## SECTION III

## Programming Utilities

## 3.1. Utility Manual Pages

The following manual pages present the details of Whitesmiths' program development utilities. For an explanation of the format and effective use of manual pages, see section 1.2.6, entitled "Manual Page Conventions".

NAME
    c - multi-pass command driver

SYNOPSIS
    c -[delim* d*^ o* prefix* proto* sep* s v x +*^] <files>

FUNCTION
c is designed to simplify multi-pass operations, such as compiling files, by expanding commands from a prototype file for each of its file arguments. Which passes of the compiler are invoked, and what the results will be relative to the commands in the prototype file, depends on the suffixes of the filenames passed to the driver.

The prototype file, called **c.pro** on most systems, is located in an appropriate system directory. It is searched for along the invoker's default search path.

The driver is invoked by typing **c**, along with all the desired flags to the various passes of the compiler, on a single command line. In the absence of flags, the system will default to compiling your files and writing the name of each to STDOUT as it is processed. Any error messages are written to STDERR.

If any step fails, the driver stops processing the current argument and creates a file called **nolink.e** to indicate that grouping (linking) should not be performed. Any given step may also choose to create this file directly, instead of reporting failure, in order to disable grouping and yet continue any remaining transformations on the current argument.

The default name of **nolink.e** may be overridden by setting the environment variable CERRFILE to the new file name.

c accepts the following flags:

-delim* change delimiter characters between the file name and suffix on host systems where the period . is not an acceptable symbol.

-d*^ specify * as the name of a user-defined programmable flag. Programmable flags are host/target-specific and are discussed below.

-o* send output to the file *, retaining and appending to * the suffix for the bound binary image (if any) specified in the prototype file.

-prefix* prefix names of all permanent output files with the pathname *. Note that under DOS, the pathname on the file is not stripped off before prepending the prefix name.

-proto* take prototype input from the file *.

-sep* separate filenames in the grouping line with the character *. The default is a space.

-s be "silent". Do not output filenames or arguments during compilation.

-v be "verbose". Before executing a command, print the command with arguments to STDOUT. The default is to output only the names of each file processed, and the root name when (and if) the grouping line is run, with each name followed by a colon and a newline. The root name consists of the source line without its suffix. If **-v** and **-s** are both specified, -v is performed.

-x do not execute the commands in the prototype file. Instead, write the commands which otherwise would have been performed to STDOUT.

+* save a permanent copy of the (otherwise temporary) intermediate file specified by the unique suffix *, and halt processing after the last file requested has been created (i.e. if output from **cp1** (+2) is requested, compilation will stop after the **cp1** pass). Up to ten instances of +* are accepted. The possible values for * in the standard product as shipped by Whitesmiths, Ltd. are any or all of the suffixes that the compiler appends to the output of each pass. Users can create their own suffixes for the various classes of files. The compiler uses the labels that precede the steps within the prototype file to generate the suffixes it appends to the output of each pass. The default suffixes for your particular system are **.c** for C source files, for the output of **pp**, for the output of **cp1**, and **.asm** for the output of p236.

## Programmable Flags

The following pages list the programmable flag options that are built into your system as part of the standard product release.

**-d*^** specify * as the name of a user-defined "programmable" flag. Up to twenty programmable flags may be specified.

<u>Standard Programmable Flags</u>

**-ddebug** compile with debugger on. Only modules compiled with this flag will execute in debug mode when the program is run.

**-dlistcs** merge C source listing with assembly code; listing output defaults to STDOUT.

**-dlistps** merge Pascal source listing with assembly code; listing output defaults to STDOUT.

**-dlistpc** merge Pascal source listing with C (the output of **ptc**); listing output defaults to STDOUT.

**-dlistpcs** merge Pascal source, C, and assembler; listing output defaults to STDOUT.

Note that only -dlistcs has meaning in the absense of Pascal.

In addition, there are two flags which can be used in conjunction with the **-dlist*** programmable flags:

> **-dlincl** force a listing of all #include files to be included with the source listing, and cause diagnostics (when produced) to indicate the actual #include file name and line number.
>
> **-dlo** redirect listing from default STDOUT to the file **root.lst**, where **root** is the source file minus its suffix.

