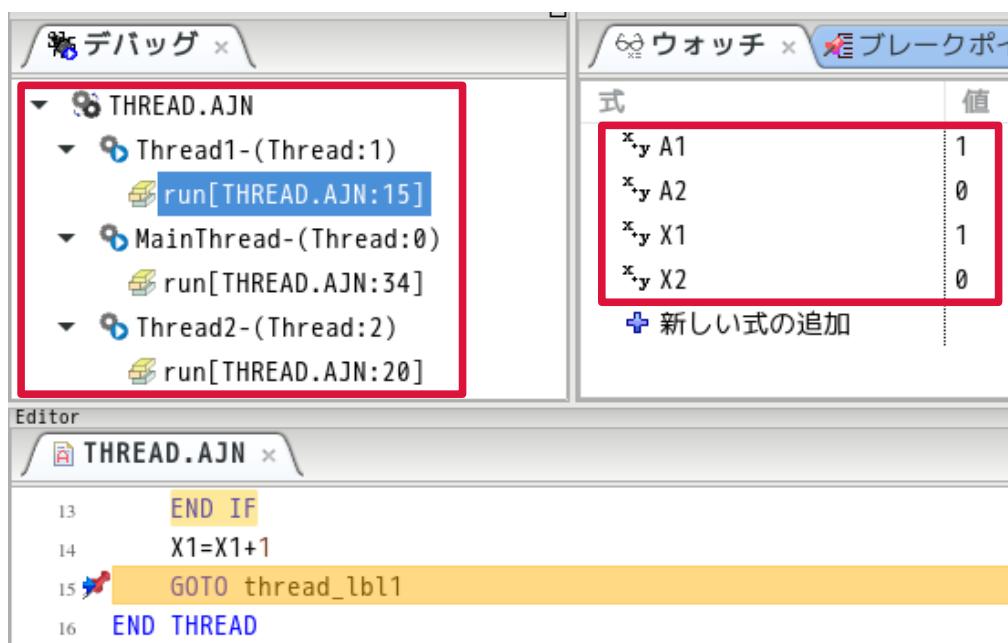
```

A green arrow points from the code editor to the 'Debug' button in the toolbar.

On the right is the 'Watch' window, which lists variables A1, A2, X1, and X2. X1 and X2 are highlighted with red boxes, indicating they are being watched.

「デバッグ実行」ボタンをクリックすると、保存するか聞いてきます。保存ボタンをクリックして下さい。

次に、「自動トレース」ボタンをクリックすると、設定されたブレークポイントでプログラムが停止します。何度か「自動トレース」ボタンをクリックしてウォッッチしている変数の値と、プログラムが停止した時に実行されていた行番号の変化を確認してみてください。



The screenshot shows the AJAN development environment with the 'Debug' view active. The left pane shows the call stack with Thread1 at the top, and the right pane shows the 'Watch' window with variable values:

式	値
x,y A1	1
x,y A2	0
x,y X1	1
x,y X2	0

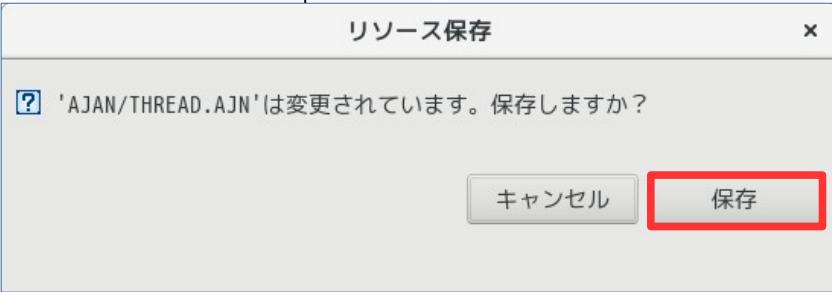
The 'Editor' view at the bottom shows the AJAN code with line 15 highlighted, indicating it is the current execution point.

デバッグ操作の概要説明は以上となります。

サンプルプログラムをベースに色々と手を加えながら、AJANを試してみてください。

## 第7章 FAQ

### 7.1 FAQ

症 状	処 理
エディタウィンドウ等のウィンドウのタイトルバーをドラッグしてアプリケーションから分離させた場合の戻し方。	分離したウィンドウのタイトルバーをダブルクリックすると、元の場所に戻ります。
ウィンドウのレイアウトが元に戻せなくなった。	下記の手順でレイアウトをリセットして下さい。 メニューバーから、「ウィンドウ(W)」→「レイアウト(L)」→「レイアウトのリセット(R)...」。
	
AJAN 統合開発環境を複数起動することが出来る？	起動できます。 但し、同じワークスペースは開けません。
プロジェクトをファイル単位に取り込みたい。	1)取り込みたいファイルやフォルダを、プロジェクト内の任意のフォルダに直接コピーする。 2)エクスプローラウィンドウ上で右クリックして「更新」を実行する。
コンテンツアシストが表示されない。	コマンド候補が無いと表示されません。 例えば、エディタ上にHOGEと入力して、コンテンツアシストを起動した場合、HOGEというワードと先頭一致するAJANコマンドが存在しないため、何も反応しません。
プログラムを印刷したい。	gedit等でソースファイルを開いて印刷してください。 geditでソースファイルを開く簡単な方法は、エクスプローラウィンドウ上のファイルをgeditにドラッグ＆ドロップすることで開きます。
デバッグの自動トレース中にON KEY CALLが反応しない。	デバッグ停止中は、ON KEY CALLは無効となります。 停止中のファンクションキーは、デバッガのキーアサインが有効となります。 自動トレースは、ステップ実行を繰り返すモードですので、実行と停止を繰り返しています。停止期間中の方が長いため、ファンクションキーの応答性は著しく落ちるため、自動トレース中のON KEY CALLの動作は保証致しません。自動トレースではなく、再開ボタンで実行させてください。

## 7.2 困ったときのチェックポイント

症 状	処 理
突然OSが再起動した。	何かの要因でシステムとして続行不能な状態に陥ったため、ウォッチドッグタイマが働き、強制的に再起動したと考えられます。何度も再現する場合、弊社カスタマーサポートセンタにご連絡ください。
ファイルが保存できない。	<p>システムの空き容量が不足している可能性があります。</p> <p>既存のファイルを削除し、空き容量を確保してください。USBメモリにファイルが保存できない場合は、以下の点を確認してください。</p> <ul style="list-style-type: none"> <li>・USBメモリのライトプロテクト機能(書き込み保護)がONに設定されていないか。</li> <li>・USBメモリの空き容量が不足していないか。</li> </ul> <p>また、書き込み権限がない場所に保存している可能性もあります。別の場所に保存してください。</p>
Linuxでファイルが保存できない。	<p>ファイルシステム全体がリードオンリーモードになっている可能性があります。コンピュータを書き込み可能モードで再起動してください。</p> <p>なお、書き込み可能モードにした場合、システムは保護されません(電源のブチ切りは行えません)。</p> <p>また、書き込み権限がない場所に保存している可能性もあります。別の場所に保存してください。</p>
サンプルプログラムが保存できない。	サンプルプログラムのディレクトリは他のユーザが使用することも想定し参照専用になっています。別の場所に保存してください。
ファイルが開けない。	<p>AJANプログラムファイルが開けない要因としては、いくつか考えられます。</p> <ul style="list-style-type: none"> <li>・ファイルパスの指定が間違っている。 →フォルダの指定、拡張子の指定、大文字小文字の指定が間違っていないか確認してください。</li> <li>・ファイルが壊れている。 →ファイル保存中システム電源断等により、ファイル自体が壊れている可能性があります。 ファイルのバックアップを積極的に行ってください。</li> <li>・ファイルのエンコードがUTF-8では無い。 →AJANプログラムを外部で作成して、USBメモリ経由で持ち込む場合、作成するAJANプログラムはUTF-8形式(BOM不要)でテキストを作成してください。</li> </ul>

症 状	処 理
電源を入れた後、ディスプレイを接続したら画面が出ない。	<p>OS起動後のディスプレイ接続はサポートしていません。 システムは起動中に接続されているディスプレイの情報を得て、最適な状態で表示しようとします。 ディスプレイを接続せずに動かすと、情報が得られないため、画面を表示することができません。 また、電気的にもシステム稼動中に抜き差しすることは好ましくありません。</p>
リードオンリーモード無効で運用していたら、空きディスク容量が減っていた。	<p>リードオンリーモード無効で使用される場合、Linuxが吐き出す各種ログが、ディスク上に追記されるようになります。放っておくとログの追記により、空きディスク容量が少なくなりますので、以下のように対策してください。</p> <ul style="list-style-type: none"> <li>・不要なログ出力を控える(ログを出力するサービスを止める等)</li> <li>・不要なログを削除する</li> </ul> <p>以下のログファイルには注意してください。</p> <ul style="list-style-type: none"> <li>・<code>/home/ユーザアカウント名/.xsession-errors</code></li> <li>・<code>/var/log</code>フォルダ下の各種ログファイル</li> </ul>
外部のコンピュータからPageGeneratorで作成したWebページが参照できない。	<p>①ローカルのコンピュータ内でWebページが正常に動作することを確認してください。</p> <p>②以下の手順で外部公開の設定が行われているか確認してください。</p> <ol style="list-style-type: none"> <li>(1)IDEで対象のWebページが含まれているプロジェクトを右クリックし、「設定」を選択する。</li> <li>(2)「外部公開設定」の「外部公開」にチェックがあれば外部公開設定されています。(デフォルトは「ローカル」に設定されており、外部からは接続できません。)</li> </ol> <p>③クライアントのブラウザから、以下のURLを入力して、正常にアクセスできることを確認してください。</p> <p>[ブラウザに入力するURL例※1 ※2 ※3]  <a href="http://192.168.100.100/AJAN/Untitled.ajn">http://192.168.100.100/AJAN/Untitled.ajn</a></p> <p>※1 AJANのインストールされているPCのIPアドレスが「192.168.100.100」の例です。</p> <p>※2 Webサーバの外部公開設定で、URLに「AJAN」と設定している場合の例です。詳しくは「4.4.1 e)プロジェクトを外部公開する」をご参照ください。</p> <p>※3 PageGeneratorファイルを作成する際に名前をUntitledと入力した場合です。</p>

## 7.3 Tips

### a) Windowsのファイルを利用する場合

Windowsで作成したテキストファイル(文字コードSJIS)をAJANで処理する際は、Linux上で事前に以下のコマンドを実行し、文字コードをSJISからUTF-8へ変換して利用してください。

※ sjis.txtというWindowsで作成したテキストファイルを、AJANで利用するためにutf8.txtに変換する場合

