2022

CAB403

SYSTEMS PROGRAMMING

ASSIGNMENT 2
CAR PARK MANAGEMENT SYSTEM AND SIMULATOR
SEMESTER 2, 2022 – GROUP 008

LUKE ALBERT	N10731300
JOON SUNG KIM	N10916059
JUSUNG HAM	N9541501
KWANHYUN KIM	N10831436

Table of contents

Statement of completeness

Following implementation of the Carpark Management System (CMS), the following features have been implemented:

Tasks(s)	**Completed**	**Remaining**	**Sign**
Manager	✓	0	[illegible]
Simulator	✓	0	[illegible]
Fire alarm	✓	0	[illegible]
Car park structure	✓	0	[illegible]
Timings - Simulator	✓	0	[illegible]
Timings - Manager timings	✓	0	[illegible]
Timings - Fire alarm timings	✓	0	[illegible]
Permitted vehicle identification	✓	0	[illegible]
Billing	✓	0	[illegible]
Fire detection - Fixed temperature detection	✓	0	[illegible]
Fire detection - Rate-of-rise fire detection	✓	0	[illegible]
Shared memory – Inter-process communication	✓	0	[illegible]
Shared memory structure	✓	0	[illegible]
Busy waiting	✓	0	[illegible]
Unusual behaviour of vehicles	x	1	[illegible]
Status display	✓	0	[illegible]
Report	✓	0	[illegible]
Video	✓	0	[illegible]

Group 008 have implemented the above functionalities outlined within the CMS v1.00 specification.

Statement of contribution

Throughout the development and implementation processes of the carpark management system, all members were involved across all aspects, including the following features and functionalities:

Task(s)	**Luke**	**Joon**	**Jusung**	**Kwanhyun**
Program structure and design	✓	✓	✓	✓
Management system (manager.c)	✓	✓	✓	✓
Simulator (simulator.c)	✓	✓	✓	✓
Fire alarm system (firealarm.c)	✓	✓	✓	✓
Safety-critical analysis	✓	✓	✓	✓
MISRA C identification, NASA (10), ISO 26262-6:2018	✓	✓	✓	✓
Fire alarm report rectifications	✓	✓	✓	✓
Current potential safety-critical concerns	✓	✓	✓	✓
Report	✓	✓	✓	✓
Video	✓	✓	✓	✓

Safety-critical assessment of provided file

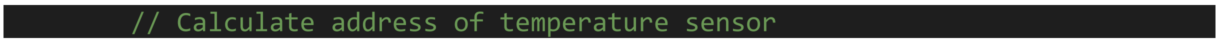
Whilst a provided “firealarm.c” was initially provided, various safety-critical violations were noted upon initial inspection. These breaches correlate to the Motor Industry Software Reliability Association (MISRA) standards documented within the MISRA “Guidelines for the use of the C language in critical systems” (MISRA, 2019). In addition, the MISRA rules are categorised into three (3) classifications including: Mandatory, Required, and Advisory.

As a result, the below breaches have been notified in accordance with MISRA Section 8 – Rules (MISRA, 2019, p38).

In addition to MISRA, “firealarm.c” has been analysed with two additional standards: National Aeronautics and Space Administration (NASA) The Power of 10 “Rules for Developing Safety-Critical Code”, and International Organisation for Standardisation (ISO) 26262-6:2018 Road vehicles – Functional safety – Part 6: Product development at the software level.

Comments

MISRA C Rule 3.2 – Line-splicing shall not be used in // comments. (Required).

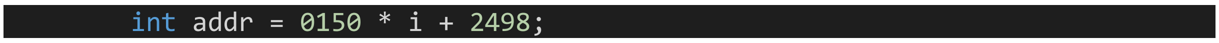


```
        // Calculate address of temperature sensor
```

(screenshot obtained via firelarm.c, line 62)

Literals and constants

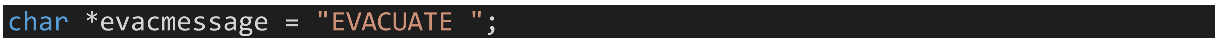
MISRA C Rule 7.1 – Octal constants shall not be used. (Required).



```
        int addr = 0150 * i + 2498;
```

(screenshot obtained via firelarm.c, line 170)

MISRA C Rule 7.4 - A string literal shall not be assigned to an object unless the object’s type is “pointer to const-qualified char”. (Required).

