Pic C Compiler Limited Edition
Grich RC Inc

このPic C Compilerは、マイクロコントローラーに
適用すべく、Kernigan-Ritchie ANSI C bookに記される
ANSI C言語仕様を使って開発されました。

Grich RC Inc.

120 Cedar Grove La. Ste 340 Somerset NJ, 08873

email: grichrc@aol.com

＊＊＊注意：添付の特許権実施許諾契約をお読みください＊＊＊

ウィンドウズ3.1へのインストール方法

－フロッピードライブに、PicC のフロッピーを入れて下さい。
－プログラムマネージャーの“FILE|RUN”を選んで下さい。
－“a：＼install”を入力して下さい。

次に、PicC C compiler のプログラム・アイテムを作成します。

－プログラムマネージャーの“FILE|NEW”を選んで下さい。
－ダイアログ・ボックスの中の“Program Item”を選んで下さい。
－“Description”フィールドに PicC を入力して下さい。
－“Command Line”フィールドに“c: ＼picc＼bin＼picc.exe”を入力して下さい。
－“Working Directory”フィールドに“c: ＼picc＼src”を入力して下さい。

次にアイコンを変更します。

－“Change Icon”ボタンをクリックして下さい。
－ウィンドウズの警告は無視して下さい。
－Change Icon ダイアログボックスの“Browse”ボタンをクリックして下さい。
－“C：＼picc”に進んで下さい。
－“picc. ico”ファイルを選んで下さい。
－Change icon ダイアログボックスの“ＯＫ”ボタンをクリックして下さい。
－Program Item Properties ダイアログボックスの“ＯＫ”ボタンをクリックして下さい。

ここで新しいプログラムアイテムをダブルクリックすることによって、初めて PicC C compiler を操作することができるようになります。ｃプログラムは、“ｃ＼picc＼src”ディレクトリーに備わっていなければなりません。.obj と.asm は“c：＼picc＼out”のディレクトリーに配置されます。また、.pre ファイルは “c: ＼picc＼tmp”のディレクトリーに配置されます。

プログラミングの用例として、“c: ＼picc＼src”ディレクトリーの中の “lcd６.c”ファイルをご覧頂けます。“c: ＼picc＼docs”ディレクトリーには、この回路／プログラムの配線図が含まれています。

ウィンドウズ９５へのインストール方法

－フロッピードライブに PicC のフロッピーを入れて下さい。
－“START”メニューの中の“RUN”を選んで下さい。
－“a: ＼install”を入力して下さい。

ここで、机上に“Shortcut to Picc”を用意して下さい。

－“START”メニューからウィンドウズ・エクスプローラをスタートさせて下さい。
－“C: ＼picc＼bin”に進んで下さい。
－“Shortcut to Picc”ファイルをクリックして、右側のマウスボタンであなたのデスクトップのスクリーンエリアまでドラッグしてきて下さい。

ここで“Shortcut to Picc”をダブルクリックすることによって、初めてPicC C compiler を操作することができるようになります。cプログラムは、“c＼picc＼src”のディレクトリーに備わっていなければなりません。.obj と.asm は“c：＼picc＼out”のディレクトリーに配置されます。また、.pre ファイルは “c: ＼picc＼tmp”のディレクトリーに配置されます。

プログラミングの用例として、“c: ＼picc＼src”ディレクトリーの中の “lcd６.c”ファイルをご覧頂けます。“c: ＼picc＼docs”ディレクトリーには、この回路／プログラムの配線図が含まれております。

はじめに

このPicC C compilerは、マイクロコントローラーに適用すべく、Kernigan－Ritchie ANSI C bookに記されるＡＮＳＩ　Ｃ 言語仕様を使って開発されました。コンパイラーは３つのファイルを作成します。あなたのPic Programmerには、編集されたコードをPic Microcontrollerの中にダウンロードするべく、obj extension付きのファイルが使われています。この.obj ファイルは、ほとんどのeprom programmerに使用されているIntel Hex 形式によって編集を行います。.asm extensionのあるファイルは、メモリー・マッピングインフォメーションと共に編集されたプログラムのアセンブリーコード・リスティングを含んでいます。このファイルは、あなたのプログラムをデバッグし、コードの遂行サイクルを決定するのに便利です。.pre ファイルは、programの予め演算処理されたバージョンを含み、#defines、#includes等は、プレ・プロセッサー（演算処理装置）によって拡張され、このファイルに配置されます。.pre ファイルは、安全に移動することができます。