**-dproto** enable prototype checking.

<u>Target Architecture Specific Flags</u>

**-dcatg** assign category number 20 to the assembled subroutine member. This flag is useful when creating overlays.

**-dcs** merge C source listing with assembly language code and redirect the listing from the default STDOUT to the file **root.asm**, where **root** is the source file minus its suffix.

**-dm** make **root** the name of the assembled subroutine member, where **root** is the source file minus its suffix. Otherwise, the subroutine member will get the name of the first global function, or, if there are no functions in the source text, the name of the first global data object.

Of the programmable flags listed here, all but **-dcatg**, **-dcs** and **-dm** are portable; they are available with every Whitesmiths Version 3.0 compiler product, regardless of target architecture. <u>To find out exactly what programmable flags are built into your package and what each one does,</u>

review the opening lines of the compiler driver prototype file **c.pro**, where the name and an English-language description of each programmable flag is listed.

RETURNS

c returns success if it can open all files successfully. It prints a message to STDERR and returns failure if there are errors in the prototype file, or if any files cannot be opened.

EXAMPLE

To compile **login.c** and observe the name and arguments of each command in the compilation process as it is executed:

```
C> c -v login.c
```

## NAME

CALINK - link C program using the ANSI library (S/36)

## SYNOPSIS

```
CALINK <main>[,YES|NO|XREF|MSG]
```

## FUNCTION

CALINK is a System/36 procedure member that will link the subroutine member **main** into a load member **main** using the library **#CALIB**.

The optional second parameter is a listing option:

| | |
|---|---|
| YES | print storage map and messages |
| NO | print no storage map or messages |
| XREF | print storage map cross-reference and messages |
| MSG | print only messages (this is the default) |

## EXAMPLE

```
CALINK MAIN
```

## SEE ALSO

IBM System/36 Overlay Linkage Editor Guide (SC21-9026)

## NAME

CEALINK - link C program using the extended ANSI library (S/36)

## SYNOPSIS

```
CEALINK <main>[,YES|NO|XREF|MSG]
```

## FUNCTION

CEALINK is a System/36 procedure member that will link the subroutine member **main** into a load member **main** using the library #CEALIB.

The optional second parameter is a listing option:

|  |  |
|---|---|
| YES | print storage map and messages |
| NO | print no storage map or messages |
| XREF | print storage map cross-reference and messages |
| MSG | print only messages (this is the default) |

## EXAMPLE

```
CEALINK MAIN
```

## SEE ALSO

IBM System/36 Overlay Linkage Editor Guide (SC21-9026)

**NAME**

frwrk36 - transfer assembler source from System/36 (PC-DOS)

**SYNOPSIS**

```
frwrk36 <file>
```

**FUNCTION**

To simplify the use of the IBM program product **PC Support** when transferring assembler source files from System/36 to a PC, use the batch file **frwrk36.bat**. This command will send the source member **<file>** in library **WRKLIB** on System/36 to the PC. The extension **.asm** will be added to the file.

You must change **frwrk36**.bat if you want to transfer files from a library other than WRKLIB.

**RETURNS**

**frwrk36** will emit a warning if it does not succeed in sending the file to the PC.

**EXAMPLE**

```
    frwrk36 main
```

NAME

lm - correlate lines between files

SYNOPSIS

lm -[c* err e* f l o* s t] <files>

FUNCTION

**lm** writes to STDOUT a copy of the lines contained in the merge files specified by **<files>**, replacing any occurrence of the sequence

**#line= <filename>:<linenumber>**

with the appropriate line or range of lines (from the last line printed to the specified line) from the "source" file named **filename**. (For the purposes of this discussion, a "merge" file is any file specified on the **lm** command line, and a "source" file is any file referred to by **#line** and **#error** sequences embedded in one or more merge files.) If that line has already been printed, no replacement occurs. If more than one merge file is specified, each is initially scanned, in order of appearance on the command line, for the sequence described. The order in which the files are further scanned depends primarily on line number orderings. If no files are specified, or if - appears as a filename, then input is expected from STDIN. At most one copy of a given filename/linerange combination will be inserted into a given output.

