Seminar Report

WEB/CWEB

Julian Lang

16 February 2013

Supervisor: Julian Nagele

Abstract

This report shorty explains the paradigm of literate programming and tries to give a overview over the first literate programming languages WEB and CWEB. The operating principle of the CWEB system will be explained and a short introduction of programming in CWEB will be given. At the end the influences of WEB/CWEB are described and the languages will be compared with more famous programming/documentation languages.

Contents

1 Introduction

In order to get good programs, a good documentation is essential for every computer scientist. In this report the paradigm of literate programming which was invented by Donald E. Knuth will be explained. Also the first literate programming languages WEB and CWEB are explained, which should help to make a better documentation for programs. The report is structured as follows. Section 2 describes the paradigm of literate programming and the basic structure of a literate program. Section 3 explains the history and the basic idea of the programming languages WEB and CWEB. In Section 4 the working principles of CWEB are presented, which will then be advanced in Section 6. In Section 5 an introduction of programming in CWEB will be given. Finally in section 7 the languages WEB and CWEB will be compared with other programming languages and some influences of the programming languages are described. This report is aimed at readers who have a basic knowledge of programming in C and TEX.

2 Literate Programming

Donald Knuth was convinced that a computer program should primarily be written for humans and not for computers. In his opinion a computer program should more be like a work of literature that is written to be easily understandable [3]. Instead of explaining the program in the order that is needed by the computer the program should be explained in the order of the thoughts of the programmer. So after he began to create his famous documentation language TEX he also created the first literate programming language with the name WEB. A literate programming language is the combination of a ordinary programming language and a document formatting language. In such a programming language the code is often divided in two different segments:

1. A coding language that is used to compile the program and execute it on the computer.
2. A document formatting language where the program or a segment of the code is documented in a format that a human can easily read.

A literate programming language has the advantage that the documentation and the code of a program can be written in the same file without loosing the advantages of a professional documentation or coding system. The code and the documentation can be directly linked such that changes in the code can easily be reflected in the documentation. A single source file can then be used to generate the executable file and also the documentation of the program. This kind of programming language should encourage the author of the program to write a better documentation. A very important feature of literate programming is the ability to write code in the order that the programmer wants and not in the order that is needed by the computer. This means a code segment can be written on a place where it is not actually meant to be and can then be linked to the right place.

3 The Vision and History of WEB/CWEB

The WEB programming system was developed by Donald E. Knuth. Knuth discovered that in order to make a good documentation for a program two programming languages are needed: A language like TEX for the documentation and a language like Pascal (or C) for programming. But to work in two different languages is very inconvenient and takes a lot of work to keep the documentation and the code always up to date. It is also very time consuming to insert pieces of the code in the documentation and to format it properly. But with a combination of both languages a very good system to program and document at the same time can be created.

So the first version of WEB was created as the combination of TEX and Pascal. The name WEB was chosen because during the time of creation the acronym WEB was not used in the computer science and the name was a good description for a complex program that is made of many connected pieces [2]. For the documentation of a complex program each part of the "web" must be explained and how it relates to the other pieces of the program. TEX provides the functionality to explain the part of the program in a natural language. Pascal provides the functionality to explain the program to the computer. Knuth was very exited about his new invention and wrote his famous programs TEX and METAFONT in this programming language [1]. He hoped that this new methodology would help people to write better documentations and so to make a program easier to develop. But he confessed also that programming is a very personal activity and that this methodology might not be the best for everyone [3]. In 1987 Silvio Levy adapted the system of Knuth to C and created CWEB. It has essentially the same commands and structure as WEB, but uses C instead of Pascal for the programming language. As documentation language TEX was retained.

4 How CWEB Works

The program and the documentation of a CWEB program is written in the same file. This file is then used to generate the documentation and also an executable program. A CWEB source file normally has the ending ".w". The CWEB system consists of two different programs that are used to generate the outputs:

- The CWEAVE program "weaves" the web and is used to generate the documentation in form of a TEX file. The generated file contains the code of the program in a pretty printed format and the documentation that has been written for each code segment in TEX.

- The CTANGLE program "tangles" the web and is used to generate a C source file out of the CWEB source. The generated file does not contain any comments because it is only needed for the creation of an executable program.

In Figure 1 the compilation of a CWEB file is illustrated. Every node of the figure stands for a different file and every edge stands for a program that has to be executed to generate the next file.

