



TMS470R1x Code Generation Tools for C/C++

Release 1.00

Getting Started Guide

Preliminary

Preliminary



TMS470R1x Code Generation Tools for C/C++ Getting Started Guide

Release 1.00

Literature Number: SPNU177
January 1998



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

About This Manual

The *TMS470R1x Code Generation Tools for C/C++ Getting Started Guide* tells you how to install release 1.00 of the TMS470R1x C/C++ code generation tools on your system. It also provides the following:

- The information you need to set environment variables for parameters that you use often
- The information you need to get started using the compiler, linker, and assembler
- A list of the media contents for your toolset, so you will know what information is associated with each file you have installed

Notational Conventions

In this document, the following notational conventions are used:

- Program listings, program examples, and interactive displays are shown in a special typeface. Examples use a **bold version** of the special typeface for emphasis. Interactive displays use **bold** to distinguish commands that you enter from items that the system displays (such as prompts, command output, error messages, etc.). Some interactive displays use *italics* to describe the type of information that should be entered.

Here is a program example:

```
.global inclw

start: MOV     r6, #0
      MOV     r7, #0

loop:  BL      inclw
      BCC    loop

      .end
```

Here is an example of a command that you might enter:

```
set PATH=c:\tool_dir;%PATH%
```

To change your path statement to use the tools, enter the command text as shown in bold and replace *tool_dir* with the name of your tools directory.

- In syntax descriptions, the instruction, command, or directive is in a **bold typeface** font and parameters are in an *italic typeface*. Portions of a syntax that are in **bold** should be entered as shown; portions of a syntax that are in *italics* describe the type of information that should be entered.

Here is an example of a command that you might use:

```
mkdir tool_dir
```

In this example, you would type `mkdir`, as shown, and replace *tool_dir* with the name of your directory.

- Square brackets ([and]) identify an optional parameter. If you use an optional parameter, you specify the information within the brackets; you do not enter the brackets themselves. Here is an example of a command that has optional parameters:

```
SET C_DIR=pathname1[:pathname2 . . .]
```

Setting the C_DIR environment variable allows you to specify one or more pathnames for the C/C++ compiler to search.

Related Documentation From Texas Instruments

The following books describe the TMS470R1x and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800)477-8924. When ordering, please identify the book by its title and literature number.

TMS470R1x Assembly Language Tools User's Guide (literature number SPNU118) describes the assembly language tools (assembler, linker, and other tools used to develop assembly language code), assembler directives, macros, common object file format, and symbolic debugging directives for the TMS470R1x devices.

TMS470R1x C Source Debugger User's Guide (literature number SPNU124) describes the TMS470R1x emulator and simulator versions of the C source debugger interface. This book discusses various aspects of the debugger interface, including window management, command entry, code execution, data management, and breakpoints. It also includes a tutorial that introduces basic debugger functionality.

TMS470R1x Optimizing C/C++ Compiler User's Guide (literature number SPNU151) describes the TMS470R1x C/C++ compiler. This C/C++ compiler accepts ANSI standard C/C++ source code and produces assembly language source code for the TMS470R1x devices.

TMS470R1x User's Guide (literature number SPNU134) describes the TMS470R1x RISC microcontroller, its architecture (including registers), ICEBreaker module, interfaces (memory, coprocessor, and debugger), 16-bit and 32-bit instruction sets, and electrical specifications.

Related Documentation

You can use the following books to supplement this getting started guide:

The Annotated C++ Reference Manual, by Margaret A. Ellis and Bjarne Stroustrup, published by Addison-Wesley Publishing Company, Reading, Massachusetts, 1990

Working Paper for Draft Proposed International Standard for Information Systems—Programming Language C++ X3J16/WG21, American National Standards Institute

Trademarks

HP-UX, HP 9000 Series 700, and PA-RISC are trademarks of Hewlett-Packard Company.

Microsoft, Windows, and Windows NT are registered trademarks of Microsoft Corporation.

PC is a trademark of International Business Machines Corp.

Pentium is a trademark of Intel Corporation.

SPARCstation is trademark of SPARC International, Inc., but licensed exclusively to Sun Microsystems, Inc.

SunOS and Solaris are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

If You Need Assistance . . .

<input type="checkbox"/> World-Wide Web Sites TI Online http://www.ti.com Semiconductor Product Information Center (PIC) http://www.ti.com/sc/docs/pic/home.htm Microcontroller Home Page http://www.ti.com/sc/micro
<input type="checkbox"/> North America, South America, Central America Product Information Center (PIC) (972) 644-5580 TI Literature Response Center U.S.A. (800) 477-8924 Software Registration/Upgrades (214) 638-0333 Fax: (214) 638-7742 U.S.A. Factory Repair/Hardware Upgrades (281) 274-2285 U.S. Technical Training Organization (972) 644-5580 Microcontroller Hotline (281) 274-2370 Fax: (281) 274-4203 Email: micro@ti.com Microcontroller Modem BBS (281) 274-3700 8-N-1
<input type="checkbox"/> Europe, Middle East, Africa European Product Information Center (EPIC) Hotlines: Multi-Language Support +33 1 30 70 11 69 Fax: +33 1 30 70 10 32 Email: epic@ti.com Deutsch +49 8161 80 33 11 or +33 1 30 70 11 68 English +33 1 30 70 11 65 Francais +33 1 30 70 11 64 Italiano +33 1 30 70 11 67 EPIC Modem BBS +33 1 30 70 11 99 European Factory Repair +33 4 93 22 25 40 Europe Customer Training Helpline Fax: +49 81 61 80 40 10
<input type="checkbox"/> Asia-Pacific Literature Response Center +852 2 956 7288 Fax: +852 2 956 2200
<input type="checkbox"/> Japan Product Information Center +0120-81-0026 (in Japan) Fax: +0120-81-0036 (in Japan) +03-3457-0972 or (INTL) 813-3457-0972 Fax: +03-3457-1259 or (INTL) 813-3457-1259
<input type="checkbox"/> Documentation When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number. Mail: Texas Instruments Incorporated Email: micro@ti.com Technical Documentation Services, MS 702 P.O. Box 1443 Houston, Texas 77251-1443

Note: When calling a Literature Response Center to order documentation, please specify the literature number of the book.

Contents

1	Setting Up the Code Generation Tools With Windows 95 and Windows NT	1-1
	<i>Provides instructions for installing the code generation tools on PCs running Windows 95 or Windows NT.</i>	
1.1	System Requirements	1-2
	Hardware checklist	1-2
	Software checklist	1-2
1.2	Installing the Code Generation Tools	1-3
1.3	Setting Up the Code Generation Environment	1-4
	Identifying the directory that contains the executable files (PATH statement)	1-5
	Identifying alternate directories for the assembler to search (A_DIR)	1-5
	Identifying alternate directories for the compiler and linker to search (C_DIR)	1-6
	Setting default shell options (C_OPTION)	1-6
	Specifying a temporary file directory (TMP)	1-7
	Resetting defined environment variables	1-7
	Verifying that the environment variables are set	1-7
1.4	Where to Go From Here	1-8
2	Setting Up the Code Generation Tools on a SPARCstation	2-1
	<i>Provides instructions for installing the code generation tools on SPARCstations running SunOS version 5.5x (or higher) or Solaris.</i>	
2.1	System Requirements	2-2
	Hardware checklist	2-2
	Software checklist	2-2
2.2	Installing the Code Generation Tools	2-3
	Mounting the CD-ROM	2-3
	Copying the files	2-4
	Unmounting the CD-ROM	2-4
2.3	Setting Up the Code Generation Environment	2-5
	Identifying the directory that contains the executable files (path statement)	2-6
	Identifying alternate directories for the assembler to search (A_DIR)	2-6
	Identifying alternate directories for the compiler and linker to search (C_DIR)	2-7
	Setting default shell options (C_OPTION)	2-7
	Specifying a temporary file directory (TMP)	2-8
	Reinitializing your shell	2-9
	Resetting defined environment variables	2-9
	Verifying that the environment variables are set	2-10
2.4	Where to Go From Here	2-10