The sequence

**#error <program> <filename>:<linenumber>'\t'<text>**

is also recognized by **lm**, which replaces it with the appropriate line or range of lines from the specified source file ('\t' refers to an ASCII tab character). The **#error** sequence will cause **lm** to output the string `~~e` to the output file the first time that a #error sequence is encountered. The #error sequence also forces **lm** to assert that all lines up to **linenumber** have indeed been copied to STDOUT. This sequence is also recopied to the output rather than replaced, (as **#line=** directives normally are). (**Note:** This only occurs the first time the original sequence of lines is seen among the files named.)

By default (i.e. if **-err** is not specified), **lm** reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

**lm** accepts the following flags:

-c* Force each "source" line copied to STDOUT to be prefixed by * as a comment delimiter.

-err Pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as being unable to write an output file due to lack of space) will be reported and cause the return status to change to failure.

**-e*** Force each "error" line copied to STDOUT to be prefixed by * as a comment delimiter.

**-f** For use with a foreign assembler, this flag suppresses outputting of the string -e as described above, counts the number of "error" lines, and outputs the number of high-level errors at the end of the listing.

**-l** Number each "source" line copied to STDOUT. Each "source" file is numbered independently and the line number appears after the comment string (which may be null).

**-o*** Send output to the file * rather than to STDOUT.

**-s** Strip all #error and excess #line= sequences from the merge files.

**-t** Strip only the excess #line= sequences from the merge files.

## RETURNS

lm returns success if it can open its output file and all source files, or if the -err flag is specified and no fatal errors occur.

## EXAMPLE

To get a listing of C interspersed with assembly language:

```
C> c -dlistcs +o echo.c
```

## NOTES

The line ranges are restricted to be monotonically increasing in value between files.

## NAME

cp1 - parse C programs

## SYNOPSIS

```
cp1 -[a b# c +dead +debug err l lreg model? m n# +old o* r# sr +std
+strict strict u] <file>
```

## FUNCTION

cp1 is the parsing pass of the C compiler. It accepts a sequential file of lexemes from the preprocessor **pp** and outputs a sequential file of flow graphs and parse trees suitable for input to the code generator **p236**. The operation of cp1 is largely independent of any target machine.

cp1 accepts the following flags:

**-a** compile code for machines with separate address and data registers (currently used only in conjunction with the MC68000 code generator).

**-b#** enforce storage boundaries according to #, which is reduced modulo 4. A bound of 0 leaves no holes in structures or auto allocations; a bound of 1 (default) requires short, int and longer data to begin on an even bound; a bound of 2 is the same as 1, except that 4-8 byte data are forced to a multiple of four byte boundary; a bound of 3 is the same as 2, except that 8 byte data (doubles) are forced to a multiple of eight byte boundary.

**-c** ignore case distinctions in testing external identifiers for equality, and map all names to lowercase on output. By default, case distinctions matter.

**+dead** flag unused variables with an error message. The default is not to complain about unused variables.

**+debug** generate debugging information for use by the source code debugger cdb. The default is to output no debugging information.

**-err** pass error diagnostics from input file to output file, add error diagnostics generated to the output file, and report success. Only fatal errors, such as not being able to write an output file due to lack of space, will be reported and cause the return status to change to failure.

**-l** take integers and pointers to be 4 bytes long. The default is 2 bytes.

**-lreg** take machine registers to be 4 bytes long. This flag should be used only in conjunction with the MC68000 code generator.

**-model?** take ? as a memory model designator. This flag is currently used only in conjunction with the 8086 and MC68000 code generators. For the 8086, the memory models supported by the compiler are **s, p, d,** and **f**, signifying small, compact, medium, and large models, respectively. The default is the small model. For the MC68000, the only valid memory model designation is **-modelf**, which, in conjunction with the absence of the -l flag, specifies integer size to be 16 bits and pointer size to be 32 bits.