```
$ nkf -SwLu sjis.txt > utf8.txt
```

上のコマンドの「-SwLu」は、文字列変換方法の指定例です。

「S」は入力ファイルがShift\_JISである事、「w」は出力ファイルをUTF-8で行う事、「Lu」は改行コードをLinuxの形式(CHRB\$(10))に変換する事、を意味します。

### b) 大文字と小文字の取り扱いについて

AJANは大文字と小文字を識別しません。内部的に全て大文字として扱われます。

ただし、ファイルとディレクトリ名だけは大文字と小文字で識別されます。ファイルをOPENしたり、プログラム内から別のファイルを呼び出したりする際は、大文字と小文字を使い分けてください。

### c) 変数とメモリの関係

文字列や配列等の各変数は、コマンドリファレンス「標準コマンド編」に記載されている制限事項を除いて、基本的にOSが確保できるメモリ量に依存します。

目安として、メモリ1GB搭載製品の場合、一つの文字列変数を100MB以上確保しようとすると、かなりシステムに負荷がかかるることを確認しています。同様に、整数の配列変数を、100MB以上の添え字で確保しようとすると、やはりシステムに相当の負荷が掛かることを確認しています。

#### d) 実数型を扱う際の注意点

実数型は、コンピュータ内部では数値が2進数の小数として格納されています。

そのため、整数型と違って全ての値を正確に表現できず、近似値で表現されます。

そのため、計算するうちに誤差が生じてしまう場合があります。

例えば、以下のような式の結果は、奇妙に感じます。

