Chapter 1

C cross referencing tool "cxref"

A program that can automatically generate documentation and cross references for a C program. The input is any C program with appropriate comments and the output is LaTeXor HTML files.

1.1 Program Options

The name of the program is cxref.

Usage: `cxref`

```
          filename [ ... filename]
          [-Idirname]
          [-Ddefine] [-Udefine]
          [-CPP cpp_program]
          [-Odirname]
          [-Nbasename]
          [-all-comments]
          [-xref[-all][-file][-func][-var][-type]]
          [-warn[-all][-comment][-xref]]
          [-index[-all][-file][-func][-var][-type]]
          [-raw]
          [-latex|-latex2e]
          [-html]
```

`filename`	The name of the file to document, any number of files may be documented at a time.
`-Idirname`	GCC option to specify the path for include files.
`-Ddefine`	GCC option to define a pre-processor symbol.
`-Udefine`	GCC option to undefine a pre-processor symbol.
`-CPP program`	The name of the program to use instead of the compile time default. The program must be able to perform all of the actions that 'gcc -E -C -dD' does to work. If the program takes arguments then the whole thing needs to be in quotes so that it is interpreted as a single argument to cxref.
`-Odirname`	The name of a directory to use for the output LaTeXfiles and the location of the cross reference files that are created.
`-Nbasename`	The name to use for the first part of the output and cross reference files instead of cxref, the file extensions remain the same.
`-all-comments`	In case you are lazy and think that the existing comments might work, (see below for description of special comments). *[Danger! This option can produce weird results.]*
`-xref`	Produce cross referencing information (see below).

`-all`	All cross references.
`-file`	Cross references for files.
`-func`	Cross references for functions.
`-var`	Cross references for variables.
`-type`	Cross references for types.

`-warn`	Produce warnings, the options must be concatenated together:

`-all` All warnings.

`-comment` Warn of missing comments.

`-xref` Warn of missing cross references.

`-index` Produce a cross reference index, the options must be concatenated together:

`-all` All indexes.

`-file` Index of files.

`-func` Index of functions.

`-var` Index of variables.

`-type` Index of types.

`-raw` Produce a raw form of output, not really of much use except with -warn.

`-latex` Produce a LaTeXfile to document each of the source files and also an extra file that includes each of these files.

`-latex2e` Produce the LaTeX file for use with the LaTeX2e version.

`-html` Produce an HTML file to document each of the source files and a main file to reference each of these files.

1.2 Program Documentation Comments

The documentation for the program is produced from comments in the code that are appropriatly formatted. The cross referencing comes from the code itself and requires no extra work.
The special comments are "/**** ****/" (for a file) and "/*++++ ++++*/" (for a data object) any number of "*" or "+" can be used inside of the standard "/*" and "*/" comment delimiters in the comments, they are ignored.
In any situation where a comment follows a ",", ";" or ")" separated only by spaces and tabs, the comment is pushed to before the punctuation to apply to the object there.
The program is implemented using a full ANSI C grammar parser with some GCC extensions, this means that the style of the code is unimportant, only the content and comments.

1.3 Automated Comment Insertion

To simplify the insertion of comments that will be parsed by cxref, the file cxref.el provides a number of Emacs lisp functions. To use them add the line `(load "cxref")` to your ".emacs" file or type `M-x load-file cxref.el` from within Emacs.
The functions and key bindings are:

Control-C Control-F Adds file comments, a /** **/ header at the top of the file and if it is a .h file then it also adds a #ifndef, #define at the beginning and #endif at the end to stop multiple inclusions.

Control-C f Adds comments to a function, the cursor must be on the line containing the start of the function definition when this function is called. The /*+ ... +*/ comment that is added is of the header type (see the examples) not inline.

Control-C v Adds a leading comment to the variable or other definition on the current line.

Control-C e Adds a trailing comment at the end of the line.

Control-C i Adds an inline comment that is ignored by cxref.