-m treat each struct/union as a separate name space, and require **x.m** to have a structure **x** with **m** as one of its members.

-n# ignore characters after the first # in testing external identifiers for equality. The default is 31. The maximum is 63.

+old generate intermediate input to p236 that is compatible with Edition 2.2 of the Whitesmiths C compiler. The default is to generate intermediate input that is compatible with Version 3.0 of Whitesmiths C compiler. This flag is for internal use only.

-o* write output to the file * and write error messages to STDOUT. The default is STDOUT for output and STDERR for error messages.

-r# assign no more than # variables to registers at any one time, where # is reduced modulo 7. The default is 3 register variables. Values above 3 are currently acceptable only for the MC68000 code generator (3 data registers + 3 address registers maximum), and the VAX-11 code generator (6 registers maximum).

-sr make strings read only (i.e. put them in the text section). The default is to make strings both readable and writeable.

+std force the input to conform to the ANSI C draft standard.

+strict enforce much stronger type checking, as described below.

-strict allow more lenient (weaker) type checking, as described below. By default, the type checking done by cp1 is between the two extremes +strict and -strict.

-u take a string "string" to be of type **array of unsigned char**, not array of char.

If <file> is present, it is used as the input file instead of the default STDIN. On many systems (other than IDRIS/UNIX), use of the **-o** option and specification of an input file <file> are mandatory because STDIN and STDOUT are interpreted as text files, and hence become corrupted.

By default (i.e. if **-err** is not specified), cp1 reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

If the -strict flag is present, no diagnostic is generated when:

1) an integer is assigned a pointer value.

2) a pointer is assigned an integer constant value other than NULL (0).

3) two different pointer types are checked for assignment compatibility.

4) the types for the result of a conditional operator do not match (i.e. x ? a : b where a and b have different types).

If the +strict flag is present, a diagnostic is generated:

1) if an argument to a function definition does not have an explicit type associated with it by the time the opening { for the function body is encountered (i.e. f(abc){}). Without strict type checking, the assumed type for an untyped argument is int.

2) for any implicit narrowing assignment. This affects initialization, function call arguments when a function prototype exists, and implicit assignment statements. A cast may be used to disable the checking on a case-by-case basis. Without strict type checking, a narrowing assignment will truncate without complaint.

3) if the return value of a function is used when the return type of the function was not previously declared. Without strict type checking, the assumed type for a function is int.

4) if an argument to a function is not the same type as the type specified in the prototype declaration.

**RETURNS**

cp1 returns success if it produces no diagnostics, or it the -err flag is specified and no fatal errors occur.

**EXAMPLE**

cp1 is usually invoked between pp and p236, as in:

```
pp -x -o temp1 file.c
cp1 -o temp2 temp1
p236 -o file.s temp2
```

**SEE ALSO**

pp

**NOTES**

cp1's processing of semicolons can be difficult to predict.

## NAME

p236 - generate code for IBM System/36 C programs

## SYNOPSIS

p236 -[bss catg# ck cs* ds* err +list m* o* ps# p x#] <file>

## FUNCTION

p236 is the code generating pass of the C compiler. It accepts a sequential file of flow graphs and parse trees from cp1 and outputs a sequential file of assembly language statements suitable for input to the IBM System/36 assembler.

As much as possible, the compiler generates free-standing code, but, for those operations which cannot be done compactly, it generates inline calls to a set of machine-dependent runtime library routines. The names of the routines in the Machine Interface Library are listed in Section IV of the <u>Interface Manual</u> for your machine.

p236 accepts the following flags:

-bss inhibit generating code by a DS (define space) directive. By default, data initialized to zero is defined by a DS directive.

-ck enable stack overflow checking.

-catg# give the resulting object module category number # by including a HEADERS CATG-# statement in the output assembly language file.

-cs* use the name * as the name of the code segment in the assembly language output. The default is the string "code". This flag is ignored when generating IBM assembler.