```
A# = 1.0 - 0.9
PRINT A# = 0.1
FALSE ' 上のPRINT文の実行結果(0=False)
```

数学上は、 $1.0 - 0.9$  の値は 0.1 ですが、計算結果は等値ではない(FALSE)と言われます。

これはコンピュータ内部の実数演算の誤差によるものです。

一般的には、実数の比較は対象の2つの差分を取り、これが誤差の範囲内か判断するようにします。

```
EPS# = 0.00001 ' 精度
A# = 1.0 - 0.9
PRINT ABS(A# - 0.1) < (A# * EPS#)
TRUE ' 上のPRINT文の実行結果(1=True)
```

実数型は、表現可能な範囲に制限があります。

例えば、単精度実数および倍精度実数の正の最大値は、以下により与える事ができます。

(最大値を設定するには、値の丸めを勘案して、桁数を多めに指定する必要があります)

```
' 単精度実数の正の最大値
FLT_MAX! = 3.40282346638528859812E+38
' 倍精度実数の正の最大値
DBL_MAX# = 1.79769313486231570815E+308
```

## e) 配列変数を扱う際の注意点

配列変数は、多くのデータを変数に代入したり、参照したりといった場合に使います。

例えば、10個の変数を用意しようとする場合、配列変数を用いないと以下のように記述する事になるでしょう。

```
A1=0: A2=0: A3=0: A4=0: A5=0: A6=0: A7=0: A8=0: A9=0: A10=0
A1 = 1
A10 = 10
```

配列変数を用いると、以下のように簡素に記述できます。

```
DIM A(9)      ' カッコ内の添え字は、0-9までの10個の入れ物ができます
A(0) = 1
A(9) = 10
```

配列変数内の一要素にアクセスする場合、添え字を用います。添え字には変数を用いる事ができるので、以下のような記述が可能です。

```
DIM A(9)      ' カッコ内の添え字は、0-9までの10個の入れ物ができます
...
FOR I=0 TO 9
    PRINT "A("; I; ")="; A(I)      ' Aの全ての要素を表示します
NEXT I
```

多次元の配列変数も使用できます。

```
DIM A(2, 3)    ' 3×4=12個の2次元配列ができます
...
FOR Y=0 TO 3
    FOR X=0 TO 2
        PRINT A(X, Y)    ' 一要素を表示します
    NEXT X
NEXT Y
```

## f) 指定範囲の配列使用(範囲配列)指定

配列変数へ値をセットしている時、1要素ずつセットを繰り返すのではなく一括でセットしたい場面が出るでしょう。

AJANでは、一括でセットする為の特殊な構文をサポートしています。これを、私たちは範囲配列と呼んでいます。



構造体の配列では、範囲配列を使用することはできません。

通常であれば、5つの要素を持つ配列変数への値のセットは、以下のように記述します。