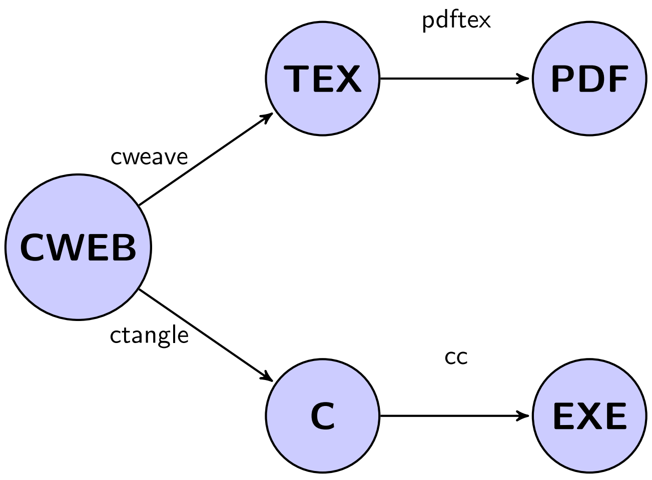


Figure 1: Compilation of a CWEB file.

5 Introduction to CWEB Programming

In this section a short introduction to CWEB programming will be given. Only the most important functions and commands will be covered. For further details about programming in CWEB please see [4]. In Section 5.1 the general structure of a CWEB program will be shown. In the Sections 5.2, 5.3 and 5.4 the three different parts of a CWEB section will be explained and some small programming examples are illustrated. Finally in Section 5.6 the special formatting abilities of CWEB will be explained.

5.1 Section

A CWEB file is divided in different sections. Each section consists of the following parts:

- A part of TEX code where the program that is written in this section can be explained. This TEX code will then be used to generate the documentation.
- A middle part where macros and the formatting of keywords can be defined.
- A C part that contains a few lines of the program and where special directives can link to source code of other sections.

These three parts must be exactly in this order, but each of these parts may also be empty. In CWEB a section can be started with two different character sequences:

- With the sequence “@␣“ where “␣” denotes a blank space. With this character sequence unnamed sections can be created.
- With the sequence “@*”a new named section can be created (also called “starred” section). The section name must be written after the sequence and must be followed by a dot (e.g. “@*section name.”). A named section denotes a new major group of sections. In the documentation a named section will create a new page with the section name as title. All following unnamed sections will have the same title as a running headline (until the next starred section starts). The section name should be a short description of the code that is written in this group.

A CWEB file only consists of many of these sections.

5.2 TEX Part

In the TEX part the source code of the section should be explained. The code that is written in the TEX section will then be copied to the resulting file. The result in the final documentation will be a block of text directly in front of the corresponding code. It is possible to use all normal TEX directives. It is not necessary to define titles, sections or other page format rules in a TEX section, because CWEB will do this work. The last TEX section has not to be finished with a “`\bye`” command like in a normal TEX file because CWEB will do this automatically at the end of the file.

5.3 The Middle Part

The middle part is the section between the TEX and the C section. The middle part begins with the first appearance of a middle part control sequence. In the middle part the following character sequences can be used to define rules:

@d With the @d command macros can be defined. The syntax of a macro definition is like the #define command in C and CTANGLE will actually compile the @d macro to a #define command. All the #define commands generated this way will be placed at the beginning of the resulting C file. This can be changed with the command @h which can be placed in a C section and which places all the defines of the middle parts at the place of the command.

In the documentation produced by CWEAVE the @d commands will also be shown as #define commands. The @d function is actually not really needed by CWEB because it does the same as the normal C define command. But because WEB's programming language Pascal did not have macros this function was needed in WEB and so it was also retained in CWEB. The @d commands do not have any effect on the TEX sections.

