**Institute of Technology, Carlow**
**B.Sc. in Software Engineering**

**CW228**

# Specification Manual

## *C Maintenance Tool*

Name: Anna-Christina Friedrich

ID: C00132716

Supervisor: Dr. Christophe Meudec

Submission Date: 11th December 2009

# Table of Contents

# 1. Vision

## 1.1. Introduction

The aspired maintenance tool for C language(from now on called 'CMT') provides cross referencing of all identifiers within a C sourcefile. It also allows the user to rename certain identifiers and to create html output files. C89 Standard will be required for the C input files.

## 1.2. Positioning

### 1.2.1 Business Opportunity

Existing maintenance tools have a lack of reliability. They don't consider the scope of variables while crossreferencing and don't include preprocessing commands. According to researches, C is the second most used programming language in the internet. Therefore there is demand on a reliable, easy to use maintenance tool, that considers the mentioned lacks.

### 1.2.2 Problem Statement

There is a big problem with existing maintenance tools for C according to the reliability. It is important that identifiers are ordered by their scope, so that the code is always executable, particularly while renaming. Furthermore a solution is needed to handle preprocessor commands, because those include identifiers as well, but are not legal C code. Building a maintenance tool, which provides crossreferencing for each identifier, with special focus on scope and preprocessing is a solution.

### 1.2.3 Product Position Statement

This product CMT is a crossreferencing tool for C language and its main stakeholder is a software developer, who requires a tool to maintain large files of C code easily and reliably. My product provides a system that crossreferences each identifier of C sourcecode using html. Additionally there is also the facility of renaming variables and I aim at the tool becoming a free plugin. Unlike past maintenance tools, my product is more reliable and easy to use.

## 1.3 Stakeholder Descriptions

Stakeholders are software engineers and students attending courses like software engineering. They want the tool to be first of all reliable, as well as easy to handle and to have a good documentation.

# 1.4 Product Overview

## 1.4.1 Product Perspective

CMT will be used for maintaining existing code by software engineers as well as software engineering students.It will provide merely few services to users, won't interact with other systems and there is just few interaction between user and system as well.

## 1.4.2 Summary of System Features

- Building a crossreferenced datastructure for each identifier considering scope, preprocessing
- providing each identifier as an html output file
- possibility to choose an identifier and renaming it without changing the meaning of the code
- use as independent software or as an plugin for an existing development tool