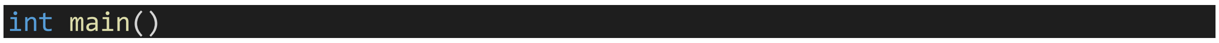


```
char *evacmessage = "EVACUATE ";
```

(screenshot obtained via firelarm.c, line 190)

Declarations and definitions

MISRA C Rule 8.2 - Function types shall be in prototype form with named parameters. (Required).

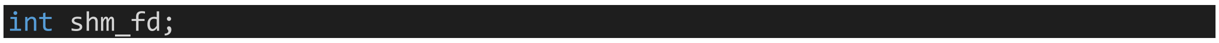


```
int main()
```

(screenshot obtained via firelarm.c, line 147)

MISRA C Rule 8.4 - A compatible declaration shall be visible when an object or function with external linkage is defined. (Required).

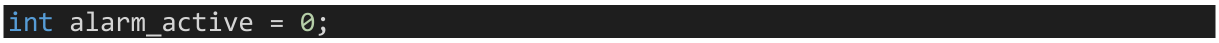
ISO 26262-6:2018 1(c) – Initialization of variables.



```c
int shm_fd;
```

(screenshot obtained via firelarm.c, line 190)

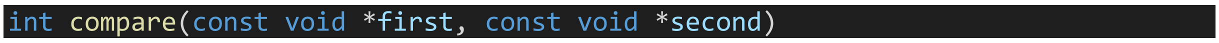
ISO 26262-6:2018 1(e) – Avoid global variables or else justify their usage.



```c
int alarm_active = 0;
```

(screenshot obtained via firelarm.c, line 13)

MISRA C Rule 8.7 – Functions and objects should not be defined with external linkage if they are referenced in only one translation unit. (Advisory).

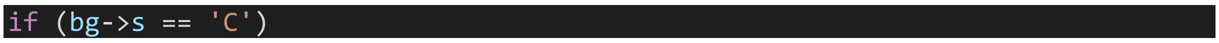


```c
int compare(const void *first, const void *second)
```

(screenshot obtained via firelarm.c, line 51)

The essential type model

MISRA C Rule 10.4 – Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category. (Required).

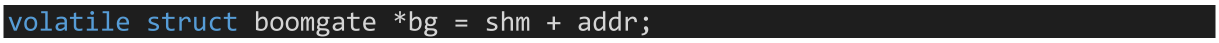


```c
if (bg->s == 'C')
```

(screenshot obtained via firelarm.c, line 135)

Pointer type conversions

MISRA C Rule 11.5 - A conversion should not be performed from pointer to void into pointer to object. (Advisory).

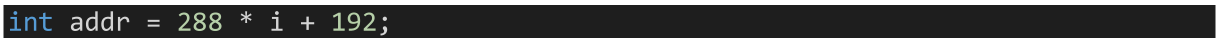


```c
volatile struct boomgate *bg = shm + addr;
```

(screenshot obtained via firelarm.c, line 179)

Expressions

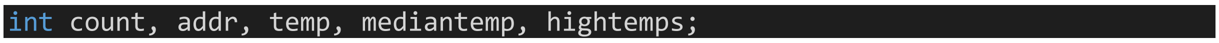
MISRA C Rule 12.1 – The precedence of operators within expressions should be made explicit. (Advisory).



```c
int addr = 288 * i + 192;
```

(screenshot obtained via firelarm.c, line 193)

MISRA C Rule 12.3 – The comma operator should not be used. (Advisory).

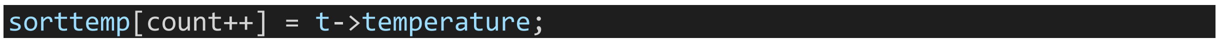


```
int count, addr, temp, mediantemp, hightemps;
```

(screenshot obtained via firelarm.c, line 59)

Side effects

MISRA C Rule 13.3 – A full expression containing an increment (++) or decrement (--) operator should have no other potential side effects other than that caused by the increment or decrement operator. (Advisory).