```
DIM A(4)
A(0) = 1 : A(1) = 2 : A(2) = 3 : A(3) = 4 : A(4) = 5
```

範囲配列記述を用いると、以下のように記述できます。

```
DIM A(4)
A = [ 1; 2; 3; 4; 5 ]' セットする値の要素数と、セットされる側の要素数は一致が必要です
A = [ 1, 2, 3, 4, 5 ]' 値のセットの場合、要素の区切りは「;」と「,」の両方使えます
```

上の用法に示すように、配列変数に一気にセットする為に、値を「;(セミコロン)」ないしは「,(カンマ)」を使って区切った値を、角括弧で囲って列挙できます。

文字列値や実数値のセットも可能です。

```
DIM A$(2)
A$ = [ "hello"; "AJAN"; "World" ]

DIM B#(2)
B# = [ 1.23; 4.56; 7.89 ]
```

配列変数の一部を指定して、セットも可能です。

この場合、セットしたい添え字を「;(セミコロン)」で区切れます。

値の列挙とは異なり、一つの次元に対して添え字の列挙の為に「,(カンマ)」を用いることはできません。(別の次元に移る意味になります)

```
DIM A(3)
A(0;1) = [ 1; 2 ]' A(0)=1 : A(1)=2 と同じ効果です

A(0,1) = [ 1; 2 ]' 添え字の列挙に「,」は使えません。この用法は二次元記述となります
```

添え字の指定が連続する場合、「<始まりの添え字> TO <終わりの添え字>」という記述で、指定可能です。(昇順で指定してください)

```
DIM A(4)
A(1 TO 3) = [ 1; 2; 3 ]' A(1)=1 : A(2)=2 : A(3)=3 と同じ効果です
```

セットする値が单精度整数の場合のみ、「TO」を使った指定も可能です。

```
DIM A(4)
```

```
A(1 TO 3) = [ 4 TO 6 ] , A(1)=4 : A(2)=5 : A(3)=6 と同じ効果です
```

添え字での「,(カンマ)」指定は、多次元配列でのセットに用います。

```
DIM A(2, 3)
```

```
A(0;2, 1;3) = [ 1;2;3;4 ] , A(0, 1)=1 : A(0, 3)=2 : A(2, 1)=3 : A(2, 3)=4 と同じ効果です
```

```
A(0;2, 1;3) = [ [1;2], [3;4] ] , 角括弧を駆使して、人間にわかり易い書き方ができます
```

値の列挙は1次元配列状に記述し、多次元配列の添え字指定順に格納されるイメージとなります。

### g) コマンドおよび関数の引数渡し時の範囲配列指定

一部のコマンドおよび関数では、引数(パラメータ)に値を渡す際、範囲配列を使った値列挙が可能です。

```
SUB TEST(A AS LIST, B)
```

```
PRINT "引数Aの要素数", LDIM(A)
```

```
PRINT "引数Bの値", B
```

```
END SUB
```

```
CALL TEST([1;2;3;4], 123) , 配列値を渡すには、角括弧で値を列挙する必要があります
```

特定の引数に配列値を渡すために、角括弧が必須となります。また、渡せる配列値は1次元配列のみです。

この機能は、SUB、FUNCTIONなどのユーザー定義関数でも使用できます。

### h) AJANの情報を表示する

現在使用中のパッケージのバージョンを確認したい場合、情報ダイアログから参照できます。

メニューの「ヘルプ(H)」 → 「AJANについて(A)」を選択してください。

下記のバージョン情報ダイアログが立ち上がります。

ここには、AJANのバージョンが記載されています。



## i) 有効桁を超えた実数演算時の注意

実数には有効桁が存在し、有効桁を超えた演算を行った時、結果が変化しない事があります。

例えば、以下のような大きな桁と小さな桁同士で、加算すると、結果が変化しません。

```
' ↓ある程度の(有効)桁数以内での演算だと、加算できているが…
? 1000000! + 1.0!      ' 1000001 と表示
' ↓ある程度の(有効)桁数以上での演算だと、加算できていない
? 10000000! + 1.0!      ' 1E+07 と表示
```

この現象は、一般的に情報落ちと呼ばれる、実数演算で起こる特有の現象です。

実数演算は、有効桁数の範囲同士で演算しないと、小さな桁の情報が無くなる場合があります。

(詳しくは、IEEE754 浮動小数点数の規格およびコンピュータ数学に関する書籍を参照ください)

対策としては、基本的に大きな値の範囲を扱える数値型を用います。

左記の例では、単精度実数で演算を行っているので、より大きな型である倍精度実数か、小数点数を扱わないなら倍精度整数などを使います。

更に精度を求める場合は、多桁での演算を扱えるプログラミング言語やライブラリなどの採用を検討ください。

## j) プログラムを実行するとデバイスオープン時にエラーとなる場合。

<シリアル通信の例>