3	Setting Up the Code Generation Tools on an HP Workstation	3-1
	<i>Provides instructions for installing the code generation tools on HP 9000 Series 700 PA-RISC computers running HP-UX.</i>	
3.1	System Requirements	3-2
	Hardware checklist	3-2
	Software checklist	3-2
3.2	Installing the Code Generation Tools	3-3
	Mounting the CD-ROM	3-3
	Copying the files	3-3
	Setting up the software tools using a C shell	3-4
	Setting up the software tools using a Korn shell	3-4
	Unmounting the CD-ROM	3-4
3.3	Setting Up the Code Generation Environment	3-5
	Identifying the directory that contains the executable files (path statement)	3-6
	Identifying alternate directories for the assembler (A_DIR)	3-6
	Identifying alternate directories for the compiler and linker (C_DIR)	3-7
	Setting default shell options (C_OPTION)	3-7
	Specifying a temporary file directory (TMP)	3-8
	Reinitializing your shell	3-9
	Resetting defined environment variables	3-9
	Verifying that the environment variables are set	3-10
3.4	Where to Go From Here	3-10
4	Getting Started With the Code Generation Tools	4-1
	<i>Provides instructions on how to invoke and use the assembler, linker, and compiler.</i>	
4.1	Getting Started With the Assembler and Linker	4-2
4.2	Getting Started With the C/C++ Compiler	4-7
5	Release Notes	5-1
	<i>Describes the media contents and the supported C++ language features for this release.</i>	
5.1	Media Contents	5-2
5.2	C++ Language Support	5-6
	Additional C++ language features supported	5-6
	Unsupported C++ language features defined in the X3J16/WG21 Working Paper ...	5-8
A	Glossary	A-1
	<i>Defines terms and acronyms used in this book.</i>	

Tables

5-1	Media Contents for SPARCstations and HP Workstations	5-2
5-2	Media Contents for PCs	5-4

Examples

4-1	file1.asm	4-2
4-2	file2.asm	4-2
4-3	file2.lst, the Listing File Created by asm470 file2.asm -l	4-3
4-4	Output Map File, Inker2.map	4-4
4-5	Sample Linker Command File, linker2.cmd	4-5
4-6	Linker Map File (linker2.map) Linked Using a Linker Command File	4-6
4-7	function.c	4-7
4-8	Screen Output After Compilation of function.c	4-7

Setting Up the Code Generation Tools With Windows 95 and Windows NT

This chapter helps you install release 1.00 of the TMS470R1x code generation tools and set up your code-development environment on a 32-bit x86-based or Pentium™ PC™ running Windows™ 95 or Windows NT™. These tools include an optimizing C/C++ compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C/C++ compiler tools include the following:

- Compiler
- Interlist utility
- Library-build utility
- C++ name demangling utility

The assembly language tools include the following:

- Assembler
- Archiver
- Linker
- Absolute lister
- Cross-reference lister
- Hex-conversion utility

Topic	Page
1.1 System Requirements	1-2
1.2 Installing the Code Generation Tools	1-3
1.3 Setting Up the Code Generation Environment	1-4
1.4 Where to Go From Here	1-8

1.1 System Requirements

To install and use the code generation tools, you need the items listed in the following hardware and software checklists.

Hardware checklist

- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | Host | 32-bit x86-based or Pentium-based PC with an ISA/EISA bus |
| <input type="checkbox"/> | Memory | Minimum of 16M bytes of RAM plus 32M bytes of hard-disk space for temporary files and 4M bytes of hard-disk space for the code generation tools |
| <input type="checkbox"/> | Display | Monochrome or color monitor (color recommended) |
| <input type="checkbox"/> | Required hardware | CD-ROM drive |
| <input type="checkbox"/> | Optional hardware | Microsoft™ compatible mouse |

Software checklist

- | | | |
|--------------------------|-------------------------|---|
| <input type="checkbox"/> | Operating system | One of the following: <ul style="list-style-type: none"><input type="checkbox"/> Windows 95 version 4.0 (or higher)<input type="checkbox"/> Windows NT Workstation version 4.0 |
| <input type="checkbox"/> | CD-ROMs | <i>TMS470R1x Code Generation Tools for C/C++</i> |

1.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The code generation tools package is shipped on CD-ROM. To install the tools on a PC running Windows 95 or Windows NT, follow these steps:

- 1) Insert the *TMS470R1x Code Generation Tools for C/C++* CD-ROM into your CD-ROM drive.
- 2) Start Windows.
- 3) Select Run from the Start menu.
- 4) In the dialog box, enter the following command (where d: is the name of your CD-ROM drive):
`d:\setup.exe`
- 5) Click on OK.
- 6) Follow the on-screen instructions.

If you choose not to have the environment variables set up automatically, you can set them up yourself in one of the following ways:

- If you are running Windows 95, you can set up the environment variables in your `autoexec.bat` file.
- If you are running Windows NT, you can set up the environment variables in the System applet of the control panel.

See Section 1.3, *Setting Up the Code Generation Environment*, on page 1-4, for more information.

1.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters that you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

By default, the installation program modifies your `autoexec.bat` file and sets up environment variables in the following manner:

```
set PATH=c:\tool_dir;%PATH%
set A_DIR=c:\tool_dir
set C_DIR=c:\tool_dir
```

These variables are set up in the registry under:

```
HKEY_CURRENT_USER\Environment
```

If you choose not to have the environment variables set up automatically, you can set them up yourself in one of the following ways:

- If you are running Windows 95, you can modify your `autoexec.bat` file to include the set commands above.
- If you are running Windows NT, you can set up the environment variables in the System applet of the control panel. Enter the same commands that you would enter on the command line in the System applet.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (PATH statement)

You must include the *tool_dir* directory in your PATH statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- You can change the path information in one of the following ways:
 - If you are running Windows 95, modify your autoexec.bat file to change the path information by adding the following to the end of the PATH statement:

```
;c:\tool_dir
```
 - If you are running Windows NT, modify the System applet of the Control Panel to change the path information by adding the following to the end of the PATH statement:

```
;c:\tool_dir
```

- If you set the PATH statement from the command line, enter the following:

```
set PATH=c:\tool_dir;%PATH%
```

The addition of `;%PATH%` ensures that this PATH statement does not undo the PATH statements in any other batch files (including the autoexec.bat file).

Identifying alternate directories for the assembler to search (A_DIR)

The assembler uses the A_DIR environment variable to name alternate directories for the assembler to search. To set the A_DIR environment variable, use this syntax:

```
set A_DIR=pathname1[;pathname2 . . .]
```

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or with a blank. Once you set A_DIR, you can use the .copy, .include, or .mlib directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the `-i` option (which names alternate directories), it searches the paths named by the A_DIR environment variable. For more information on the `-i` option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C/C++ Compiler User's Guide*.

Identifying alternate directories for the compiler and linker to search (C_DIR)

The compiler and linker use the C_DIR environment variable to name alternate directories that contain #include files and function libraries. To set the C_DIR environment variable, use this syntax:

```
set C_DIR=pathname1[:pathname2 . . .]
```

The *pathnames* are directories that contain #include files or libraries (such as stdio.h). You can separate the pathnames with a semicolon or with a blank. In C/C++ source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker default shell options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting up default options with the C_OPTION environment variable is useful when you want consecutive shell runs with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

To set the C_OPTION environment variable, use this syntax:

```
set C_OPTION=option1 [option2 . . .]
```

Environment variable options are specified in the same way and have the same meaning that they would if they were specified on the command line. For example, if you want to always run quietly (the -q option), enable C/C++ source interlisting (the -s option), and link (the -z option), set up the C_OPTION environment variable as follows:

```
set C_OPTION=-qs -z
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following -z on the command line or in C_OPTION are passed to the linker. This enables you to use the C_OPTION environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set -z in the environment variable and want to compile only, use the -c option of the shell. These additional examples assume C_OPTION is set as shown above:

```
clp470 *.c                ; compiles and links
clp470 -c *.c             ; only compiles
clp470 *.c -z lnk.cmd     ; compiles and links using a
                          ; command file
clp470 -c *.c -z lnk.cmd ; only compiles (-c overrides -z)
```

For more information about shell options, see the *TMS470R1x Optimizing C/C++ Compiler User's Guide*. For more information about linker options, see the *TMS470R1x Assembly Language Tools User's Guide*.

Specifying a temporary file directory (TMP)


The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the TMP environment variable.

Using the TMP environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory in which the source resides. This is useful for compiling from protected directories.

To set the TMP environment variable, use this syntax:

```
set TMP=pathname
```

For example, to set up a directory named temp for intermediate files on your hard drive, enter:


```
set TMP=c:\temp 
```

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

```
set variable_name=
```

For example, to reset the A_DIR environment variable, enter:

```
set A_DIR= 
```

Verifying that the environment variables are set

To verify that the environment variables are set, open a DOS box and enter:

```
set 
```

This command lists the path and environment variables and their current values.

1.4 Where to Go From Here

Your code generation tools are now installed on your Windows 95 or Windows NT system. Now you should read the following:

- Chapter 4, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- Chapter 5, *Release Notes*. This chapter describes the media contents and explains the C++ language features supported by release 1.00 of the code generation tools.

Setting Up the Code Generation Tools on a SPARCstation

This chapter helps you install release 1.00 of the TMS470R1x code generation tools and set up your code-development environment on a SPARCstation™ running SunOS™ version 5.5x or higher (also known as Solaris™ version 2.5x or higher). These tools include an optimizing C/C++ compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C/C++ compiler tools include the following:

- Compiler
- Interlist utility
- Library-build utility
- C++ name demangling utility

The assembly language tools include the following:

- Assembler
- Archiver
- Linker
- Absolute lister
- Cross-reference lister
- Hex-conversion utility

Topic	Page
2.1 System Requirements	2-2
2.2 Installing the Code Generation Tools	2-3
2.3 Setting Up the Code Generation Environment	2-5
2.4 Where to Go From Here	2-10

2.1 System Requirements

To install and use the code generation tools, you need the items in the following hardware and software checklists.

Hardware checklist

- | | | |
|--------------------------|--------------------------|--|
| <input type="checkbox"/> | Host | SPARCstation-compatible system with a SPARCstation class 2 or higher performance |
| <input type="checkbox"/> | Display | Monochrome or color monitor (color recommended) |
| <input type="checkbox"/> | Disk space | 4M bytes of disk space |
| <input type="checkbox"/> | Required hardware | CD-ROM drive |
| <input type="checkbox"/> | Optional hardware | Mouse |

Software checklist

- | | | |
|--------------------------|-------------------------|---|
| <input type="checkbox"/> | Operating system | SunOS version 5.5x or higher (also known as Solaris 2.5x or higher) |
| <input type="checkbox"/> | Root privileges | Root privileges to mount and unmount the CD-ROM |
| <input type="checkbox"/> | CD-ROMs | <i>TMS470R1x Code Generation Tools for C/C++</i> |

2.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The software package is shipped on a CD-ROM. To install the tools on a SPARCstation running SunOS or Solaris, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Note:

You *must* have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

Mounting the CD-ROM

The first step in installing the code generation tools is to mount the CD-ROM.

- 1) If your CD-ROM drive is already attached, load the CD-ROM into the drive and enter the following command from a command shell:

```
cd /cdrom/cdrom0/sunos
```

- 2) If you do not have a CD-ROM drive attached, you must shut down your system to the PROM level, attach the CD-ROM drive, and enter the following command:

```
boot -r
```

After you log into your system, load the CD-ROM into the drive and enter the following command from a command shell:

```
cd /cdrom/cdrom0/sunos
```

Copying the files

Be sure you are not logged on as root. After you mount the CD-ROM, you must create the directory that will contain the software tools and copy the software to that directory.

- 1) Create a tools directory on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/tool_dir
```

- 2) Copy the files from the CD-ROM to your hard-disk system:

```
cp -r * /your_pathname/tool_dir
```

Unmounting the CD-ROM

You must unmount the CD-ROM after copying the files. From a command shell, enter:

```
cd
```

```
eject
```


2.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

You can set up the environment variables on the command line or in your `.login` or `.cshrc` file (for C shells) or `.profile` file (for Bourne or Korn shells). To set up these environment variables in your system initialization file, enter the same commands that you would enter on the command line in the file.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- If you modify your *.cshrc* file (for C shells) or *.profile* file (for Bourne or Korn shells) to change the path information, add the following path to the end of the path statement:

```
/your_pathname / tool_dir
```

- If you set the path statement from the command line, use this format:

- For C shells:

```
set path=(/your_pathname / tool_dir $path)
```

- For Bourne or Korn shells:

```
PATH=/your_pathname / tool_dir $PATH
```

The addition of *\$path* or *\$PATH* ensures that this path statement does not undo the path statements in the *.cshrc* or *.profile* file.

Identifying alternate directories for the assembler to search (A_DIR)

The assembler uses the *A_DIR* environment variable to name alternate directories for the assembler to search. To set the *A_DIR* environment variable, use this syntax:

- For C shells:

```
setenv A_DIR "pathname1[:pathname2 . . .]"
```

- For Bourne or Korn shells:

```
A_DIR="pathname1[:pathname2 . . .]"  
export A_DIR
```

(Be sure to enclose the directory names in quotes.)

The *pathnames* are directories that contain *copy/include* files or macro libraries. You can separate the pathnames with a semicolon or a blank. Once you set *A_DIR*, you can use the *.copy*, *.include*, or *.mlib* directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the *-i* option (which names alternate directories), it searches the paths named by the *A_DIR* environment variable.

For more information on the `-i` option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C/C++ Compiler User's Guide*.

Identifying alternate directories for the compiler and linker to search (C_DIR)

The compiler and linker use the `C_DIR` environment variable to name alternate directories that contain `#include` files and function libraries. To set the `C_DIR` environment variable, use this syntax:

- For C shells:

```
setenv C_DIR "pathname1[:pathname2 . . .]"
```

- For Bourne or Korn shells:

```
C_DIR="pathname1[:pathname2 . . .]"  
export C_DIR
```

(Be sure to enclose the directory names in quotes.)

The *pathnames* are directories that contain `#include` files or libraries (such as `stdio.h`). You can separate pathnames with a semicolon or with blanks. In C/C++ source, you can use the `#include` directive without specifying path information. Instead, you can specify the path information with `C_DIR`.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker default shell options using the `C_OPTION` environment variable. If you do this, the shell uses the default options and/or input filenames that you name with `C_OPTION` every time you run the shell.

Setting up default options with the `C_OPTION` environment variable is useful when you want consecutive shell runs with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the `C_OPTION` environment variable and processes it.

To set the `C_OPTION` environment variable, use this syntax:

- For C shells:

```
setenv C_OPTION "option1 [option2 . . .]"
```

- For Bourne or Korn shells:

```
C_OPTION="option1 [option2 . . .]"  
export C_OPTION
```

(Be sure to enclose the options in quotes.)