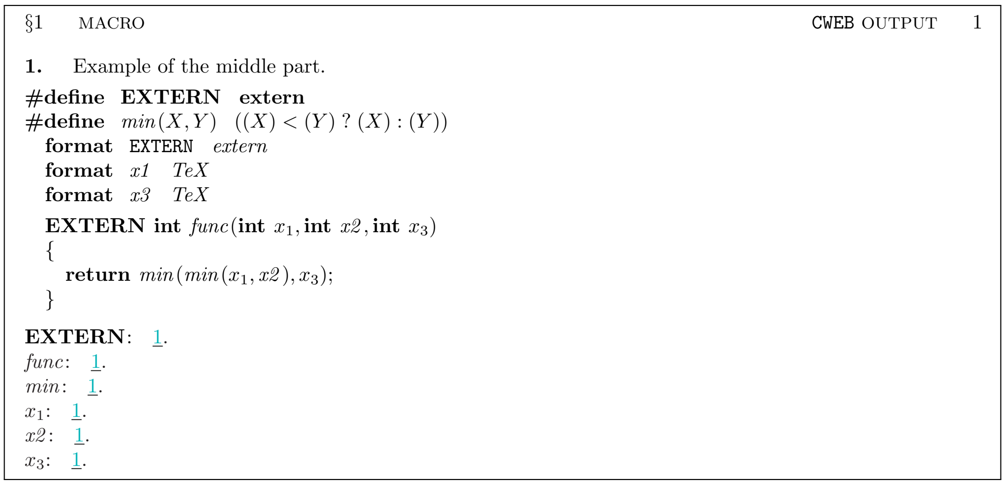
@f With the @f command special formatting rules can be defined. The definition of a format is in the form "@f l r" where l and r are identifiers. A comment enclosed between /* and */ can also be specified. With a format definition of this shape CWEB will handle the identifier l as it handles r. This means with this command l will get the same format as r. This allows a programmer to define new reserved words and to specify a special formatting for self defined keywords. If an identifier should have no special formatting the keyword "normal" can be used for r (e.g. "@f int normal"). With the special keyword "TeX" for the identifier r it is also possible to format a keyword as a TeX sequence. For example the word foo would be converted to \foo in the TeX output. When a word is converted to a TeX command an underline character will be replaced with a 'x' and a dollar sign will be replaced with a 'X' (To avoid complications with TeX). All other characters and also digits are simply copied to the command. With this formatting it is possible to define very complex formatting rules which are then applied automatically.

As shown in Listing 1 and Figure 2 the format command and the "TeX" keyword can be used for example to make the strings $x1$ and $x3$ printed as x_1 and x_3.

With the @f command the formatting definition (and the comment for the definition) will also be documented in the TeX documentation. If it is not wanted that the format definition is shown in the documentation the @s command can be used. With the @s command it is not possible do define a comment after the definition.

Listing 1: Example of the middle part

```
\def\x#1{x_{#1}}
@
Example of the middle part.

@d EXTERN extern
@d min(X, Y)   ((X) < (Y) ? (X) : (Y))
@f EXTERN extern
@f x1 TeX @f x3 TeX

@c
EXTERN int func(int x1, int x2, int x3)
{
        return min(min(x1,x2),x3);
}
```



§1 MACRO CWEB OUTPUT 1

1. Example of the middle part.

#define EXTERN extern
#define $min(X,Y)$ $((X) < (Y) \mathbin{?} (X) : (Y))$
  format EXTERN *extern*
  format *x1* *TeX*
  format *x3* *TeX*
  EXTERN int *func*(**int** x_1, **int** *x2*, **int** x_3)
  {
    return $min(min(x_1, x2), x_3)$;
  }

EXTERN: 1.
func: 1.
min: 1.
x_1: 1.
x2: 1.
x_3: 1.

Figure 2: Resulting document for the code in listing 1

5.4 C Part

In CWEB a C part can be defined in two different ways:

- An unnamed C part starts with the sequence "@c". A unnamed C part represents a part of the C file which will not be included by other C parts. All unnamed C parts will be used to initially generate the resulting C file. This means if no unnamed C part is defined, the C file produced by CTANGLE will be empty.

- A named C part starts with the sequence "`<@section name@>=`". A named C part can be included in another C part by writing the sequence "`<@inclusion part name@>`" in an C part. The name of the part should be long enough to clearly identify the code section. A section with a very long name can also be included with an abbreviation. An abbreviation can be created with 3 dots (e.g. `@<A very long section name@>` can be abbreviated to `@<A very...@>`). An abbreviation "`<@`α`...@>`" matches a section if the defined section begins with α. If an abbreviation matches to more than one section a compile error will be thrown. In Listing 2 an example of a inclusion with a abbreviation is shown. If a good formatting of the code is wanted it is recommended that a ";" is written after each inclusion. Otherwise this will lead to a strange format in the resulting documentation (The inclusion and the following C command will be in the same line). The ";" after each section will then also be copied to the resulting code file but this will only lead to errors when one C command is divided to two different C sections (which should never happen with a good coding style). In CWEB it is also possible to give the same name to two named C sections. When the second or all following sections are defined the a '+' must be added to the definition directive (e.g. `<@Second section@>=+`). The complete section will then be constructed by putting together all sections with the same name.