1.4 C Preprocessor

To improve the output that is available a modified version of the GNU CPP V2.6.3 is supplied (named cxref-cpp).
Some comment styles are only available if cxref-cpp is used, see the "Example Special Comments" section of this README.
The #include filenames are also full pathnames if cxref-cpp is not used.
To enable this option, the Makefile for cxref must be modified when cxref is compiled, the variable CPP_TO_USE must be set to point at the place where the C preprocessor is to be installed. If CP-P_TO_USE is defined to an empty value then "gcc -E -C -dD" is used as the C preprocessor and the extra features are not available. To see which option is enabled, run cxref with no arguments, this prints the name of the preprocessor that is used, if the command is "cxref-cpp" then the option is enabled.

1.5 Example Special Comments

The file "README.c" is included in this document to show that the comments are indeed seen in the code, the result of running this through cxref is also included in this document.
The comments only available with the cxref-cpp pre-processor are indicated.

1.6 Cross Referencing

The cross referencing is performed for the following items

Files
- The files that the current file is included in (even when included via other files).

#includes
- Files included in the current file.
- Files included by these files etc.

Variables
- The location of the definition of external variables.
- The files that have visibility of global variables.
- The files / functions that use the variable.

Functions
- The file that the function is prototyped in.
- The functions that the function calls.
- The functions that call the function.
- The files and functions that reference the function.
- The variables that are used in the function.

Each of these items is cross referenced in the output.
The cross referencing uses files 'cxref.variable', 'cxref.function', 'cxref.include' and 'cxref.typedef' in the output directory.
These are a complete list of the function and variable usage in the program and could be used to generate a function call hierarchy or variable usage diagram for example.
Two cxref passes of each file is needed, the first to build up the cross referencing files and the second to use them.
(The file names are different if the "-N" option is used.)

1.7 LaTeXOutput

The default LaTeXoutput is a file for each of the source files with one extra file "cxref.tex" that includes each of the other files. This is to allow a makefile to only update the changed files (although the references may require all of the files to be checked again). When the cxref.tex file has been written it can be modified by the user, any new files that are added are added at the end before the table of contents, the rest of the file being unchanged.
The index is written to a file called "cxref.apdx.tex" and cxref.tex is updated to refer to it.
Also written out are two LaTeXstyle files "page.sty" and "fonts.sty", these use a smaller margin and smaller font to allow more to appear on a page.
(The file names "cxref.tex" and "cxref.apdx.tex" are different if the "-N" option is used.)

1.8 HTML Output

The default HTML output is a file for each of the source files with one extra file "cxref.html" that includes each of the other files. This is to allow a makefile to only update the changed files (although the references may require all of the files to be checked again). When the cxref.html file has been written it can be modified by the user, any new files that are added are added at the end before the table of contents, the rest of the file being unchanged.
The index is written to a file called "cxref.apdx.html" and cxref.html is updated to refer to it.
(The file names "cxref.html" and "cxref.apdx.html" are different if the "-N" option is used.)

1.9 AUTHOR and Copyright

The cxref program was written by Andrew M. Bishop in 1995,96.
The cxref program is copyright Andrew M. Bishop 1995,96.
email: amb@gedanken.demon.co.uk [Please put cxref in the subject line]
The cxref program can be freely distributed according to the terms of the GNU General Public License (see the file "COPYING").