Environment variable options are specified in the same way and have the same meaning that they would if they were specified on the command line. For example, if you want to always run quietly (the `-q` option), enable C/C++ source interlisting (the `-s` option), and link (the `-z` option), set up the `C_OPTION` environment variable as follows:

- For C shells:

```
setenv C_OPTION "-qs -z"
```
- For Bourne or Korn shells:

```
C_OPTION="-qs -z"  
export C_OPTION
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following `-z` on the command line or in `C_OPTION` are passed to the linker. This enables you to use the `C_OPTION` environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set `-z` in the environment variable and want to compile only, use the `-c` option of the shell. These examples assume `C_OPTION` is set as shown above:

```
clp470 *.c                ; compiles and links  
clp470 -c *.c             ; only compiles  
clp470 *.c -z lnk.cmd     ; compiles and links using a  
                           ; command file  
clp470 -c *.c -z lnk.cmd  ; only compiles (-c overrides -z)
```

For more information about shell options, see the *TMS470R1x Optimizing C/C++ Compiler User's Guide*. For more information about linker options, see the *TMS470R1x Assembly Language Tools User's Guide*.

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the `TMP` environment variable.

Using the `TMP` environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory in which the source resides. This is useful for compiling from protected directories.

To set the TMP environment variable, use this syntax:

- For C shells:
setenv TMP "pathname"
- For Bourne or Korn shells:
TMP="pathname"
export TMP



(Be sure to enclose the directory name in quotes.)

For example, to set up a directory named temp for intermediate files, enter:

- For C shells:
setenv TMP "/temp"
- For Bourne or Korn shells:
TMP="/temp"
export TMP



Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. Use one of the following commands to reread your system initialization file:



- For C shells:
source ~/.cshrc 
- For Bourne or Korn shells:
source ~/.profile 

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:


- For C shells:
unsetenv variable_name 
- For Bourne or Korn shells:
unset variable_name 

For example, to reset the A_DIR environment variable, enter one of these commands:

- For C shells:
unsetenv A_DIR 
- For Bourne or Korn shells:
unset A_DIR 

Verifying that the environment variables are set

To verify that the environment variables are set, enter:

```
set 
```

This command lists the path and environment variables and their current values.

2.4 Where to Go From Here

Your code generation tools are now installed. Now you should read the following:

- Chapter 4, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- Chapter 5, *Release Notes*. This chapter describes the media contents and explains the C++ language features supported by release 1.00 of the code generation tools.

Setting Up the Code Generation Tools on an HP Workstation

This chapter helps you install release 1.00 of the TMS470R1x code generation tools and set up your code-development environment on an HP 9000 Series 700™ PA-RISC™ computer with HP-UX™ 10.2x. These tools include an optimizing C/C++ compiler and a full set of assembly language tools for developing and manipulating assembly language and object (executable) code.

The C/C++ compiler tools include the following:

- Compiler
- Interlist utility
- Library-build utility
- C++ name demangling utility

The assembly language tools include the following:

- Assembler
- Archiver
- Linker
- Absolute lister
- Cross-reference lister
- Hex-conversion utility

Topic	Page
3.1 System Requirements	3-2
3.2 Installing the Code Generation Tools	3-3
3.3 Setting Up the Code Generation Environment	3-5
3.4 Where to Go From Here	3-10

3.1 System Requirements

To install and use the code generation tools, you need the items in the following hardware and software checklists.

Hardware checklist

- | | | |
|--------------------------|--------------------------|---|
| <input type="checkbox"/> | Host | An HP 9000 Series 700 PA-RISC computer |
| <input type="checkbox"/> | Display | Monochrome or color monitor (color recommended) |
| <input type="checkbox"/> | Disk space | 4M bytes of disk space |
| <input type="checkbox"/> | Required hardware | CD-ROM drive |
| <input type="checkbox"/> | Optional hardware | Mouse |

Software checklist

- | | | |
|--------------------------|-------------------------|--|
| <input type="checkbox"/> | Operating system | HP-UX 10.2x operating system |
| <input type="checkbox"/> | Root privileges | Root privileges to mount and unmount the CD-ROM |
| <input type="checkbox"/> | CD-ROMs | <i>TMS470R1x Code Generation Tools for C/C++</i> |

3.2 Installing the Code Generation Tools

This section helps you install the code generation tools on your hard-disk system. The software package is shipped on a CD-ROM. To install the tools on an HP workstation, you must mount the CD-ROM, copy the files to your system, and unmount the CD-ROM.

Note:

You *must* have root privileges to mount or unmount the CD-ROM. If you do not have root privileges, get help from your system administrator.

Mounting the CD-ROM

As root, you can mount the CD-ROM using the UNIX™ mount command or the SAM (system administration manager):

- To use the UNIX mount command, enter the following command from a command shell:

```
mount -F cdfs -o cdcase /dev/dsk/your_cdrom_device /cdrom
exit
```

Make the hp directory on the CD-ROM the current directory. For example, if the CD-ROM is mounted at /cdrom, enter:

```
cd /cdrom/hp
```

- To use SAM to mount the CD-ROM, see *System Administration Tasks*, the HP documentation about SAM, for instructions.

Copying the files

After you mount the CD-ROM, log out as root and log back on as yourself. You must create the directory that will contain the software tools and copy the software to that directory.

- 1) Create a tools directory on your hard disk. To create this directory, enter:

```
mkdir /your_pathname/tool_dir
```

- 2) Copy the files from the CD-ROM to your hard-disk system:

```
cp -r * /your_pathname/tool_dir
```

Setting up the software tools using a C shell

If you are using a C shell, enter the following commands:

```
setenv C_DIR "tool_dir"
setenv A_DIR "tool_dir"
set path=(tool_dir $path)
```

You can move the **setenv** and **set path** commands into your `.login` or `.cshrc` file to avoid entering these commands each time you invoke a new shell.

Setting up the software tools using a Korn shell

If you are using a Bourne or Korn shell, enter the following commands:

```
C_DIR=tool_dir
A_DIR=tool_dir
PATH=tool_dir:$PATH
```

You can move the environment variable instructions into your `.kshrc` file to avoid entering these commands each time you invoke a new shell.

Unmounting the CD-ROM

You must unmount the CD-ROM after copying the files. As root, enter the following commands from a command shell:

```
cd
umount /cdrom
exit
```

3.3 Setting Up the Code Generation Environment

Before or after you install the code generation tools, you can define environment variables that set certain software tool parameters you normally use. An *environment variable* is a special system symbol that you define and assign to a string. A program uses this symbol to find or obtain certain types of information.

When you use environment variables, default values are set, making each individual invocation of the tools simpler because these parameters are automatically specified. When you invoke a tool, you can use command-line options to override many of the defaults that are set with environment variables.

The code generation tools use the following environment variables:

- A_DIR
- C_DIR
- C_OPTION
- TMP

You can set up the environment variables on the command line or in your `.login` or `.cshrc` file (for C shells) or `.profile` file (for Bourne or Korn shells). To set up these environment variables in your system initialization file, enter the same commands that you would enter on the command line in the file.

In addition to setting up environment variables, you must modify your path statement. The following subsections describe how to modify your path statement and how to define the environment variables that the code generation tools use.

Identifying the directory that contains the executable files (path statement)

You must include the *tool_dir* directory in your path statement so that you can specify the assembler and compiler tools without specifying the name of the directory that contains the executable files.

- If you modify your *.cshrc* file (for C shells) or *.profile* file (for Bourne or Korn shells) to change the path information, add the following path to the end of the path statement:

Iyour_pathname/tool_dir

- If you set the path statement from the command line, use this format:

- For C shells:

set path=(*Iyour_pathname/tool_dir* \$path)

- For Bourne or Korn shells:

PATH=*Iyour_pathname/tool_dir* \$PATH

The addition of \$path or \$PATH ensures that this path statement does not undo the path statements in the *.cshrc* or *.profile* file.

Identifying alternate directories for the assembler (A_DIR)

The assembler uses the *A_DIR* environment variable to name alternate directories for the assembler to search. To set the *A_DIR* environment variable, use this syntax:

- For C shells:

setenv A_DIR "*pathname₁[:pathname₂ ...]*"

- For Bourne or Korn shells:

**A_DIR="*pathname₁[:pathname₂ ...]*"
export A_DIR**

(Be sure to enclose the directory names in quotes.)

The *pathnames* are directories that contain copy/include files or macro libraries. You can separate the pathnames with a semicolon or a blank. Once you set *A_DIR*, you can use the *.copy*, *.include*, or *.mlib* directive in assembly source without specifying path information.

If the assembler does not find the file in the directory that contains the current source file or in directories named by the *-i* option (which names alternate directories), it searches the paths named by the *A_DIR* environment variable. For more information on the *-i* option, see the *TMS470R1x Assembly Language Tools User's Guide* or the *TMS470R1x Optimizing C/C++ Compiler User's Guide*.

Identifying alternate directories for the compiler and linker (C_DIR)

The compiler and linker use the C_DIR environment variable to name alternate directories that contain #include files and libraries. To set the C_DIR environment variable, use this syntax:

- For C shells:

```
setenv C_DIR "pathname1[:pathname2 . . .]"
```

- For Bourne or Korn shells:

```
C_DIR="pathname1[:pathname2 . . .]"  
export C_DIR
```

(Be sure to enclose the directory names in quotes.)

The *pathnames* are directories that contain #include files or libraries (such as stdio.h). You can separate pathnames with a semicolon or with blanks. In C/C++ source, you can use the #include directive without specifying path information. Instead, you can specify the path information with C_DIR.

Setting default shell options (C_OPTION)

You may find it useful to set the compiler, assembler, and linker shell default options using the C_OPTION environment variable. If you do this, the shell uses the default options and/or input filenames that you name with C_OPTION every time you run the shell.

Setting up default options with the C_OPTION environment variable is useful when you want consecutive shell runs with the same set of options and/or input files. After the shell reads the command line and the input filenames, it reads the C_OPTION environment variable and processes it.

To set the C_OPTION environment variable, use this syntax:

- For C shells:

```
setenv C_OPTION "option1 [option2 . . .]"
```

- For Bourne or Korn shells:

```
C_OPTION="option1 [option2 . . .]"  
export C_OPTION
```

(Be sure to enclose the options in quotes.)

Environment variable options are specified in the same way and have the same meaning that they would if they were specified on the command line. For example, if you want to always run quietly (the `-q` option), enable C/C++ source interlisting (the `-s` option), and link (the `-z` option), set up the `C_OPTION` environment variable as follows:

- For C shells:

```
setenv C_OPTION "-qs -z"
```
- For Bourne or Korn shells:

```
C_OPTION="-qs -z"  
export C_OPTION
```

In the following examples, each time you run the compiler shell, it runs the linker. Any options following `-z` on the command line or in `C_OPTION` are passed to the linker. This enables you to use the `C_OPTION` environment variable to specify default compiler and linker options and then specify additional compiler and linker options on the shell command line. If you have set `-z` in the environment variable and want to compile only, use the `-c` option of the shell. These examples assume `C_OPTION` is set as shown above:

```
clp470 *.c                ; compiles and links  
clp470 -c *.c             ; only compiles  
clp470 *.c -z lnk.cmd     ; compiles and links using a  
                          ; command file  
clp470 -c *.c -z lnk.cmd ; only compiles (-c overrides -z)
```

For more information about shell options, see the *TMS470R1x Optimizing C/C++ Compiler User's Guide*. For more information on linker options, see the *TMS470R1x Assembly Language Tools User's Guide*.

Specifying a temporary file directory (TMP)

The compiler shell program creates intermediate files as it processes your program. By default, the shell puts intermediate files in the current directory. However, you can name a specific directory for temporary files by using the `TMP` environment variable.

Using the `TMP` environment variable allows use of a RAM disk or other file systems. It also allows you to compile source files from a remote directory without writing any files into the directory in which the source resides. This is useful for compiling from protected directories.

To set the TMP environment variable, use this syntax:

- For C shells:
setenv TMP "pathname"
- For Bourne or Korn shells:
TMP="pathname"
export TMP



(Be sure to enclose the directory name in quotes.)

For example, to set up a directory named temp for intermediate files, enter:

- For C shells:
setenv TMP "/temp"
- For Bourne or Korn shells:
TMP="/temp"
export TMP

Reinitializing your shell

When you modify your shell configuration file, you must ensure that the changes are made to your current session. Use one of the following commands to reread your system initialization file:

- For C shells:
source ~/.cshrc 
- For Bourne or Korn shells:
source ~/.profile 

Resetting defined environment variables

The environment variables that you define remain set until you reboot the system. If you want to clear an environment variable, use this command:

- For C shells:
unsetenv variable name
- For Bourne or Korn shells:
unset variable name

For example, to reset the A_DIR environment variable, enter one of these commands:

- For C shells:
unsetenv A_DIR
- For Bourne or Korn shells:
unset A_DIR

Verifying that the environment variables are set

To verify that the environment variables are set, enter:

```
set 
```

This command lists the path and environment variables and their current values.

3.4 Where to Go From Here

Your code generation tools are now installed. Now you should read the following:

- Chapter 4, *Getting Started With the Code Generation Tools*. This chapter provides you with an overview of how to invoke and use the assembler, linker, and compiler.
- Chapter 5, *Release Notes*. This chapter describes the media contents and explains the C++ language features supported by release 1.00 of the code generation tools.

Getting Started With the Code Generation Tools

This chapter helps you start using the assembler, linker, and compiler by providing a quick walkthrough of these tools. For more information about invoking and using these tools, see the *TMS470R1x Assembly Language Tools User's Guide* and the *TMS470R1x Optimizing C/C++ Compiler User's Guide*.

Topic	Page
4.1 Getting Started With the Assembler and Linker	4-2
4.2 Getting Started With the C/C++ Compiler	4-7

4.1 Getting Started With the Assembler and Linker

This section provides a quick walkthrough of the assembler and linker so that you can get started without reading the entire *TMS470R1x Assembly Language Tools User's Guide*.

- 1) Create two short source files to use for the walkthrough; call them file1.asm and file2.asm. (See Example 4–1 and Example 4–2.)

Example 4–1. file1.asm

```
        .global inclw

start:  MOV     r6, #0
        MOV     r7, #0

loop:   BL      inclw
        BCC     loop

        .end
```


Example 4–2. file2.asm

```
        .global inclw

inclw:  ADDS    r7, r7, #1
        ADDCSS r6, r6, #1
        MOV    pc, lr

        .end
```

- 2) Enter the following command to assemble file1.asm:

```
asm470 file1 
```

The **asm470** command invokes the assembler. The input source file is file1.asm. (If the input file extension is .asm, you do not have to specify the extension; the assembler uses .asm as the default.)

This example creates an object file called file1.obj. The assembler creates an object file only if there are no errors in assembly. You can specify a name for the object file, but if you do not, the assembler uses the input file-name with the extension .obj.

- 3) Now enter the following command to assemble file2.asm:

```
asm470 file2.asm -l
```

This time, the assembler creates an object file called file2.obj. The `-l` (lowercase L) option tells the assembler to create a listing file; the listing file for this example is called file2.lst. It is not necessary to create a listing file, but you can use the listing file to verify that the assembly has resulted in the desired object code. The listing file for this example is shown in Example 4–3.

Example 4–3. file2.lst, the Listing File Created by asm470 file2.asm -l

```

TMS470 COFF Assembler      Version 1.00      Sat Feb  8 15:22:13 1997
Copyright (c) 1996-1997    Texas Instruments Incorporated

file2.asm                                     PAGE      1

   1
   2                               .global inclw
   3
   4 00000000 E2977001  inclw:  ADDS    r7, r7, #1
   5 00000004 22966001      ADDCSS  r6, r6, #1
   6 00000008 E1A0F00E      MOV     pc, lr
   7
   8                               .end

No Errors, No Warnings

```

- 4) Now enter the following command to link file1.obj and file2.obj:

```
lnk470 file1 file2 -m lnker2.map -o prog.out
```

The **lnk470** command invokes the linker. The input object files are file1.obj and file2.obj. (If the input file extension is .obj, you do not have to specify the extension; the linker uses .obj as the default.) The linker combines file1.obj and file2.obj to create an executable object module called prog.out. The `-o` option supplies the name of the output module. Example 4–4 shows the map file resulting from this operation. (The map file is produced only if you use the `-m` option.)

Example 4-4. Output Map File, Inker2.map

```

*****
TMS470 COFF Linker          Version 1.00
*****
Sat Feb  8 15:24:43 1997

OUTPUT FILE NAME:  <prog.out>
ENTRY POINT SYMBOL: 0

SECTION ALLOCATION MAP

  output
section  page      origin      length      attributes/
-----  -
.text    0      00000000  0000001c   file1.obj (.text)
          00000000  00000010   file2.obj (.text)
          00000010  0000000c

.const   0      00000000  00000000   UNINITIALIZED

.data    0      00000000  00000000   UNINITIALIZED
          00000000  00000000   file2.obj (.data)
          00000000  00000000   file1.obj (.data)