# 2. Use Case Diagram

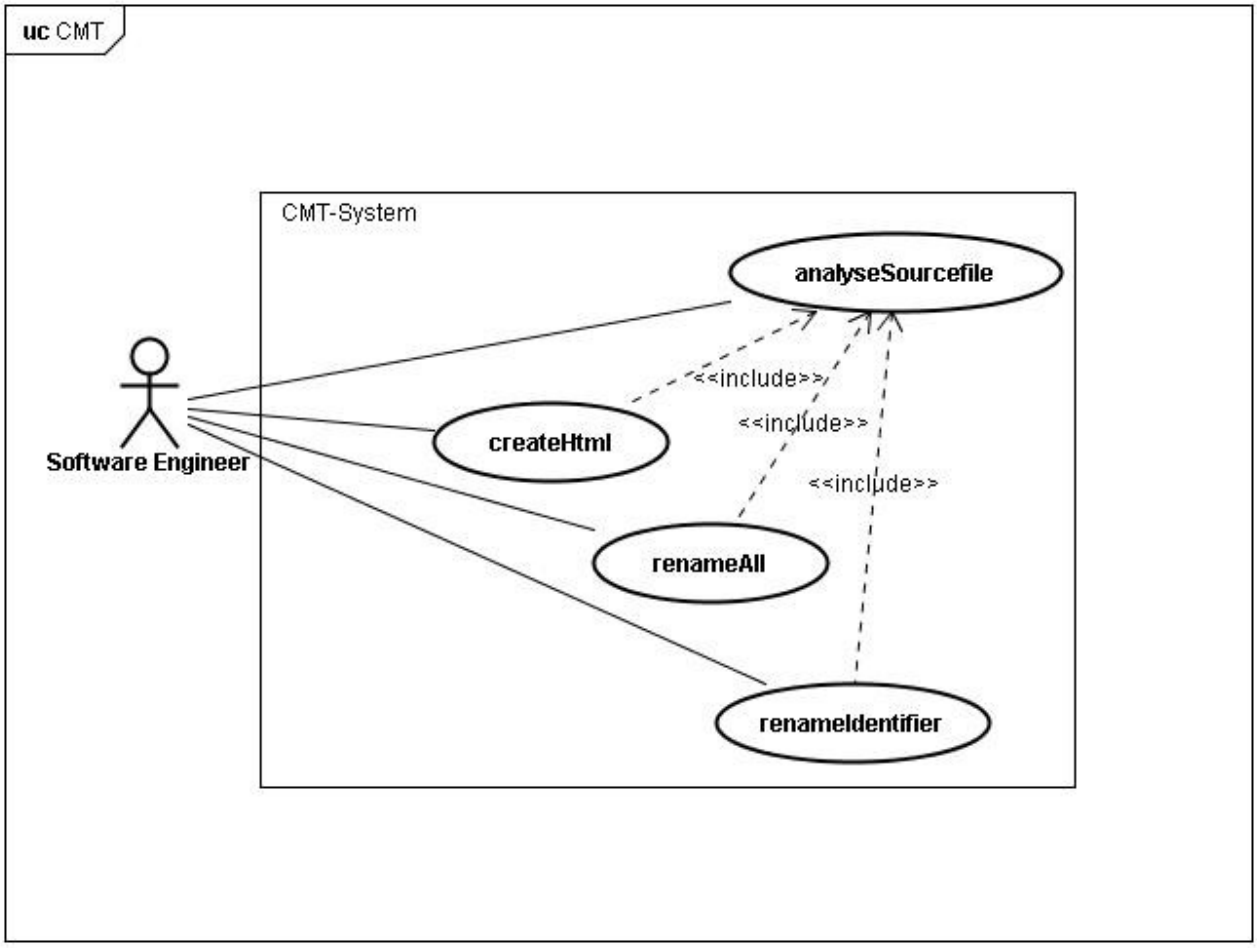


*Figure 1: Use Case Diagram for the C Maintenance Tool (CMT)*

# 3. Use Cases

## 3.1 analyseSourcefile

**Use case:** analyseSourcefile

**Scope:** CMT application

**Primary actor:** software engineer

**Stakeholders and interests:**

software engineer: wants reliable and performant outcome

**Preconditions:** none

**Postconditions:** a datastructure with each occurence of every identifier was created

**Main success scenario:**

1. user starts the system and passes the sourcefile that is to analyse
2. system parses the sourcecode
3. system creates a datastructure in which each identifier is stored with its occurences
4. system provides definition line,column and name of each identifier stored in a textfile

**Extensions:**

2a. there is an identifier that can't be refactored

1. system signals error and refuses the referencing of the code

## 3.2 createHtml

**Use case:** createHtml

**Scope:** CMT application

**Primary actor:** software engineer

**Stakeholders and interests:**

software engineer: wants reliable and performant outcome

**Preconditions:** a datastructure and a textfile was generated

**Postconditions:** html files that name each identifier with its definition and occurences

**Main success scenario:**

1. user chooses html facility
2. system builds html files out of the given datastructure and/or textfile

**Extensions:** none

## 3.3 renameIdentifier

**Use case:** renameIdentifier

**Scope:** CMT application

**Primary actor:** software engineer

**Stakeholders and interests:**

software engineer: wants reliable and performant outcome

**Preconditions:** a datastructure was generated

**Postconditions:** one or more identifiers are renamed in each appearance

**Main success scenario:**

1. user chooses rename facility and passes the identifier and its definiton line that is supposed to be renamed
2. based on the given datastructure, the system changes the name of each occurence of the chosen identifier and creates a new sourcefile including the changes