```
コンソール × エラーログ
Error! &H01000025: 実行時権限が不足しています(デバイスのオープンに失敗しました(スーパーユーザ)) [/tmp/NG]
```

ファイル:demo/samples/IML/UART/COMSEND.AJN

シリアル通信は、スーパーユーザー(root権限)で実行しないと、上記のようにデバイスオープン時にエラーとなります。

この場合、スーパーユーザーモードでプログラムを実行してください。。

詳細は、『4.7.6 スーパーユーザーモードでプログラムを実行』を参照して下さい。

## 7.4 AJAN 統合開発環境を使用しないでAJANを実行する方法

AJAN 統合開発環境を使用しないで、AJANを実行させる方法を説明します。

	<p>AJANの実行ファイルとライブラリのバージョンは密接に関連しています。 AJANをアップデートする際には、作成したAJANプログラムを再コンパイルしてください。</p>
---	---

### a) 「端末」から実行させる方法

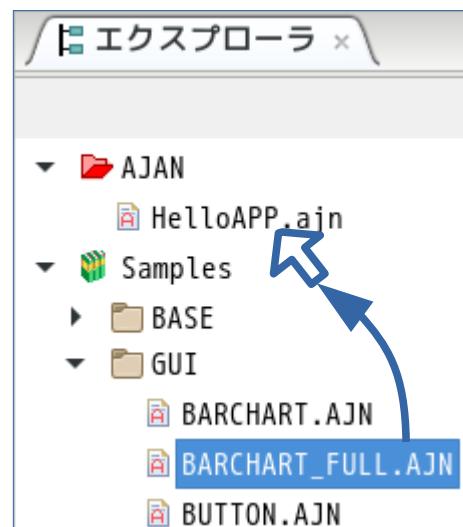
GUIサンプルプログラムBARCHART\_FULL.AJNを例にIDEを使用しないで実行する方法を説明します。

<事前準備>

サンプルプログラムが格納されたプロジェクトは書き込み不可に設定されていますので、編集可能領域にコピーするまでは、IDEで作業を行います。

SamplesプロジェクトのGUI/BARCHART\_FULL.AJNをマウスでドラッグして、編集可能領域のAJANプロジェクト上にドロップします。

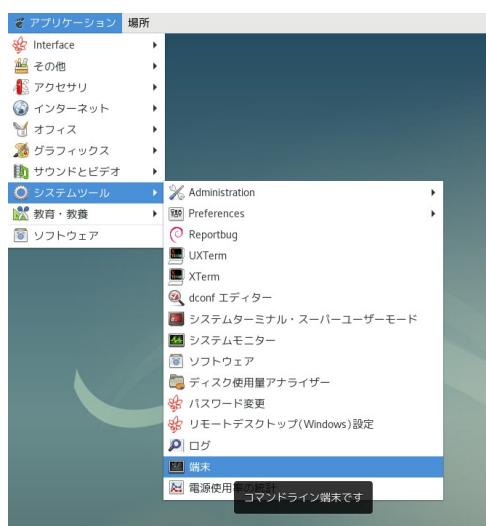
「ファイルをコピーしますか?」と表示されるので、「はい」をクリックします。