-ds* use the name * as the name of the data segment in the assembly language output. The default is the string "data". This flag is ignored when generating IBM assembler.

-err pass error diagnostics from input file to output file, add error diagnostics generated to the output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

+list generate line number directives interspersed with assembly code to be used by the lm utility to provide a listing file. The default is to turn listings off.

-o* write the output to the file * and write error messages to STDOUT. Default is STDOUT for output and STDERR for error messages.

-m* set object output module name to * when generating the assembly language code through the use of a START assembly directive. The default module name is the name of the first non local function. If there are no functions in the source file, the module will get the name of the first data variable, which must not be local.

- **-p** emit profiler calls on entry to each function. Currently this flag can be used only with the IDRIS profiler.

- **-x#** map the three virtual sections, for Functions (04), Constant Data (02), and Variables (01), to the two physical sections: Code (bit is a one) and Data (bit is a zero). Thus, **-x4** is for separate text and data segments, which is the default. This has no effect when used with the IBM assembler as it does not maintain separate text and data sections.

If **<file>** is present, it is used as the input file instead of the default STDIN. On many systems (other than IDRIS/UNIX), specifying a **<file>** is mandatory, because STDIN is interpreted as a text file, and hence becomes corrupted.

Files output from cp1 for use with the IBM System/36 code generator must be generated with the following options: no **-l** flag, since pointers are short; use of -b0 for compact data, **-r0** since there are no register variables and no **-a** flag, since registers are interchangeable. cp1 should also be run with the flags -cn6 to check externals properly.

Wherever possible, labels in the emitted code each contain a comment which gives the source line to which the code immediately following pertains, along with a running count of the number of bytes of code produced for a given function body.

**RETURNS**

p236 returns success if it produces no diagnostics, or if the **-err** flag is specified.

**EXAMPLE**

p236 usually follows pp and cp1, as follows:

```
pp -o temp1 -x file.c
cp1 -o temp2 -cmu -b0 -n6 -r0 temp1
p236 -o file.s temp2
```

**SEE ALSO**

cp1

**NOTES**

Stack overflow checking is only approximate, since a calculation of the exact stack high water mark is not attempted.

## NAME

pp - preprocess defines and includes

## SYNOPSIS

```
pp -[d*^ err i* +lincl +map* +old o* +pas p? +std s? x] <files>
```

## FUNCTION

pp is the preprocessor used by the C compiler to expand #defines, #includes, and other directives signalled by a #, before actual compilation begins. It can be used to advantage, however, with most language processors.

pp accepts the following flags:

-d*^ where * has the form name=def, define name with the definition string def before reading the input. If =def is omitted, the definition is taken as 1. The name and def must be in the same argument, i.e., no blanks are permitted unless the argument is quoted. Up to ten definitions may be entered in this fashion.

-err pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

-i* change the prefix used with #include <filename> from the default (no prefix) to the string *. Multiple prefixes (including the null prefix) to be tried in order may be specified, separated by the character |.

+lincl force a listing of all include files to be included with the source listing, and make any diagnostic output indicate the actual #include file name and line number. The default is not to include a listing of #include files.

+map* map characters specified in strings according to values given in the map file. The map file should contain 256 bytes, each representing the value of the character set for the target machine. The default is no mapping.

+old generate tokens that are compatible with Edition 2.2 of Whitesmiths C compiler. This flag is for internal use only.

-o* write the output to the file * and write error messages to STDOUT. Default is STDOUT for output and STDERR for error messages. On DOS, use of the -o option is mandatory with -x because STDOUT is interpreted as a text file, and therefore becomes corrupted.

+pas assume Pascal-style {comments} in the source; i.e., treat anything between { and } as a comment and everything between /* and */ as source code.

-p? change the preprocessor control character from # to ?.

+std enforce lexical elements in the input file to meet the lexical requirements given in the ANSI C draft standard.

-s? change the secondary preprocessor control character to ?. By default, the secondary preprocessor control character is disabled.