**Extensions:** none

## 3.4 renameAll

**Use case:** renameAll

**Scope:** CMT application

**Primary actor:** software engineer

**Stakeholders and interests:**

software engineer: wants reliable and performant outcome

**Preconditions:** a datastructure was generated

**Postconditions:** each identifier is renamed in each appearance

**Main success scenario:**

1. user chooses facility to rename each identifier
2. based on the given datastructure, the system changes the name of each occurence of each identifier automatically and stores the modificated sourcefile as a new file

**Extensions:** none

# 4. Supplementary Specification

## 4.1 Introduction

This document is the repository of all CMT requirements not captured in the

use cases.

## 4.2 Functionality

(Functionality common across use cases and example output)

*Accuracy*

The tool has to provide correct and appointed outcome, especially while renaming. This will be tested by using the facility 'renameAll'. After compiling the original file and the modificated file, the outcome will be compared. If the outputs are equal, the tool was successfull in renaming all identifiers.

*Interoperability*

The ability of cooperating with different systems will be achieved with a plugin of the tool for Eclipse IDE.

## 4.3 Usability

*Human Factors*

Although users like software engineers are commonly used to handle new, unknown systems, an understandable user manual supports the first steps of handling the system. CMT will be based on command line, but the user does not have to spend much effort on learning to work with the system. Handling will be easy.

## 4.4 Reliability

*Fault Tolerance*

The tool has to work absolutely correct, otherwise the whole input code can become invalid. Therefore it is important to catch exceptions before code is changed and to interrupt the system. The user gets an error information, why the action cannot be continued.

*Recoverability*

The facility of undoing an action needs to be supported, if the user made e.g. a mistake while renaming. This will be done by duplicating the given sourcefile, so that there is still the original file

and a modificated one.

## 4.5 Performance

Performance is always important but under the conditions of CMT, the performance has minor priority. Nonetheless the best performance will be assured with using an adequate programming language(Java, see 4.7) that handles large datastructures the best.

## 4.6 Supportability

*Adaptability*

First of all the tool will run on Windows as a executable jar file. It is also aimed at providing a support for Linux systems.

*Maintainability*

CMT system will be structured into layers and built up with the help of design patterns, therefore a good maintainability can be ensured. Furthermore Javadoc annotations will be used to make maintaining the sourcecode of CMT more comfortable.

## 4.7 Implementation

*Programming Language*

To support maintainability and performance, the programming language will be Java.

*Free Open Source Components*

There have already been made decisions about free tools that support the development of CMT:

- Eclipse IDE for Java Developers
- Antlr and AntlrWorks for parsing input
- PDE, the plugin development environment for Eclipse
- JUDE/Community, free UML modeling tool
- Mozilla Firefox to display html documents

## 4.8 Interfaces

*Hardware*

none

*Software Interfaces*

It is aimed to provide a plugin for an IDE.

## 4.9 Legal Issues

CMT won't be a commercial tool, neither it is planned to get released. Therefore I don't have to worry about legal issues.

# 5. Example Output

## 5.1 Sourcefiles

The following files will be exemplary analysed, i. e. each identifier is listed with its name, definition, declaration and occurences. Pre-defined identifier(like int, char...) and system files(like stdio.h, stdlib.h...) are excluded from analysing. This will probably be realised by adding certain rules to the grammar used in parsing generator. Parameters and variables which are only used within macros won't be referenced for now. Column numbers are given exemplary for the first three identifiers, but they will be given for each identifier in the executable tool.