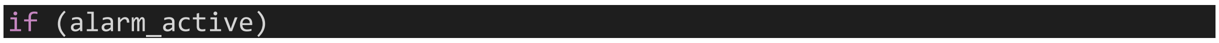


```
sorttemp[count++] = t->temperature;
```

(screenshot obtained via firelarm.c, line 85)

Control statement expressions

MISRA C Rule 14.4 – The controlling expression of an if statement and the controlling expression of an iteration-statement shall have essentially Boolean type. (Required).



```
if (alarm_active)
```

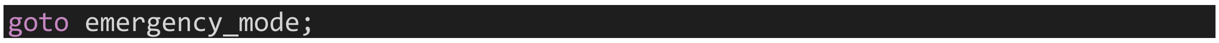
(screenshot obtained via firelarm.c, line 158)

Control flow

MISRA C Rule 15.1 – The goto statement should not be used. (Advisory).

NASA Rule 1 – Restrict all code to very simple control flow constucts.

ISO 26262-6:2018 1(i) – No unconditional jumps.

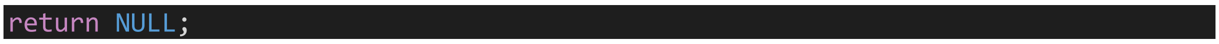


```
goto emergency_mode;
```

(screenshot obtained via firelarm.c, line 159)

MISRA C Rule 15.5 – A function should have a single point of exit at the end. (Required).

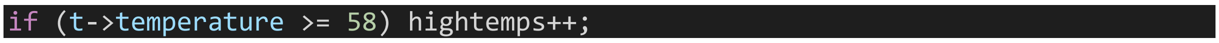
ISO 26262-6:2018 1(a) – One entry and one exit point in subprograms and functions.



```
return NULL;
```

(screenshot obtained via firelarm.c, line 47)

MISRA C Rule 15.6 – The body of an iteration-statement or a selection-statement shall be a compound-statement. (Required).



```
if (t->temperature >= 58) hightemps++;
```

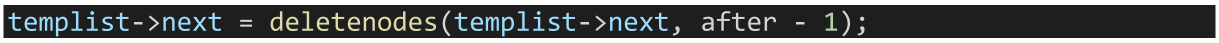
(screenshot obtained via firelarm.c, line 105)

Functions

MISRA C Rule 17.2 - Functions shall not call themselves, either directly or indirectly. (Required).

NASA Rule 8 – The use of the preprocessor must be limited to the inclusion of header files and simple macro definitions.

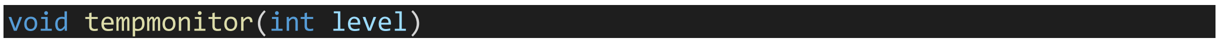
ISO 26262-6:2018 1(j) – No recurstions.



```
templist->next = deletenodes(templist->next, after - 1);
```

(screenshot obtained via firelarm.c, line 43)

NASA Rule 4 – No function should be longer than what can be printed on a single sheet of paper.

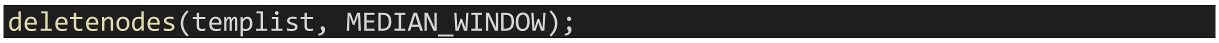


```
void tempmonitor(int level)
```

(screenshot obtained via firelarm.c, line 56)

MISRA C Rule 17.7 - The value returned by a function having non-void return type shall be used. (Required).

NASA Rule 7 – Each calling function must check the return value of non-void functions, and each called function must check the validity of all parameters provided by the caller.

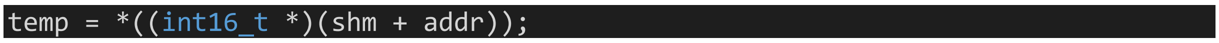


```
deletenodes(templist, MEDIAN_WINDOW);
```

(screenshot obtained via firelarm.c, line 73)

Pointers and arrays

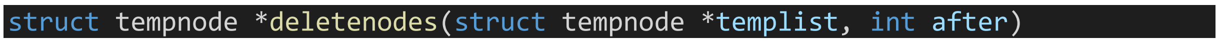
MISRA C Rule 18.4 – The +, -, += and -= operators should not be applied to an expression of pointer type. (Advisory).



```
temp = *((int16_t *)(shm + addr));
```

(screenshot obtained via firelarm.c, line 64)

NASA Rule 9 – Function pointers are not permitted.