<手順>

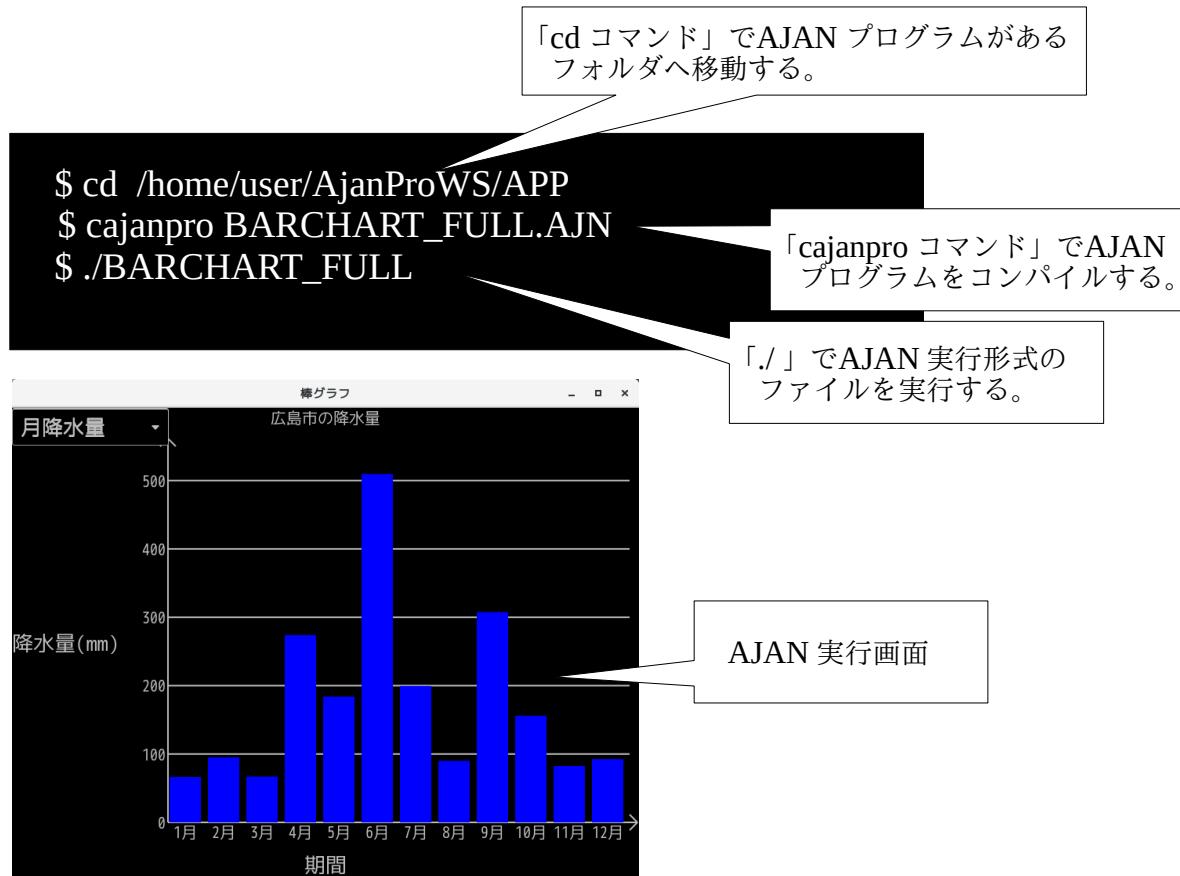
①「端末」の起動

デスクトップから左上の「アプリケーション」 → 「システムツール」 → 「端末」を起動してください。



## ②AJANプログラムのコンパイルと実行

以下のように端末に入力して、AJANプログラムをコンパイルして、実行形式のファイルを実行します。



## b) 作成したAJANプログラムを「アプリケーション」メニューに追加する方法

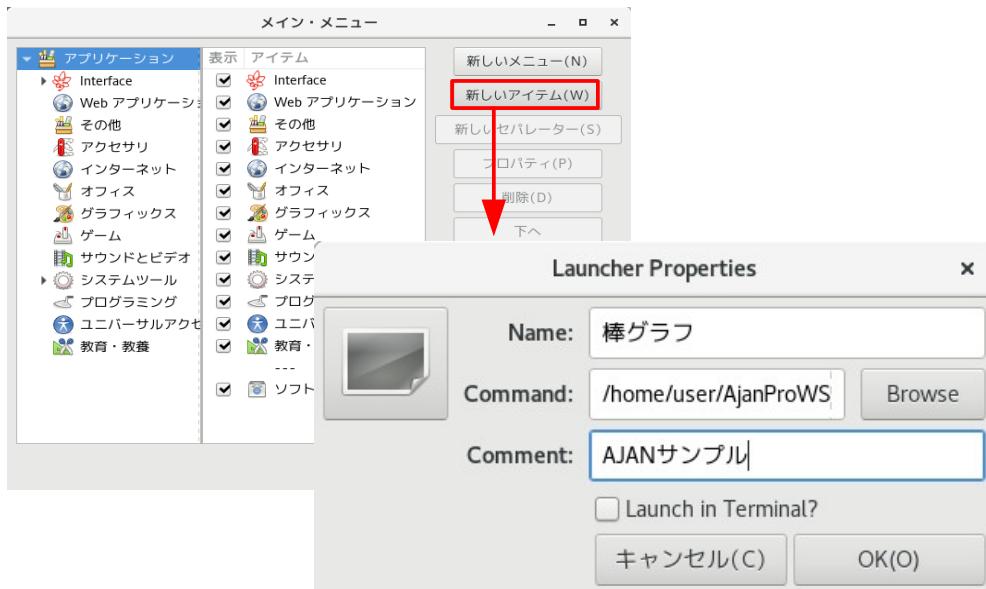
## ①「メイン・メニュー」の起動

デスクトップから左上の「アプリケーション」 → 「アクセサリ」 → 「メイン・メニュー」から、メイン・メニュー画面を表示してください。



## ②「メイン・メニュー」の設定

メイン・メニュー画面の「新しいアイテム」ボタンを押して、設定する内容を入力してください。



Name: メニューに追加する名前を入力する。

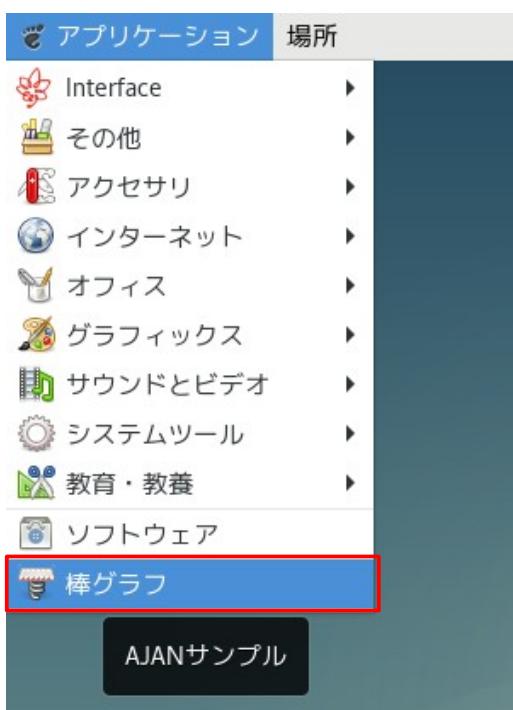
Command: 「Browse」ボタンを押してAJAN 実行ファイルを選択する。

パス : /home/user/AjanProWS/AJAN/BARCHART\_FULL

Comment: ツールチップ（カーソル等を項目に合わせた時に補足情報として表示する枠）のコメントを入力する。

## ③アプリケーションメニューからAJANプログラムを実行

アプリケーションメニューから、AJANプログラムを選択して実行できます。

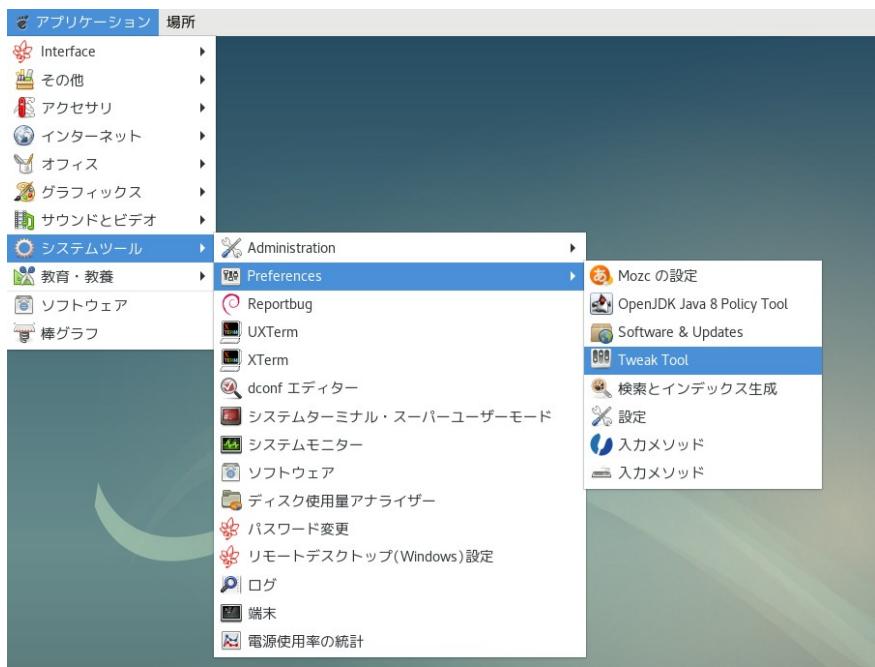