### 5.1.1 example.c

```
1  #include <stdio.h>
2  #include "myCalc.h"
3  #include “do.h”
4
5  int main(void){
6     int index;
7     int firstNum = 12;
8     int secondNum = 13;
9
10    printf("First summand is: %d", firstNum);
11    printf("\nSecond summand is: %d", secondNum);
12
13    int sum;
14    sum = plus_int(firstNum,secondNum);
15    printf("\n\nThe sum is: %d", sum);
16
17    do{
18          int index = 2;
19          sum+=index;
20    }while(sum<MAXSIZE);
21
22    index = MAKEIT(sum,secondNum);
23
24    write(“This is a string!”);
25    write(“And so is this.”);
26
27    return 0;
28 }
```

### 5.1.2 myCalc.h

```
1  #ifndef myCalc_h
2  #define myCalc_h
3
4  #define MAXSIZE 100
5
6  int plus_int(int, int);
7
```

```
8  double plus_double(double, double);
9
10 #endif
```

### 5.1.3 myCalc.c

```
1  #include "myCalc.h"
2
3  int plus_int(int inum1, int inum2){
4
5     return (inum1+inum2+MAXSIZE);
6  }
7
8  double plus_double(double dnum1, double dnum2){
9
10    return (dnum1+dnum2);
11 }
```

### 5.1.4 do.h

```
1  #ifndef do_h
2  #define do_h
3
4  #define MAKEIT(one, two) (one+two)
5
6  void write(char*);
7
8  int readInt();
9
10 #endif
```

### 5.1.5 do.c

```
1  #include "do.h"
2  #include "myCalc.h"
3
4  void write(char* text){
5
6     printf("%s", text);
7  }
8
9  int readInt(){
10
11    int read;
12    scanf("%d", &read);
13    return (read+MAXSIZE);
14 }
```

# 5.2 Output

**variables**

*index*

| | | | |
|---|---|---|---|
| type: | int | | |
| definition: | example.c | line 6 | column 9 |
| occurances: | example.c | line 22 | column 5 |

*index*

| | | | |
|---|---|---|---|
| type: | int | | |
| definition: | example.c | line 18 | column 15 |
| occurances: | example.c | line 19 | column 16 |

*firstNum*

| | | | |
|---|---|---|---|
| type: | int | | |
| definition: | example.c | line 7 | column 9 |
| occurances: | example.c | line 10 | column 36 |
| | example.c | line 14 | column 20 |

*secondNum*

| | | | |
|---|---|---|---|
| type: | int | | |
| definition: | example.c | line 8 | |
| occurances: | example.c | line 11 | |
| | example.c | line 14 | |
| | example.c | line 22 | |

*sum*

| | | | |
|---|---|---|---|
| type: | int | | |
| definition: | example.c | line 13 | |
| occurances: | example.c | line 14 | |
| | example.c | line 15 | |
| | example.c | line 19 | |
| | example.c | line 22 | |

*inum1*

type: int
definition: myCalc.c line 3
occurances: myCalc.c line 5

*inum2*
type: int
definition: myCalc.c line 3
occurances: myCalc.c line 5

*dnum1*
type: double
definition: myCalc.c line 8
occurances: myCalc.c line 10

*dnum2*
type: double
definition: myCalc.c line 8
occurances: myCalc.c line 10

**functions**
*plus_int*
return type: int
declaration: myCalc.h line 6
definition: myCalc.c line 3
occurances: example.c line 14

*plus_double*
return type: double
declaration: myCalc.h line 8
definition: myCalc.c line 8
occurances: none

*write*
return type: void
declaration: do.h line 6

| | | |
|---|---|---|
| definition: | do.c | line 4 |
| occurances: | example.c | line 24 |
| | example.c | line 25 |

*readInt*

| | | |
|---|---|---|
| return type: | int | |
| declaration: | do.h | line 8 |
| definition: | do.c | line 9 |
| occurances: | none | |

**preprocessor**

*MAXSIZE*

| | | |
|---|---|---|
| definition: | myCalc.h | line 4 |
| occurances: | myCalc.c | line 5 |
| | do.c | line 13 |
| | example.c | line 20 |

*MAKEIT*

| | | |
|---|---|---|
| definition: | do.h | line 4 |
| occurances: | example.c | line 22 |

**sources**

| | |
|---|---|
| myCalc.c | C:\Program Files\Common Files |
| myCalc.h | C:\Program Files\Common Files |
| do.c | C:\Program Files\Common Files |
| do.h | C:\Program Files\Common Files |
| example.c | C:\Program Files\Common Files\my_project |