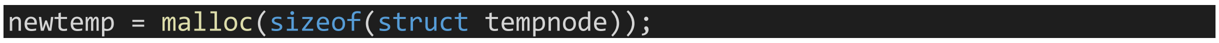
```
struct tempnode *deletenodes(struct tempnode *templist, int after)
```

(screenshot obtained via firelarm.c, line 40)

Standard libraries

MISRA C Rule 21.3 – The memory allocation and deallocation functions of <stdlib.h> shall not be used. (Required).

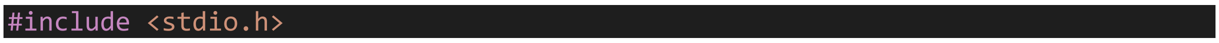
NASA Rule 3 – Do not use dynamic memory allocation after initialization.



```
newtemp = malloc(sizeof(struct tempnode));
```

(screenshot obtained via firelarm.c, line 91)

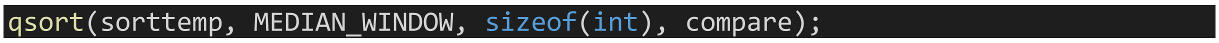
MISRA C Rule 21.6 – The Standard Library input/output functions shall not be used. (Required).



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

(screenshot obtained via firelarm.c, line 1)

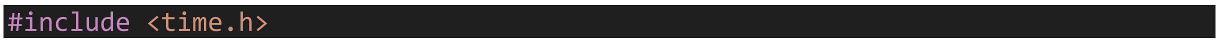
MISRA C Rule 21.9 – The library functions bsearch and qsort of <stdlib.h> shall not be used. (Required).



```
qsort(sorttemp, MEDIAN_WINDOW, sizeof(int), compare);
```

(screenshot obtained via firelarm.c, line 87)

MISRA C Rule 21.10 - The Standard Library time and date functions shall not be used. (Required).