.bss     0      00000000  00000000   UNINITIALIZED
          00000000  00000000   file2.obj (.bss)
          00000000  00000000   file1.obj (.bss)

GLOBAL SYMBOLS

address  name                address  name
-----  -
00000000 .bss                 00000000 .bss
00000000 .data                00000000 ___edata__
00000000 .text                00000000 edata
00000000 ___bss__     00000000 ___end__
00000000 ___data__    00000000 end
00000000 ___edata__   00000000 ___bss__
00000000 ___end__     00000000 ___data__
0000001c ___etext__   00000000 .text
00000000 ___text__   00000000 ___text__
00000000 edata        00000000 .data
00000000 end          00000010 inclw
0000001c etext        0000001c ___etext__
00000010 inclw        0000001c etext

[13symbols]

```

The two files, file1 and file2, can be linked together with or without a command file. However, using a command file allows you to configure your memory using the MEMORY and SECTIONS directives:

- ❑ The MEMORY directive lets you specify a model of target memory so that you can define the types of memory your system contains and the address ranges they occupy.
- ❑ The SECTIONS directive describes how input sections are combined into output sections and specifies where output sections are placed in memory.

You can include the linker options and filenames in the linker command file, or you can enter them on the command line. If you do not include a linker command file, the linker uses a default allocation algorithm. Refer to the *TMS470R1x Assembly Language Tools User's Guide* for more information about the linker command file and the default allocation algorithm.

Example 4–5. Sample Linker Command File, linker2.cmd

```

/* Specify the System Memory Map */


MEMORY
{
    D_MEM : org = 0x00000000   len = 0x00001000 /* Data Memory (RAM) */
    P_MEM : org = 0x00001000   len = 0x00001000 /* Program Memory (ROM) */
}

/* Specify the Sections Allocation Into Memory */

SECTIONS
{
    .data : {} > D_MEM /* Initialized Data */
    .text : {} > P_MEM /* Code */
}

```

Typing in the following command line using the linker command file shown in Example 4–5 results in the map file shown in Example 4–6.

```
lnk470 file1 file2 linker2.cmd -m linker2.map -o prog.out 
```

Example 4-6. Linker Map File (linker2.map) Linked Using a Linker Command File

```

*****
TMS470 COFF Linker          Version 1.00
*****
Sat Feb  8 15:36:45 1997

OUTPUT FILE NAME:  <prog.out>
ENTRY POINT SYMBOL: 0

MEMORY CONFIGURATION

      name      origin      length      used      attributes      fill
-----
D_MEM      00000000  000001000  00000000      RWIX
P_MEM      00001000  000001000  0000001c      RWIX

SECTION ALLOCATION MAP

  output
section  page      origin      length      attributes/
-----  -
.data    0      00000000  00000000  UNINITIALIZED
          00000000  00000000  file2.obj (.data)
          00000000  00000000  file1.obj (.data)

.text    0      00001000  0000001c
          00001000  00000010  file1.obj (.text)
          00001010  0000000c  file2.obj (.text)

.bss     0      00000000  00000000  UNINITIALIZED
          00000000  00000000  file2.obj (.bss)
          00000000  00000000  file1.obj (.bss)

GLOBAL SYMBOLS

address  name
-----  -
00000000 .bss
00000000 .data
00001000 .text
00000000 __bss__
00000000 __data__
00000000 __edata__
00000000 __end__
0000101c __etext__
00001000 __text__
00000000 edata
00000000 end
0000101c etext
00001010 inclw

address  name
-----  -
00000000 end
00000000 __edata__
00000000 .bss
00000000 __bss__
00000000 .data
00000000 __end__
00000000 edata
00000000 __data__
00001000 __text__
00001000 .text
00001010 inclw
0000101c __etext__
0000101c etext

[13 symbols]

```

4.2 Getting Started With the C/C++ Compiler

This section provides a quick walkthrough of the C/C++ compiler so that you can get started without reading the entire *TMS470R1x Optimizing C/C++ Compiler User's Guide*.

- 1) Create a sample file called `function.c` that contains the code in Example 4–7.

Example 4–7. `function.c`

```

/*****
/*      function.c      */
/* (Sample file for walkthrough) */
/*****
int main(int i)
{
    return(i < 0 ? -i : i );
}

```

- 2) To invoke the shell program to compile and assemble `function.c`, enter:

```
clp470 -o function.c
```

By default, the TMS470R1x shell program compiles and assembles 32-bit instructions. To compile 16-bit instructions, use the `-mt` option:

```
clp470 -o -mt function.c
```

The shell program prints the information shown in Example 4–8 as it compiles the program.

Example 4–8. Screen Output After Compilation of `function.c`

```

[function]
TMS470 ANSI C/C++ Compiler Version 1.00
Copyright (c) 1996-1997 Texas Instruments Incorporated
"function.c" ==> main
TMS470 ANSI C Optimizer Version 1.00
Copyright (c) 1996-1997 Texas Instruments Incorporated
"function.c" ==> main
TMS470 ANSI C/C++ Codegen Version 1.00
Copyright (c) 1996-1997 Texas Instruments Incorporated
"function.c": ==> main
TMS470 COFF Assembler Version 1.00
Copyright (c) 1996-1997 Texas Instruments Incorporated
PASS 1
PASS 1.1 ON SECTION .text
PASS 2

No Errors, No Warnings

```

By default, the shell deletes the assembly language file from the current directory after the file is assembled. If you want to inspect the assembly language output, use the `-k` option to retain the assembly language file:

```
clp470 -o -k function.c
```

- 3) Also by default, the shell creates a COFF object file as output; however, if you use the `-z` option, the output is an *executable* object module. The following examples show two ways of creating an executable object module:

- a) The command in step 2 creates an object file called `function.obj`. To create an executable object module, run the linker separately by invoking `lnk470`:

```
lnk470 -c function.obj lnk32.cmd -o function.out -l rtsc_32.lib
```

The `-c` linker option tells the linker to observe the C/C++ language linking conventions. The linker command file, `lnk32.cmd`, is shipped with the code generation tools. The `-o` option names the output module, `function.out`; if you do not use the `-o` option, the linker names the output module `a.out`. The `-l` option names the runtime-support library. You must have a runtime-support library before you can create an executable object module; the prebuilt runtime-support libraries `rtsc_32.lib` and `rtsc_16.lib` (C runtime library object) and `rtscpp_32.lib` and `rtscpp_16.lib` (C++ runtime library object) are included with the code generation tools.

- b) In this example, use the `-z` shell option, which tells the shell program to run the linker. The `-z` option is followed by linker options.

```
clp470 -o function.c -z lnk32.cmd -o function.out -l rtsc_32.lib
```

For more information on linker commands, see the *Linker Description* chapter of the *TMS470R1x Assembly Language Tools User's Guide*.

- 4) The TMS470R1x compiler package also includes an *interlist utility*. This program interlists the C/C++ source statements as comments in the assembly language compiler output, allowing you to inspect the assembly language generated for each line of C/C++. To run the interlist utility, invoke the shell program with the `-s` option. For example:

```
clp470 -s function.c -z lnk32.cmd -o function.out
```

The output of the interlist utility is written to the assembly language file created by the compiler. (The shell `-s` option implies `-k`; that is, when you use the interlist utility, the assembly file is automatically retained.)

Release Notes

This chapter describes the media contents of the TMS470R1x tools kit. The tools are supported on SPARCstations, HP workstations, and PCs with Windows 95 or Windows NT.

This chapter also documents the C++ language features that are supported by the '470 C/C++ compiler in this release.

Topic	Page
5.1 Media Contents	5-2
5.2 C++ Language Support	5-6

5.1 Media Contents

The CD-ROM included in the TMS470R1x tools kit for SPARCstations and HP workstations contains the files listed in Table 5–1. The CD-ROM included in the TMS470R1x tools kit for PCs contains the files listed in Table 5–2.

Table 5–1. Media Contents for SPARCstations and HP Workstations

File	Description
README.1ST	Online release bulletin
abs470	Absolute lister
acp470	C/C++ parser
ar470	Archiver
asm470	Assembler
cg470	Code generator
clist	C/C++ source interlist utility
clp470	Compiler shell program
dem470	C++ name demangling utility
hex470	Hex-conversion utility
intvecs.asm	Sample interrupt vector setup file
lnk470	COFF linker
lnk16.cmd	Sample 16-bit linker command file
lnk32.cmd	Sample 32-bit linker command file
mkp470	Library-build utility
opt470	C/C++ optimizer
rtsc_16.lib	16-bit C runtime-support library
rtsc_32.lib	32-bit C runtime-support library
rtscpp_16.lib	16-bit C++ runtime-support library
rtscpp_32.lib	32-bit C++ runtime-support library
rtsc.src	C runtime-support source library
rtscpp.src	C++ runtime-support source library

Table 5–1. Media Contents for SPARCstations and HP Workstations (Continued)

File	Description
xref470	Cross-reference utility
*.h	#include header files for ANSI C runtime support:
	assert.h limits.h stdarg.h stdlib.h
	ctype.h math.h stddef.h string.h
	errno.h setjmp.h stdio.h time.h
	float.h
	#include header files for C++ runtime support:
	cassert cmath cstdlib stdexcept
	cctype csetjmp cstring typeinfo
	cerrno cstdarg ctime
	cfloat cstddef exception
	climits cstdio new

Table 5–2. Media Contents for PCs

File	Description
readme.1st	Online release bulletin
abs470.exe	Absolute lister
acp470.exe	C/C++ parser
ar470.exe	Archiver
asm470.exe	Assembler
cg470.exe	Code generator
clp470.exe	Compiler shell program
clist.exe	C/C++ source interlist utility
dem470.exe	C++ name demangling utility
hex470.exe	Hex-conversion utility
intvecs.asm	Sample interrupt vector setup file
lnk470.exe	COFF linker
lnk16.cmd	Sample 16-bit linker command file
lnk32.cmd	Sample 32-bit linker command file
mkp470.exe	Library-build utility
opt470.exe	C/C++ optimizer
rtsc_16.lib	16-bit runtime-support library
rtsc_32.lib	32-bit runtime-support library
rtsc.src	C runtime-support source library
rtscpp.src	C++ runtime-support source library
rtscpp_16.lib	16-bit C++ runtime-support library
rtscpp_32.lib	32-bit C++ runtime-support library
xref470.exe	Cross-reference utility

Table 5–2. Media Contents for PCs (Continued)

File	Description
*.h	#include header files for ANSI C RTS: assert.h limits.h stdarg.h stdlib.h ctype.h math.h stddef.h string.h errno.h setjmp.h stdio.h time.h float.h #include header files for C++ runtime support: cassert cmath cstdlib stdexcept cctype csetjmp cstring typeinfo cerrno cstdarg ctime cfloat cstdef exception climits cstdio new

5.2 C++ Language Support

The '470 C/C++ compiler for release 1.00 of the code generation tools supports C++ as defined by Ellis and Stroustrup's *Annotated C++ Reference Manual* (ARM) except that it does not support exception handling.

Notes:

- 1) See the *TMS470R1x Optimizing C/C++ Compiler User's Guide* for a description of the C language features supported in this release.
- 2) The lists of supported and unsupported C++ language features given in this section are subject to change as the product matures and as the C++ standard comes into existence.

Additional C++ language features supported

In addition to supporting C++ as defined in the ARM, the compiler accepts the following features, which are defined in the X3J16/WG21 Working Paper:

- The dependent statement of an if, while, do-while, or for loop is considered to be a scope, and the restriction on having such a dependent statement be a declaration is removed.
- The expression tested in an if, while, do-while, or for loop as the first operand of a ? operator or as an operand of the &&, ||, or ! operator can have a pointer-to-member type or a class type that can be converted to a pointer-to-member type in addition to the scalar cases permitted by the ARM.
- Qualified names are allowed in elaborated type specifiers.
- The global-scope qualifier is allowed in member references of the form x.: :A: :B and p->: :A: :B.
- The precedence of the third operand of the ? operator is changed.
- If control reaches the end of the main() routine and main() has an integral return type, it is treated as if a return 0; statement were executed.
- Pointers to arrays with unknown bounds as parameter types are diagnosed as errors.
- A functional-notation cast of the form A() can be used even if A is a class without a (nontrivial) constructor. The temporary copy that is created gets the same default initialization to 0 as a static object of the class type.
- A cast can be used to select one out of a set of overloaded functions when taking the address of a function.

- Template friend declarations and definitions are permitted in class definitions and class template definitions.
- Type template parameters are permitted to have default arguments.
- Function templates may have nontype template parameters.
- A reference to const volatile cannot be bound to an rvalue.
- Qualification conversions such as conversion from T** to T const * const * are allowed.
- Digraphs are recognized.
- Operator keywords (and, bitand, etc.) are recognized.
- Static data member declarations can be used to declare member constants.
- wchar_t* is recognized as a keyword and a distinct type.
- bool* is recognized as a keyword and a distinct type.
- RTTI (runtime type identification), including *dynamic_cast* and the *typeid* operator, is implemented.
- Declarations in tested conditions (in *if*, *switch*, *for*, and *while* statements) are supported.
- Array *new* and *delete* are implemented.
- New-style casts (*static_cast*, *reinterpret_cast*, and *const_cast*) are implemented.
- The definition of a nested class outside its enclosing class is allowed.
- mutable* is accepted on nonstatic data member declarations.
- Namespaces are implemented, including *using* declarations and directives. Access declarations are broadened to match the corresponding *using* declarations.
- Explicit instantiation of templates is implemented.
- The *typename* keyword is recognized.

Unsupported C++ language features defined in the X3J16/WG21 Working Paper

The following features, which are not defined in the ARM but are defined in the X3J16/WG21 Working Paper, are not supported:

- C++ standard library support is not included. (C subset and basic language support is included.)
- Virtual functions in derived classes may not return a type that is the derived-class version of the type returned by the overridden function in the base class.
- enum* types are not considered to be nonintegral types.
- Operators cannot be overloaded using functions that take enum types and no class types.
- The new lookup rules for member references of the form `x.A: :B` and `p->A: :B` are not implemented.
- Classes are not assumed to always have constructors, and the distinction between trivial and nontrivial constructors is not implemented.
- enum* types cannot contain values larger than those that can be contained in an `int`.
- Type qualifiers are not retained on rvalues (in particular, on function return values).
- reinterpret_cast* does not allow casting a pointer to member of one class to pointer to member of another class if the classes are unrelated.
- Explicit qualification of template functions is not implemented.
- Member templates are not implemented.
- Name binding in templates in the style of N0288/03-0081 is not implemented.
- The scope of a variable declared in a for loop is still the whole surrounding scope, not just the loop.
- As required by the ARM, in a reference of the form `f()->g()`, with `g` equal to a static member function, `f()` is not evaluated. (The Working Paper, however, does require that `f()` be evaluated.)
- `(p->*pm) = 0` cannot be written as `p->*pm=0`. (The syntax still conforms to the ARM.)

- Nonconverting constructors are not implemented.
- Class name injection is not implemented.