### c) 作成したAJANプログラムを自動実行させる方法

	本項を行うには、前提条件として『b) 作成したAJANプログラムを「アプリケーション」メニューに追加する方法』を先に行っておく必要があります。
---	---

#### ①自動実行設定画面の表示

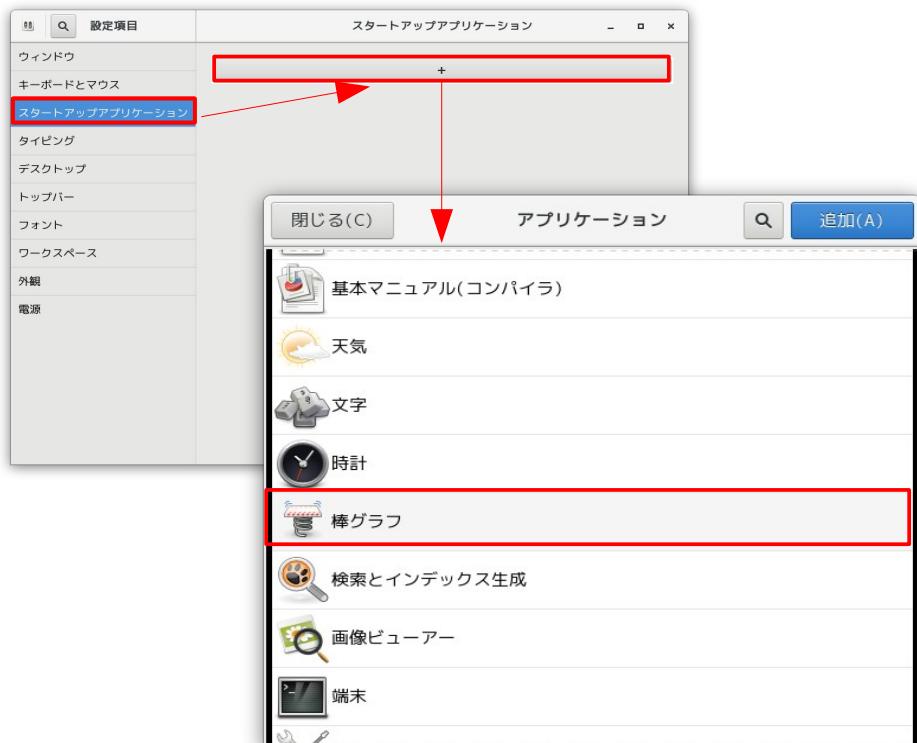
デスクトップから左上の「アプリケーション」 → 「システムツール」 → 「Preferences」 → 「Tweak Tool」を選択してください。



## ②自動実行設定画面の表示

デスクトップから左上の「アプリケーション」 → 「システムツール」 → 「Preferences」 → 「Tweak Tool」を選択してください。

スタートアップアプリケーションを選択して、「+」ボタンを押すとアプリケーションの一覧が表示されます。



追加するアプリケーション(AJANプログラム)を選択すると、スタートアップアプリケーションに追加され次回ログイン時に、自動実行を行います。



## 7.5 AJAN Webサーバの停止方法

AJANとAJAN以外のWebサーバ (Apache等)は、ネットワークポートが競合するために同時に使用することができません。AJAN以外のWebサーバを使用される場合は、以下の手順でAJANのWebサーバを無効にしてください。

※AJANのWebサーバを停止した場合は、AJANのWebアプリケーションや、本Web サーバを使用している開発ツール/エディタ等が使用できなくなります。  
AJAN組込アプリケーションへの影響はありません。

【AJAN Webサーバを無効にする手順】

- (1) デスクトップ画面の画面上部のメニューバーから「アプリケーション」→「システムツール」→「端末」をクリックします。
- (2) 端末で以下コマンドを入力してください。

Web Serverの停止