Chapter 2

Example comments, “README.c”

```
/*****************
  $Header: /home/amb/cxref/RCS/README.c 1.2 1995/07/30 17:16:07 amb Exp $

  A comment for the file, RCS header comments are treated specially when first.
  *****************/



/*+ A #include comment +*/
#include <stdio.h>


#include <math.h>  /*+ An alternative #include comment. +*/

/* Note: The above comment is ignored without cxref-cpp, see README. */



/*+ A #define comment. +*/
#define def1 1


#define def2 2  /*+ An alternative #define comment. +*/

/* Note: The above comment is ignored without cxref-cpp, see README. */

/*++++++++++++
  A #define with args

  arg1 The first arg

  arg2 The second arg
  ++++++++++++*/

#define def3(arg1,arg2) (arg1+arg2)


/*+ An alternative #define with args. +*/

#define def4(arg1 /*+The first arg+*/,  \
             arg2 /*+The second arg+*/) \
           (arg1+arg2)

/* Note: the above works with cxref-cpp only, see README. */



/*+ An example typedef comment +*/
typedef enum
{
 one,            /*+ one value +*/
 two             /*+ another value +*/
}
```

```
type1;


/*+ Another example typedef comment, +*/
typedef struct
{
 int a;         /*+ A variable in a struct. +*/
 union bar
  {
   void a;      /*+ Each element +*/
   int  b,      /*+ of a struct +*/
        c;      /*+ or a union +*/
   long d;      /*+ can have a comment +*/
  }
 e;             /*+ Nested structs and unions also work. +*/
}
type2,          /*+ a type that is a struct. +*/
*ptype2;        /*+ a pointer to a struct type. +*/



/*+ A leading comment only. +*/
int var1,var2;


int var3; /*+ A trailing comment only. +*/


/*+ A variable for +*/
int var4,    /*+ one thing. +*/
    var5,    /*+ a second thing. +*/
    var6;    /*+ a third thing. +*/

/* Note: The leading comment is combined with each of the trailing comments. */
/* Note: the push through of the comment above on the ',' and ';', see README. */



/*++++++++++++
  A function comment (the comments for the args need to be separated by a blank line).

  int function1 The return value.

  int arg1 The first argument.

  int arg2 The second argument.

  Some more comment
  ++++++++++++*/

int function1(int arg1,int arg2)
{
 /*+ An internal comment in a function that appears as a
   new paragraph at the end of the comment. +*/
}


/*+ An alternative function comment +*/

int function2(int arg1,  /*+ The first argument.  +*/
              int arg2)  /*+ The second argument. +*/
/*+ Returns a value +*/
{
}

/* Note: the push through of the comment above on the ',' and ')', see README. */
```

Chapter 3

cxref output for "README.c"

3.1 File README.c

RCS Header: /home/amb/cxref/RCS/README.c 1.2 1995/07/30 17:16:07 amb Exp
A comment for the file, RCS header comments are treated specially when first.

Included Files

A #include comment
`#include <stdio.h>`

An alternative #include comment.
`#include <math.h>`

Preprocessor definitions

A #define comment.
`#define def1 1`

An alternative #define comment.
`#define def2 2`

A #define with args
`#define def3( arg1, arg2 )`

· `arg1`	The first arg
· `arg2`	The second arg

An alternative #define with args.
`#define def4( arg1, arg2 )`

· `arg1`	The first arg
· `arg2`	The second arg

3.1.1 Type definitions

3.1.1.1 Typedef type1

An example typedef comment
`typedef enum {...} type1`

```
enum
  {
   one                                     one value
   two                                     another value
  }
```

3.1.1.2 Type union bar

Nested structs and unions also work.

```
union bar
  {
    void a                                   Each element
    int b                                    of a struct
    int c                                    or a union
    long d                                   can have a comment
  }
```

3.1.1.3 Typedef type2

Another example typedef comment, a type that is a struct.

`typedef struct {...} type2`

```
struct
  {
    int a                                    A variable in a struct.
    union bar                                Nested structs and unions also work.
      {
        void a                               Each element
        int b                                of a struct
        int c                                or a union
        long d                               can have a comment
      }
      e
  }
```

3.1.1.4 Typedef ptype2

Another example typedef comment, a pointer to a struct type.

`typedef struct {...}* ptype2`

See: Typedef type2 *(Section 3.1.1.3)*

3.1.2 Variables

3.1.2.1 Variable var1

A leading comment only.

`int var1`

3.1.2.2 Variable var2

A leading comment only.

`int var2`

3.1.2.3 Variable var3

A trailing comment only.

`int var3`

3.1.2.4 Variable var4

A variable for one thing.

`int var4`

3.1.2.5 Variable var5

A variable for a second thing.

`int var5`

3.1.2.6 Variable var6

A variable for a third thing.

`int var6`

3.1.3 Functions

3.1.3.1 Global Function function1()

A function comment (the comments for the args need to be separated by a blank line).

`int function1 ( int arg1, int arg2 )`

· `int function1`	The return value.
· `int arg1`	The first argument.
· `int arg2`	The second argument.

Some more comment
An internal comment in a function that appears as a new paragraph at the end of the comment.

3.1.3.2 Global Function function2()

An alternative function comment

`int function2 ( int arg1, int arg2 )`

· `int function2`	Returns a value
· `int arg1`	The first argument.
· `int arg2`	The second argument.

Contents