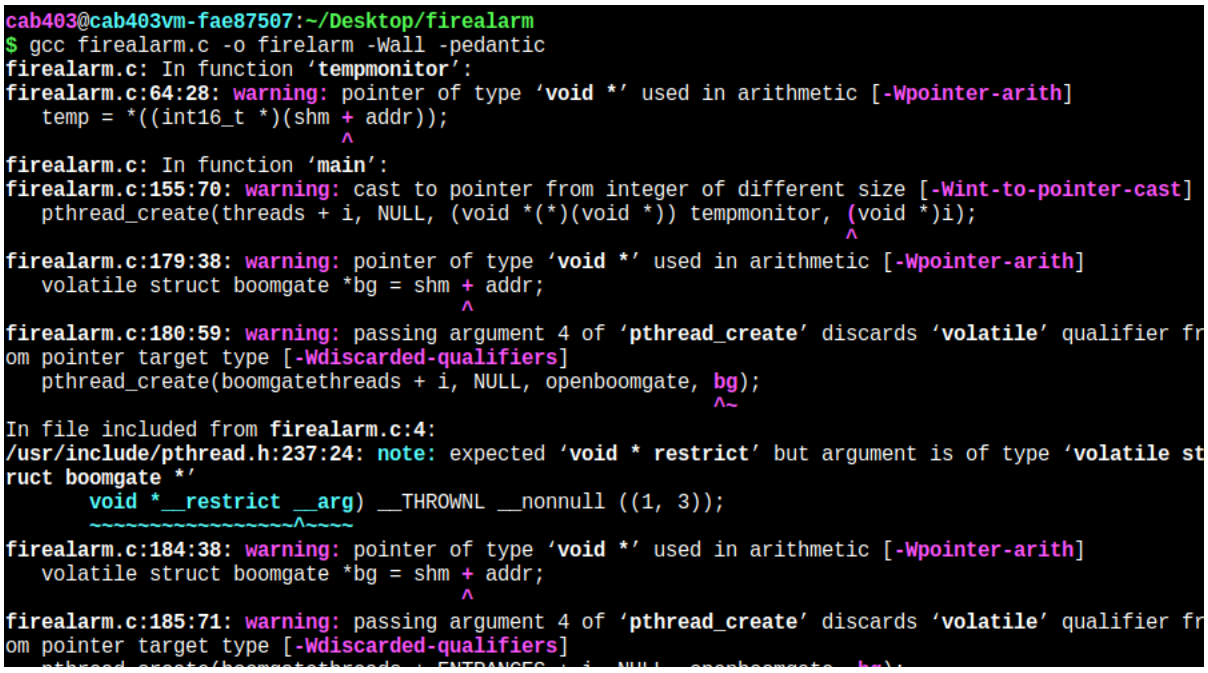


```
#include <time.h>
```

(screenshot obtained via firelarm.c, line 3)

Compilation

NASA Rule 10 –All code must be compiled, from the first day of development, with all compiler warnings enabled at the most pedantic setting available. All code must compile without warnings.



```
cab403@cab403vm-fae87507:~/Desktop/firealarm
$ gcc firealarm.c -o firelarm -Wall -pedantic
firealarm.c: In function ‘tempmonitor’:
firealarm.c:64:28: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
   temp = *((int16_t *)(shm + addr));
                            ^
firealarm.c: In function ‘main’:
firealarm.c:155:70: warning: cast to pointer from integer of different size [-Wint-to-pointer-cast]
   pthread_create(threads + i, NULL, (void *(*)(void *)) tempmonitor, (void *)i);
                                                                      ^
firealarm.c:179:38: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
   volatile struct boomgate *bg = shm + addr;
                                      ^
firealarm.c:180:59: warning: passing argument 4 of ‘pthread_create’ discards ‘volatile’ qualifier fr
om pointer target type [-Wdiscarded-qualifiers]
   pthread_create(boomgatethreads + i, NULL, openboomgate, bg);
                                                           ^~
In file included from firealarm.c:4:
/usr/include/pthread.h:237:24: note: expected ‘void * restrict’ but argument is of type ‘volatile st
ruct boomgate *’
       void *__restrict __arg) __THROWNL __nonnull ((1, 3));
       ~~~~~~~~~~~~~~~~~~~~~~^~~~~
firealarm.c:184:38: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
   volatile struct boomgate *bg = shm + addr;
                                      ^
firealarm.c:185:71: warning: passing argument 4 of ‘pthread_create’ discards ‘volatile’ qualifier fr
om pointer target type [-Wdiscarded-qualifiers]
```

(screenshot obtained via firelarm.c)

To have the provided firealarm.c file compile with -Wall -Wextra and -Werror, the following changed could be made.

```
    for (int i = 0; i < LEVELS; i++) {
        /*
        To adhere to -Wall -Wextra -Werror
        declared long l = i
        replaced (void *(*)(void *)) to (void*)
        replaced i to (void*)l
        */
        long l = i;
        pthread_create(threads + i, NULL, (void *)tempmonitor, 
(void *)l);
    }
```

The initial issue in the above is that the function makes a pointer from integer without a cast.

The other issue is that void(*)(int) is incompatible with void*(*)(void*)

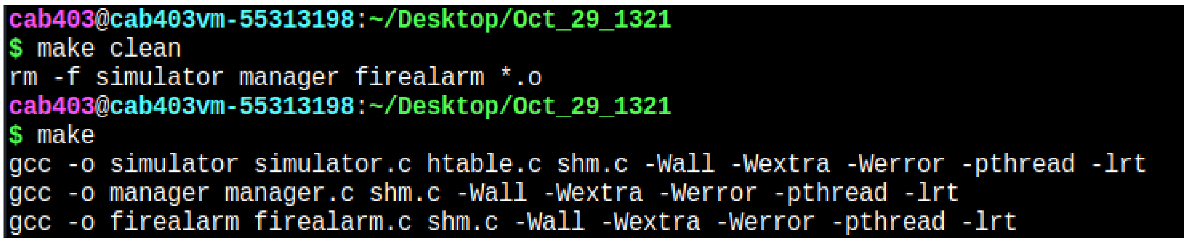
```
        /*
        To adhere to -Wall -Wextra -Werror
        replaced bg to (void*restrict)bg
        */
        pthread_create(boomgatethreads + i, NULL, openboomgate, 
(void*restrict)bg);
```

The issue in the above is that volatile is discarded from pointer.

```
                /*
                To adhere to -Wall -Wextra -Werror
                replaced &sign->m to (void*)&sign->m
                replaced &sign->c to (void*)&sign->c
                */
                pthread_mutex_lock((void*)&sign->m);
                sign->display = *p;
                pthread_cond_broadcast((void*)&sign->c);
                pthread_mutex_unlock((void*)&sign->m);
```

The issue in the above is that volatile is discarded from pointer.