-x put out lexemes for input to the C compiler parser **cp1**, not lines of text.

pp processes the named files (or STDIN if none are given) in the order specified, and the resultant text is written to STDOUT. Preprocessor actions are described in detail in Section II of the C Language Manual.

The presence of a secondary preprocessor control character permits two levels of parameterization. For instance, the invocation

```
pp -p@
```

will expand **@define** and **@ifdef** conditionals, leaving all # commands intact; invoking pp with no arguments would expand only # commands.

By default (i.e. if **-err** is not specified), pp reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

RETURNS

pp returns success if it produces no error diagnostics, or if the **-err** flag is specified.

EXAMPLE

The standard style for writing C programs is:

```
/*  name of program
 */
#include <wslxa.h>

#define MAXN    100

COUNT things[MAXN];
...
```

The use of uppercase-only identifiers is not required by pp, but is strongly recommended to distinguish parameters from normal program identifiers and keywords.

NOTES

Floating constants longer than 38 digits may compile incorrectly on some host machines.

SEE ALSO

cp1, ptc

NAME
pr - print files in pages

SYNOPSIS
pr -[attn* c# err f# h it# l# m n ot# p s? t* w# +## ##] <files>

FUNCTION
pr prints to STDOUT the files in the list <files>, adding a title and empty lines for page breaks, and padding to an integral number of pages. If no <files> are given, pr takes input from STDIN. A filename of - also causes STDIN to be read, at the point in the list of files where the - appears. Each page output has a 5-line heading containing two empty lines, a title line, a subtitle line, and another empty line, along with a 5-line footing.

The standard title consists of the last modification date of the file being output (where available), its name, and the current page number. When a user-specified title is given, or when STDIN is the file being output, the current date and time are used instead of a modification date. The standard subtitle is an empty line.

pr also has features which allow it to be used in conjunction with the C/Pascal compiler and the lm listing merger utility to generate more readable listings. pr examines each line of the files printed for special commands to itself. pr understands three basic kinds of embedded commands, each of which must begin with an "attention sequence". The default attention sequence is ~~ (two ASCII "tilde" characters). It can be changed via the -attn* flag, as described below. A line on which only the attention sequence appears (~~ if the default is in effect) is taken as a request to begin a new page (a newpage command). If a line begins with the attention sequence followed by a 1, (~~1 if the default is in effect) everything between the command sequence and the next ASCII newline character is taken as a new title for the page header, to be output at the top of the next page output. If a line begins with the attention sequence followed by a 2, (~~2 if the default is in effect) everything between the command sequence and the next ASCII newline character is taken as a new subtitle, to be output below the header at the top of the next page output. If a line begins with the attention sequence followed by an e, (~~e if the default is in effect), and the -err flag is specified, pr will by default create a file called nolink.e. This default can be overridden by setting the environment variable CERRFILE to the new file name.

pr accepts the following flags:

-attn* set the attention sequence to *. The default attention sequence is ~~.

-c# print each file in # columns. The default is 1 column.

-err pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

**-f#** set the number of the first page output to #.

**-h** suppress headings and footings on output.

**-it#** space incoming tabs (tabs in the input files) at intervals of #. The default is 8 spaces between tabstops (4 spaces between tabstops under IDRIS). The **-it0** flag suppresses input detabbing.

**-l#** output pages # lines in length. The default is 66 lines per page.

**-m** print all files simultaneously, each in a separate column.

**-n** number the output lines.

**-ot#** entab the output assuming a tabstop every # columns. The default is 0, meaning that output is not entabbed unless otherwise specified.

**-p** print one page of output and pause, waiting for input from STDIN.

**-s?** use a single ? to separate multiple columns of output, instead of whitespace. ? always matches the next argument character, if any, or a NUL character. When this option is specified, entabbing is suppressed.

**-t*** use * as the filename field of the heading title. Note that any ~~? sequences encountered by **pr** will override this flag.

**-w#** specifies a page width of # positions. The default is 72.

**+##** start output of each file with page ##. The default is to start output at page 1.

**-##** stop output of each file after page ##. The default is huge.

At most one of the flags **-c#** and **-m** may be specified.