```
sudo ifweb stop
```

Web Serverの自動起動を無効化

```
sudo ifweb disable
```

※コマンド入力時にパスワードを求められた場合は、ログイン時のパスワードを入力してください。

※コマンドの実行ができなかった場合は、スーパーユーザ(管理者)でログインしてコマンドを実行してください。(スーパーユーザ(管理者)でログインする方法はOSマニュアルをご参照ください。)

【AJAN Webサーバを有効にする手順】

- (1) デスクトップ画面の画面上部のメニューバーから「アプリケーション」→「システムツール」→「端末」をクリックします。
- (2) 端末で以下コマンドを入力してください。

Web Serverの自動起動を有効化

```
sudo ifweb enable
```

Web Serverの開始

```
sudo ifweb start
```

## 第8章 重要な情報

---

### 保証の内容と制限

弊社は本ドキュメントに含まれるソースプログラムの実行が中断しないこと、またはその実行に誤りが無いことを保証していません。

本製品の品質や使用に起因する、性能に起因するいかなるリスクも使用者が負うものとします。

弊社はドキュメント内の情報の正確さに万全を期しています。万一、誤記または誤植などがあつた場合、弊社は予告無く改訂する場合があります。ドキュメントまたはドキュメント内の情報に起因するいかなる損害に対しても弊社は責任を負いません。

ドキュメント内の図や表は説明のためであり、ユーザ個別の応用事例により変化する場合があります。

### 著作権、知的所有権

弊社は本製品に含まれるおよび本製品に対する権利や知的所有権を保持しています。

本製品はコンピュータ ソフトウェア、映像/音声(例えば図、文章、写真など)を含んでいます。

### 医療機器/器具への適用における注意

弊社の製品は人命に関わるような状況下で使用される機器に用いられる事を目的として設計、製造された物では有りません。

弊社の製品は人体の検査などに使用するに適する信頼性を確保する事を意図された部品や検査機器と共に設計された物では有りません。

医療機器、治療器具などの本製品の適用により、製品の故障、ユーザ、設計者の過失などにより、損傷/損害を引き起こす場合が有ります。

### 複製の禁止

弊社の許可なく、本ドキュメントの全て、または一部に関わらず、複製、改変などを行うことはできません。

### 責任の制限

弊社は、弊社または再販売者の予見の有無にかかわらず発生したいかなる特別損害、偶発的損害、間接的な損害、重大な損害について、責任を負いません。

本製品(ハードウェア、ソフトウェア)のシステム組み込み、使用、ならびに本製品から得られる結果に関する一切のリスクについては、本製品の使用者に帰属するものとします。

本製品に含まれる不都合、あるいは本製品の供給(納期遅延)、性能もしくは使用に起因する付帯的損害もしくは間接的損害に対して、弊社に全面的に責がある場合でも、弊社はその製品に対する改良(有償サービスの利用)、代品交換までとし、製品の予防交換並びに、代金減額等、金銭面での賠償の責任は負わないものとします。

本製品は、日本国内仕様です。

### 商標/登録商標

本書に掲載されている会社名、製品名は、それぞれ各社の商標または登録商標です。

## 改訂履歴

Ver.	年 月	改 訂 内 容
0.93	2023年3月	エクスプローラの機能ボタンの説明を追加。その他、一部画像を現在のものに更新。
0.92	2021年12月	キーワード検索機能追加。Webサーバーの停止方法追加。
0.91	2021年3月	Webアプリ作成機能の削除、及びPageGeneratorの追加。
0.90	2020年6月	新規作成