プログラムのコンパイル方法

“INCLUDE_DIR pragma”を使って、includeディレクトリーの位置を特定して下さい。
注意：“C: ＼picc＼src＼picc4.cfg”ファイルが存在し、なお且つINCLUDE_DIR定義を含む場合は、
　　このpragmaを特定する必要はありません。（V1.0.2 より）

例：
```
#pragma INCLUDE_DIR  c: ＼picc＼include
#include〈16F84.h〉
main（）
{

int portb @ 6 ;          ／★ nail variable portb to memory location 6      ★／
int trisb  @ 86 ;        ／★ nail variable trisb  to memory location 86     ★／

trisb = 0 ;          ／★ set tris register to all lines output      ★／
portb = 0xaa ;           ／★ binary 10101010      ★／
}
```

履行の拡張

－オプティマイザー（Optimizer）：
　　このコンパイラーは、オプティマイザーを内蔵しており、それによって自動的に非能率的なコードを排除します。

－プレ・プロセッサー（演算処理装置）：
　　以下の構造が組み込まれております：
　　#define identifier token－sequence
　　#ifdef token

```
#else
#endif
#include
#pragma
```

－メモリーロケーション・アドレス明細（変数をメモリーロケーションに位置します）：
変数は@の明細を使って特定のメモリーロケーションに位置することができます。

例：
```
int porta @ 0x5 ;
int trisa @ 0x85 ;
```

注意：　構造における明細の中の変数を、特定のメモリーロケーションに位置しないで下さい。予想もできない結果を生む恐れがあります。

－リザーブされたラベル：
メインはリザーブされた単語であり、プログラムの作成が始まったルーチンです。

__interrupt はリザーブされた単語で、interrupt が受信されたときに作成されたルーチンです。

注意：　コンパイラーは、retfie インストラクションによって、__interrupt ルーチンを終了します。

例：
```
__interrupt ( )
 {
int porta @ 0x5 ;

porta | = (1≪ 3) ;      /★ set bit 3 of port a ★/
porta &= ～(1≪ 3) ;     /★ clear bit 3 of port a ★/

}
```

－埋め込まれているアセンブリーコード：
特定のアセンブリーの指示に対しては、asm（<assembly code>）をお使い下さい。アセンブリーの指示の中で使われる変数は、Cスコープにおいて定義されているはずです。

例 1：
```
int i
asm ( label loop
        nop
        decfsz      i , 1
        goto   loop) ;
```

－ストリング・エレメント（構成分子）にアクセスするには、@オペレーターをお使い下さい；

例：
```
main ( )
 {
char ★ ptr ;
int  count ;
int  upper ;

ptr=  "Hello World ! " ;
```

```
count= 0 ;
while ( @ptr  ! = '\0') {
        upper = toup  (@ptr) ;
        count ++ ;
        ptr ++ ;
        }
}
toup ( char  c)
 {

if (   (c  >=  'a') &&  (c <= 'z' ))  return  (c - 'a' +  'A') ;
else return (c) ;

}
```

ヒント：プログラム・メモリー（ストリング・メモリー）に表を作るには：

```
char ★ table__base ;
int ele ;

table__ base =  "\x0\x1\x2\x3\x4\x5\x6\x7\x8\x9" ;

ele=@ (table__base + 3) ;  /★ assign element  3  of table to ele ★/
```

更に…　：ansi C 標準文字の通常の定義には無い、フォーマット "\d" が付け加えられております。
この構造により、標準のオクタルやヘックスに並び、小数も特定することができます。

例：　table __ base = "\d1\d2\d3\d4\d5\d6\d7\d8\d9\d10" ;