By default (i.e. if the -err flag is not specified), **pr** reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

**RETURNS**

**pr** returns success if no error messages are written to STDERR, or if the **-err** flag is specified.

**EXAMPLE**

To number the output lines of a file:

```
    C> pr -n file1


Tue Sep 24 15:15:56 1986  file1  Page 1


    1       file 1 line 1
    2       file 1 line 2
    3       file 1 line 3
    4       file 1 line 4
    5       file 1 last
```

To add a special filename field to the heading title:

```
    C> pr -t"EXAMPLE OF pr OUTPUT" file2
```

```
Tue Sep 24 15:24:44 1986 EXAMPLE OF pr OUTPUT  Page 1


file 2 line 1
file 2 line 2
file 2 line 3
file 2 line 4
file 2 line 5
file 2 line 6
file 2 line 7
file 2 line 8
file 2 line 9
file 2 last
```

## NAME

ptc - Pascal to C translator

## SYNOPSIS

ptc -[c err f i* +list m# n# o* r +std str# s# v] <ifile>

## FUNCTION

**ptc** is a program that accepts as input lines of Pascal text and produces as output a corresponding C program which is acceptable to the Whitesmiths C compiler. If **<ifile>** is present, it is taken as the Pascal program to translate; otherwise input is taken from STDIN. The Whitesmiths Pascal Language Manual describes the Pascal language as processed by **ptc.**

ptc accepts the following flags:

-c pass comments through to the C program.

**-err** pass error diagnostics from input file to output file, add error diagnostics generated to the output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

-f set the precision for reals to single precision (float). The default is double precision.

-i* change the prefix used with the "#include <filename>" from the default (no prefix) to the string *. Multiple prefixes to be tried in order may be specified, provided they are separated by the character |.

+list insert #line directive into input to be processed by the C compiler.

-m# make # the number of bits in MAXINT excluding the sign bit, e.g., MAXINT becomes 32767 for -m15, 1 for -m1, etc. Default for MAXINT is 32766 [sic]. Acceptable values for # are in the range [0, 32). Declaring MAXINT to be greater than the size of a pointer (16 or 32 bits) will give unpredictable results. On a target machine with 16-bit pointers, # should be less than 16 bits.

-n# Make # the number of significant characters in external names. The default is 31-character external names.

-o* write the C program to the file * and diagnostics to STDOUT. The default is STDOUT for the C program and STDERR for diagnostics.

-r turn off runtime array bounds checks.

+std force the input to ptc to conform to the ISO Pascal standard.

-str# set string size to #. The default is 64 characters.

-s# make # the number of bits in the maximum allowable set size, i.e., the size of all sets, the basetype of which is integer, becomes the specified power of two. Acceptable values are in the range [0, 32). Default is 8 (256 elements).

-v be verbose.

The compiler prepends a leading underscore _ character to the name of any global or local identifier whose name conflicts with a C keyword. The compiler does not alter global identifiers that do not conflict with C keywords. Identifiers local to a procedure, however, have a unique four-digit number prepended to their names to ensure that all local names are unique. Moreover, structure declarations may be produced that contain conflicting field declarations; and declarations are present for library functions that may not be needed. All of these idiosyncracies are forgiven by the use of appropriate C compiler options.

By default (i.e. if -err is not specified), ptc reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

**RETURNS**

**ptc** returns success if it produces no diagnostics, or if the **-err** flag is specified.

## NAME

towrk36 - transfer assembler source to System/36 (PC-DOS)

## SYNOPSIS

towrk36 <files>

## FUNCTION

To simplify the use of the IBM program product **PC Support** when transferring assembler source files to System/36, use the batch file **towrk36.bat**. This command adds the extension **.asm** to all named files on the PC and transfers them to source members in the library WRKLIB on System/36.

You must change towrk36.bat if you want to transfer files to a library other than WRKLIB.

## RETURNS

**towrk36** will emit a warning if it does not succeed in sending the file to System/36.

## EXAMPLE

```
towrk36 main
```