- Overloading of function templates (partial specialization) is not implemented.
- Partial specialization of class templates is not implemented.
- Placement delete is not implemented.
- The notation `:: template` (and `->template`, etc.) is not implemented.

Glossary

A

asm470: The name of the command that invokes the assembler for the TMS470R1x.

assembler: A software program that creates a machine-language program from a source file that contains assembly-language instructions, directives, and macro definitions. The assembler substitutes absolute operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

C

C/C++ compiler: A software program that translates C/C++ source statements into assembly-language source statements.

code generator: A compiler tool that takes the file produced by the parser or the optimizer and produces an assembly-language source file.

COFF: *Common object file format.* A binary object file format that promotes modular programming by supporting the concept of *sections*. All COFF sections are independently relocatable in memory space; you can place any section into any allocated block of target memory.

clp470: The name of the command that invokes the compiler shell program for the TMS470R1x. (The second character in the shell name is a lower-case L.)

E

environment variables: System symbols that you define and assign to a string. They are usually included in batch files (for example, `.cshrc`).

I

interlist utility: A compiler utility that inserts your original C/C++ source statements as comments in the assembly language output from the assembler. The C/C++ statements are inserted next to the equivalent assembly instructions.

L

linker: A software program that combines object files to form an object module that can be allocated into system memory and executed by the device.

lnk470: The name of the command that invokes the linker for the TMS470R1x.

O

optimizer: A software tool that improves the execution speed and reduces the size of C/C++ programs.

options: Command parameters that allow you to request additional or specific functions when you invoke a software tool.

P

pragma: A preprocessor directive that provides directions to the compiler about how to treat a particular statement.

S

shell program: A utility that lets you compile, assemble, and optionally link in one step. The shell runs one or more source modules through the compiler (including the parser, optimizer, and code generator), the assembler, and the linker.

structure: A collection of one or more variables grouped together under a single name.

swap file: The file in which virtual memory (secondary memory) is allocated on the hard disk.

V

veneer: A sequence of instructions that serves as an alternate entry point into a routine if a state change is required.

virtual memory: The ability of a program to use more memory than the computer actually has available as RAM. This is accomplished by using a swap file on disk to augment RAM. When RAM is not sufficient, part of the program is swapped out to a disk file until it is needed again. The combination of the swap file and available RAM is the virtual memory.

Index

A

- A_DIR environment variable
 - for HP workstations 3-6
 - for SPARCstations 2-6 to 2-7
 - for Windows 95/NT systems 1-5
- asm470
 - definition A-1
 - invoking 4-2
- assembler
 - definition A-1
 - walkthrough 4-2 to 4-4
- assistance from TI vi

C

- C++ language support 5-6 to 5-10
- C/C++ compiler walkthrough 4-7 to 4-8
- C_DIR environment variable
 - for HP workstations 3-7
 - for SPARCstations 2-7
 - for Windows 95/NT systems 1-6
- C_OPTION environment variable
 - for HP workstations 3-7 to 3-8
 - for SPARCstations 2-7 to 2-8
 - for Windows 95/NT systems 1-6
- CD-ROM
 - mounting
 - for HP workstations 3-3
 - for SPARCstations 2-3
 - requirements
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
 - retrieving files from (copying)
 - for HP workstations 3-4
 - for SPARCstations 2-4

- CD-ROM (continued)
 - unmounting
 - for HP workstations 3-4
 - for SPARCstations 2-4
- clp470
 - definition A-1
 - invoking 4-7
- code generation tools
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
- code generator, definition A-1
- COFF
 - creating object file 4-8
 - definition A-1
- compiler, definition A-1
- conventions, notational iii to iv
- .cshrc file
 - for HP workstations 3-5 to 3-10
 - for SPARCstations 2-5 to 2-10

D

- digraphs 5-7
- directives
 - MEMORY 4-5
 - SECTIONS 4-5
- directories, software tools
 - for HP workstations 3-3, 3-6, 3-7
 - for SPARCstations 2-4, 2-6, 2-7
 - for Windows 95/NT systems 1-5, 1-6
- disk space requirements
 - for HP workstations 3-2
 - for SPARCstations 2-2
- display requirements
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
- documentation, related iv to v

E

- environment setup
 - for HP workstations 3-5 to 3-10
 - for SPARCstations 2-5 to 2-10
 - for Windows 95/NT systems 1-4 to 1-7
- environment variables
 - A_DIR 1-5, 2-6 to 2-7, 3-6
 - C_DIR 1-6, 2-7, 3-7
 - C_OPTION 1-6, 2-7 to 2-8, 3-7 to 3-8
 - definition A-1
 - for HP workstations 3-5 to 3-10
 - for SPARCstations 2-5 to 2-10
 - for Windows 95/NT systems 1-4 to 1-7
 - resetting 1-7, 2-9, 3-9
 - TMP 1-7, 2-8 to 2-9, 3-8 to 3-9
 - verifying 1-7, 2-10, 3-10
- example
 - assembler 4-2 to 4-3
 - compiler 4-7 to 4-8
 - linker 4-4 to 4-6

H

- hardware checklist
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
- host system
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
- HP workstations
 - requirements 3-2
 - setting up the environment 3-5 to 3-10
 - software installation 3-3 to 3-4

I

- installation, software
 - for HP workstations 3-3 to 3-4
 - for SPARCstations 2-3 to 2-4
 - for Windows 95/NT systems 1-3
- interlist utility
 - definition A-2
 - described 4-8

Index-2

- invoking
 - assembler 4-2 to 4-4
 - compiler 4-7 to 4-8
 - linker 4-2 to 4-6

L

- linker
 - definition A-2
 - walkthrough 4-2 to 4-4
- Ink470
 - definition A-2
 - invoking 4-3

M

- media contents
 - HP workstations 5-2
 - PCs with DOS 5-4
 - PCs with Windows 3.1x 5-4
 - PCs with Windows 95/NT 5-4
 - SPARCstations 5-2
- MEMORY directive 4-5
- memory requirements, for Windows 95/NT systems 1-2
- modifying PATH statement
 - for HP workstations 3-6
 - for SPARCstations 2-6
 - for Windows 95/NT systems 1-5
- mounting CD-ROM
 - for HP workstations 3-3
 - for SPARCstations 2-3
- mouse requirements
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2

N

- notational conventions iii to iv

O

- operating system
 - for HP workstations 3-2
 - for SPARCstations 2-2
 - for Windows 95/NT systems 1-2
- optimizer, definition A-2
- options, definition A-2

P

PATH statement
 for HP workstations 3-6
 for SPARCstations 2-6
 for Windows 95/NT systems 1-5

permissions
 for HP workstations 3-2
 for SPARCstations 2-2

pragma, definition A-2

.profile file
 for HP workstations 3-5 to 3-10
 for SPARCstations 2-5 to 2-10

R

reinitializing
 .cshrc file 2-9, 3-9
 .profile file 2-9, 3-9
 shell 2-9, 3-9

related documentation iv to v

requirements. *See* hardware checklist; software checklist

resetting environment variables
 for HP workstations 3-9
 for SPARCstations 2-9
 for Windows 95/NT systems 1-7

retrieving files from CD-ROM
 for HP workstations 3-4
 for SPARCstations 2-4

root privileges
 for HP workstations 3-2
 for SPARCstations 2-2

runtime-support library 4-8

S

SECTIONS directive 4-5

setting up the environment
 for HP workstations 3-5 to 3-10
 for SPARCstations 2-5 to 2-10
 for Windows 95/NT systems 1-4 to 1-7

setup.exe, for Windows 95/NT systems 1-3

shell, reinitializing 2-9, 3-9

shell program

COFF file 4-8
 definition A-2

software checklist

for HP workstations 3-2
 for SPARCstations 2-2
 for Windows 95/NT systems 1-2

Solaris. *See* SPARCstations

SPARCstations

requirements 2-2
 setting up the environment 2-5 to 2-10
 software installation 2-3 to 2-4

structure, definition A-2

SunOS. *See* SPARCstations

supported C++ language features 5-6 to 5-7

swap file, definition A-2

system initialization files

.cshrc
for HP workstations 3-5 to 3-10
for SPARCstations 2-5 to 2-10

.profile
for HP workstations 3-5 to 3-10
for SPARCstations 2-5 to 2-10

system requirements. *See* hardware checklist; software checklist

T

TMP environment variable

for HP workstations 3-8 to 3-9
 for SPARCstations 2-8 to 2-9
 for Windows 95/NT systems 1-7

U

unmounting CD-ROM

for HP workstations 3-4
 for SPARCstations 2-4

unsupported C++ language features 5-8 to 5-9

V

vener, definition A-3

virtual memory, definition A-3

W

walkthrough

assembler 4-2 to 4-4

C/C++ compiler 4-7 to 4-8

linker 4-2 to 4-4

Windows 95/NT systems

requirements 1-2

setting up the environment 1-4 to 1-7

software installation 1-3

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.