更に…　：また、ストリングの明細を取り除く為、construct identifier=string string ...　;
も付け加えられております。

例：
```
char  ★   ptr ;
ptr=  "This is a string which is specified"
      "in multiple lines.    Note that no semi- colon"
      "is specified until the last last quoted string"
      "In this way strings or tables can be specified"
      "much easier.    Note that this last line has the semicolon." ;
```

－メモリー構成や、プログラムメモリー・サイズ、そしてイントラプトハンドリングを特定するためには、以下のプラグマが使用されます；

内蔵されたディレクトリーを特定するには：

```
#pragma INCLUDE__DIR  C : \picc\include
```

－プログラムの中のプラグマを特定して下さい。それにより、PicCコンパイラーは、内蔵されたファイルの中のどこを探せばよいかを知ることができます。

周辺のイントラプト・ベクターのアドレスを特定するには：

```
#pragma INTERRUPT__LOC  0x4
```

－このプラグマを定義する際には、__INTERRUPT サブルーチンを与えなければなりません。さもないと、コンパイラーはそれをエラーとしてマークしてしまいます。

メモリーバンクを特定するには（4 つ有ります）：

```
#pragma BANK__START__00    0xC
#pragma BANK__END__00      0x2F
```

メモリーバンクの中で一般的なメモリーロケーションを特定するには（FSR、PC…）：

```
#pragma COMMON__BANK__START__0    0x2
#pragma COMMON__BANK__END__0      0x4
```

プラグマ・メモリーのサイズを特定するには：

```
#pragma ROM__START   0n
#pragma ROM__END     0x3ff
```

リセット・ベクター・アドレスを特定するには：

```
#pragma RESET__LOC   0
```

構成ファイル

構成ファイルは picc4.cfg と命名され、PicC 遂行がスタートするのと同じディレクトリー内に備わっていなければなりません。このファイルの不履行は、c：＼picc＼src＼picc４. Cfg の中に位置されます。

以下のタグ値のペアは、PicC C コンパイラーの多種多様な構成変数を特定すべく、picc4.cfg ファイルの中に存在しています。

内蔵されたディレクトリーの位置を特定するには、INCLUDE __DIR タグを使って、内蔵されたファイルの中のディレクトリーの位置を指定して下さい。

INCLUDE __DIR C：＼picc＼include＼

暫定ディレクトリーの位置を特定するには、TMP__DIR タグを使って、暫定ファイルの中のディレクトリーの位置を指定して下さい。暫定ディレクトリーは、コンパイルのプロセスにおいて使われる中間のファイルを保存するのに使われます。

TMP__DIR　C：＼picc＼tmp＼

アセンブリー・アウトプット・ファイルの位置を特定するには、ASM__DIR タグを使用して下さい。同様に、インテルヘックスアウトプットファイルには OBJ __DIR を、あなたの c ソース・コードの位置を特定するには、SRC__DIR を使用して下さい。

```
ASM__DIR   C：＼picc＼out＼
OBJ__DIR   C：＼picc＼out＼
SRC__DIR   C：＼picc＼src＼
```

どのファイルにどのエクステンション（拡張）を使用するか、特定することができます。これは、いくつかのエプロムプログラマーが.obj エクステンションを求めるのに対し、他のものが.hex エクステンションを求める、オブジェクトファイルには特に便利です。一つのエクステンションを特定するのに、３つまでの文字を使用することができます。点（.）は含みません。

```
OBJ__EXTENSION hex
ASM__EXTENSION asm
```

SRC__EXTENSION c
PRE__EXTENSION pre

1．1．4版を初めとし、PicC C コンパイラーは、コールツリーが何の対立も示さないメモリーの位置を再利用することができるようになります。この機能をオンにするには、config ファイルの REUSE__MEMORY 1をセットして下さい。また、オフにするにはREUSE__MEMORY 0 をセットして下さい。

REUSE MEMORY 1

不履行のpicc4．cfg ファイルの内容は（1．0．4版にて紹介されています）：

INCLUDE__DIR C：＼picc＼include＼
TMP__DIR C：＼picc＼tmp＼
ASM__DIR C：＼picc＼out＼
OBJ__DIR C：＼picc＼out＼
SRC__DIR C：＼picc＼src＼