Listing 2: Example of the inclusion mechanism in CWEB

```
@
The main program
@c
int main()
{
        <@The main...@>;
}

@
The sub program
<@The main code of the program@>=
printf("Hello World");
```

5.5 In Limbo

In the CWEB file there is a special part that is located before the first section. This part is called “in limbo” and is reserved for TEX commands. It will be copied in verbatim to the beginning of the TEX file produced by CWEAVE, so that it can be used to load files or to write TEX code that does not belong to a special section. The limbo section will be ignored by CTANGLE because it isn’t relevant for the source code file. The limbo section is often used to define formatting parameters like the page layout, the page title or to load special fonts. But it can also be used for example to define TEX commands which can then be used in other parts. In Listing 3 a sample code with a limbo section that defines a document title and some simple commands is shown.

Listing 3: Example of the in limbo part

```
%In limbo section
\def\title{Document Title}
\def \beginquote {\par \begingroup \narrower}
\def \endquote {\par \endgroup}

@
\beginquote QUOTE \endquote
@c
int i;
```

5.6 Formatting C Code in the TEX Section and Vice Versa

When a documentation for a C part is written, often C keywords and a C like highlighting is needed. Because it is very exhausting to format the C code manually, CWEB offers a function to automatically format C code in the TEX section. So all the keywords used in a C section can be formatted exactly the same way in the TEX section. To write C code in a TEX section it has to be written in a special environment which is started and ended with a '|' symbol. Every text between the '|' symbols will be formatted like the code in the C section. The text written in such an environment cannot be used by CTANGLE or in other C parts because it is only a special formatting environment for the TEX section.

And because it is also very comfortable to have a formatting system like TEX in the C section it is also possible to write TEX commands in a C comment. So it is possible to write formulas or other formatting sequences as comment which can be very useful if for example mathematical formulas have to be explained. The C comments will not be copied to the source code file and so the TEX code will make no problems during the C compilation.

The function of these formatting abilities are shown in the Listing 4 and in Figure 3 where a few simple format definitions of C code and mathematical formulas are shown.

Listing 4: Example of the Formatting abilities in CWEB

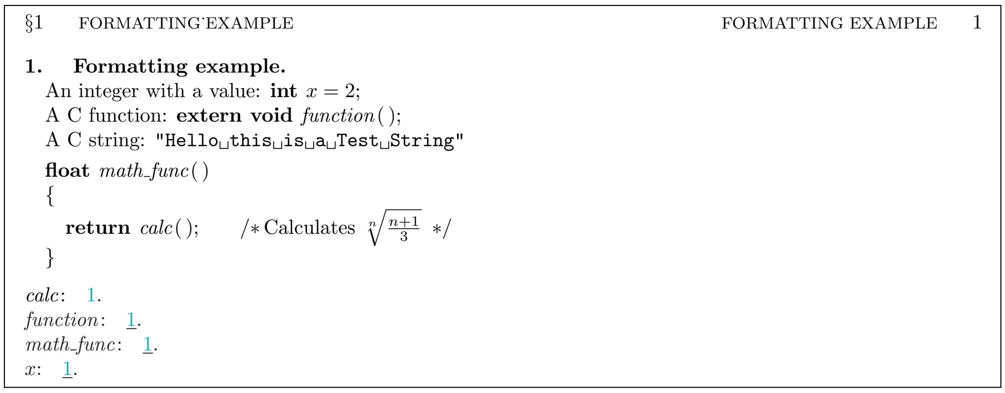
```
@ An integer with a value: |int x=2;| \par
A C function: |extern void function();| \par
A String:  |"Hello this is a Test String"|

@c
int math_func(){
        calc(); //Calculates $\root n\of{n+1\over 3}$
}
```

6 Compilers

In this section the operating principle of the CWEB compilers will be explained. Then in Section 6.3 an error which is very easy to make but very hard to find will be presented.

6.1 Operating Principle of CWEAVE

The first thing CWEAVE inserts in the TEX file that is generated is a inclusion of a file which will load the formatting definitions of CWEB. After that CWEAVE begins to process the CWEB file with the limbo section which will be copied to the resulting file as plain-text. The next step is to parse all middle parts



§1 FORMATTING·EXAMPLE FORMATTING EXAMPLE 1

1. Formatting example.
An integer with a value: **int** $x = 2$;
A C function: **extern void** *function*( );
A C string: `"Hello␣this␣is␣a␣Test␣String"`

float *math_func*( )
{
  return *calc*( ); /* Calculates $\sqrt[n]{\frac{n+1}{3}}$ */
}

calc: 1.
function: 1.
math_func: 1.
x: 1.

Figure 3: Resulting document for the code in Listing 4

and to save all formatting definitions because they are needed for the following parts. Then the sections of the CWEB file will be processed in the order of appearance. The TEX code will be copied in plain-text and every C part will be formatted according to the self defined and the standard CWEB rules. Finally a cross-reference for all variables and sections that appear in the CWEB file will be created.