OBJ__EXTENSION hex
ASM__EXTENSION asm
SRC__EXTENSION c
PRE__EXTENSION pre
REUSE__MEMORY 1

イントラプト・ハンドリング：

―__interrupt（）という機能を含んだとき、コンパイラーはW、STAT、そしてPCLATH レジスターをセーブするために、コードを挿入します。FSR はセーブされません。FSR は構成、結合、または配列の際に使われます。もしあなたがこれらのうちのどのデータタイプも使用しないなら、このレジスターをセーブする必要はありません。

―現在のバージョンレベル・V1.1.5 においては、／ * 〉〉〈〈 やφなどといった、特有の数学的記号は__interrupt（）の中には入っておりません。これらを使用すると、予想のできない結果を生む恐れがあります。将来的にはこれらの数学的記号も加わることと思われます。

―以下のプラグマは、イントラプトを使用する際に使われるものです。それらが定義されなかったとき、コンパイラーは警告を出し、一般的な値にデフォルトします。が、それらがあなたがお持ちの PIC に当てはまるとは限りません。適切なデータシート（メモリー明細）により、適切な値をお調べください。

```
#pragma INTERRUPT__SAVE__LOC
#pragma INTERRUPT__LOC
#pragma PCLATH__LOC
```

施行にあたっての詳細：

以下は、ANSI C specification とのコンパイラーの違いについて述べております。

―オープン配列についての定義はされておりません。

Not Implemented： int a［］；

－配列は１次元に限られております。

```
Implemented :       int a [10] ;
Not Implemented :   int a [10] [10] ;
```

－配列の構成／結合は行われません。

```
Implemented :       struct test  {
                    Int a ;
                    Long [b] ;
                    } noarr ;
Not Implemented :   struct test  {
                    Int a ;
                    Long [b] ;
                    } noarr [10] ;
```

－配列を含んだ構成はサポートされております。

例：

```
struct test  {
 int a [10] ;
 long b [10] ;
 } noarr ;
```

－ストリング：

－常にプログラム・メモリーの中に保存されており、８Kバイトに限定されております。

注意：プログラム・メモリーは“LIMITED EDITION （限定版）”においては１Kに限定されており、ストリングもまたこれによって限定されます。

－*オペレーターの代わりに@オペレーターを使ってアクセスされます。

－ストリングには自動準備システムはサポートされておりません。

```
Not Implemented :   char str [10] = “Hello World” ;
Not Implemented :   char str [ ] = “Hello World” ;

Implemented :               char  *  str ;
                      s tr  =  “Hello World” ;
```

－Elipsys... 機能の詳細としてはサポートされておりません。
－タイプキャスティング（鋳造）はサポートされておりません。
－typedef はサポートされておりません。
－sizeof はサポートされておりません。
－Recursion はサポートされておりません。
－機能の試作は履行されません。
－一度にコンパイルされるのは一つのファイルのみです。
－V1．１を初めとし、４つすべてのメモリーバンクがサポートされております。
－プログラムメモリーは、現在最高 1FFF ヘックス（8K）までサポートされております。
注意：リミテッドエディション（限定版）におけるプログラムメモリーは３ｆｆ（1K）に限定されております。
－流動点算術はサポートされておりません。

よくある問題

－内蔵されたコンパイラーの指令や、#ifdef／#endif ブロックは扱われません。

Workaround :　なし

－機能外の指定の変数の定義は、施行されません。

Workaround：そういった変数を、機能の中で準備動作に入らせて下さい。
(main)

問題の例：

```
#include〈16F84. h〉
int  a = 5 5 ;
main  ( )
 {
／／ a is not set to 55 at this point. (a はこの時点においては 55 に設定されません)
  }
```

解決法の例：

```
#include〈16F84. h〉
int  a ;
main ( )
 {
a = 5 5 ;
 }
```

1.1. x における新機能

－4 つのメモリーバンクをサポート
－ページ配分のコールツリー最適化 (pclath bits)

－使われない機能は削除されます
－すべてのページに配置されたレジスターメモリーによって、pic の中のメモリー配分方法論が部分的に改善されています。
－矛盾しない変数間のレジスターメモリーの再使用が可能です。
－多様なコード最適化が追加されています。