6.2 Operating Principle of CTANGLE

The first step when compiling a file with CTANGLE is to process all middle parts The definitions of the middle parts are then copied to the beginning of the C file (as #define commands). After that, all unnamed C parts of the file are copied to the resulting C file. The parts are ordered as they were in the source file, which is important because a wrong ordering can lead to compilation errors of the C file, e.g. wrong ordered functions. In the resulting file all named sections are replaced by the corresponding C code. This is repeated until no named section can be found. If two or more named sections have the same name the sections are put together in the order of appearance in the source file.

All comments are removed because the code generated by CTANGLE is only meant for the compiler. CTANGLE appends #line preprocessor commands to the resulting C file. The #line command specifies the line number which should be reported for the following line of input. With these commands it is possible that the C compiler can tell in which line of the CWEB file a compilation error occurs (Which is more useful than a line number of the generated C file).

6.3 Error Handling of the Compilers

The programs CTANGLE and CWEAVE are generally very tolerant with errors in the source files. This means that often these two programs compile your program successfully but in the next compilation step (TEX or C compilation) many errors are found. For example the code in Listing 5 shows a very short example code which will be compiled correctly with CWEAVE, but when you try to compile the resulting file with pdftex the cryptic error in Listing 6 will be shown. These errors are often very hard to find because the error was produced by the commands generated by CWEAVE. The error in Listing 5 is that no section name after "@*" was specified and this leads to errors in the TEX commands generated by CWEAVE.

Listing 5: A code that will lead to an error in the second compilation step

```
@*
A simple code that will lead to an error
@c
int i=0;
```

Listing 6: Error log for the code in listing 5

```
Runaway argument?
 Test section \Y \B \&{int} \|i${}\K \T {0}{}$;
! Paragraph ended before \N was complete.
<to be read again>
                  \par
l.5 \Y\B\&{int} \|i${}\K\T{0}{}$;\par
```

7 Usage and Influence of WEB/CWEB

In this section the influences of WEB and CWEB will be explained. In Section 7.1 other literate programming languages that where influenced by WEB and CWEB will be shown. In Section 7.2 web and CWEB will be compared with other documentation systems like Javadoc, doxygen or ocamldoc.

7.1 Other Literate Programming Languages

Many programming languages where influenced by WEB/CWEB:

- noweb: A literate programming language that is not bound to a specific programming language and where many documentation languages can be used.

- nuweb: A literate programming language that has TEX as documentation language but any programming language can be used.

- Funnelweb: A platform independent literate programming language which can generate TEXand HTML documents. Can be used with any programming language.

- Lp4all: A literate programming language that has a wiki like syntax. The documentation of the program is written in the comments of the programming language.

A important language that was influenced by literate programming is Haskell which has an own mode for literate programming. In Haskell only the suffix ".lhs" instead of ".hs" has to be used and a full literate programming system can be used. In Haskell LaTeX can be used as documentation language and code can be written between "`\begin{code}`" and "`\end{code}`" statements.

7.2 Comparison to Other Documentation Systems

The languages WEB and CWEB are very different from other modern documentation systems like Javadoc, doxygen or ocamldoc. CWEB provides the functionality to document the whole computer program while documentation systems like Javadoc only provide the functionality to document the interfaces of a program (e.g. functions). CWEB also offers the functionality to order a program in a different way while in other programs the structure of a program is defined by the computer. But the main idea of generating a documentation out of a source code file was adopted by all of these documentation systems.

8 Conclusion

In this report the main concepts of literate programming are described and a short introduction of programming in CWEB was given. The languages WEB and CWEB offer many different functions to make a good documentation for a program. The functionality of creating a documentation for a whole program and not only for a interface has not yet become a standard in newer languages. And also the feature of ordering code in a different way is not provided by many languages. Because of this CWEB is a very special language which could influence many more languages in the future.

References

[1] D. Knuth. *TEX: The Program.* Addison Wesley Publishing Company, 1986.

[2] D. E. Knuth. The WEB system of structured documentation. Stanford Computer Science Report CS980, Stanford University, Stanford, CA, Sept. 1983.

[3] D. E. Knuth. Literate programming. *The Computer Journal*, 27(2), 1984.

[4] D. E. Knuth and S. Levy. *The CWEB System of Structured Documentation.* Addison Wesley, Reading, 1994.

[5] N. Ramsey and C. Marceau. Literate programming on a team project. *Softw., Pract. Exper.*, 21(7):677–683, 1991.