As such, the following compilation followed:



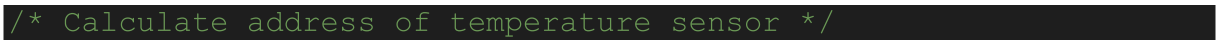
```
cab403@cab403vm-55313198:~/Desktop/Oct_29_1321
$ make clean
rm -f simulator manager firealarm *.o
cab403@cab403vm-55313198:~/Desktop/Oct_29_1321
$ make
gcc -o simulator simulator.c htable.c shm.c -Wall -Wextra -Werror -pthread -lrt
gcc -o manager manager.c shm.c -Wall -Wextra -Werror -pthread -lrt
gcc -o firealarm firealarm.c shm.c -Wall -Wextra -Werror -pthread -lrt
```

Safety-critical rectifications of provided file

Following identification of safety-critical violations outlined within the above section "Safety-critical assessment", it was decided that it would be best to rewrite portions of the initial code. As a result of the provided file, the following breaches have been rectified:

Comments

MISRA C Rule 3.2 – Line-splicing via "//" has been eliminated, and replaced with compliant "/* */" style comments. In the event that the "\" character is placed at the end of a "//" style comment, the proceeding line of code will be treated as a comment, thus not executing and causing potentially unaware behaviour (MISRA, 2019).

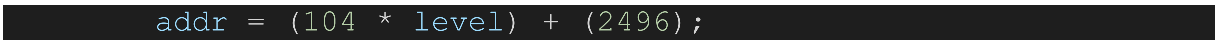


```
/* Calculate address of temperature sensor */
```

(screenshot obtained via firelarm.c)

Literals and constants

MISRA C Rule 7.1 – Octal constants have been eliminated, and replaced with decimal constants (no leading zeros). The use of octal constants may unintendingly result in incorrectly desired decimal declarations. As a result, decimal values have been implemented without leading zeros (MISRA, 2019).



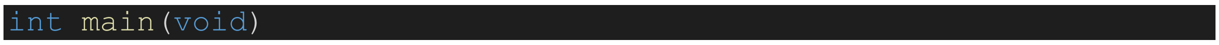
```
        addr = (104 * level) + (2496);
```

(screenshot obtained via firelarm.c)

MISRA C Rule 7.4 – String literal "EVACUATE" requires, unfortunately there was no way for us to rectify this occurrence, however its safety critical impact has been noted.

Declarations and definitions

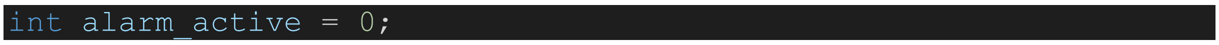
MISRA C Rule 8.2 - Function types have all been provided with prototypes, inclusive of named parameters. In addition, the main() function has been supplied with void (main(void)) to ensure that it takes no arguments/parameters. This prevents it from appearing as a variadic function. In addition, functions declared after the main function should be defined as a prototype prior to main (MISRA, 2019).



```
int main(void)
```

(screenshot obtained via firelarm.c)

MISRA C Rule 8.4, ISO 26262-6:2018 1(c) – Initialization of variables have been provided, such as initialising integer values to 0. This ensures the variable provides a value when called, even if a value not expected.



```
int alarm_active = 0;
```

(screenshot obtained via firelarm.c)

ISO 26262-6:2018 1(e) – Global variables should be avoided, however unfortunately there was no way for us to rectify this occurrence, however its safety critical impact has been noted.

MISRA C Rule 8.7 – External linkage of functions and objects is regarded as 'Advisory', and as such has been noted however its implementation remains as it is not within the 'Required' category.

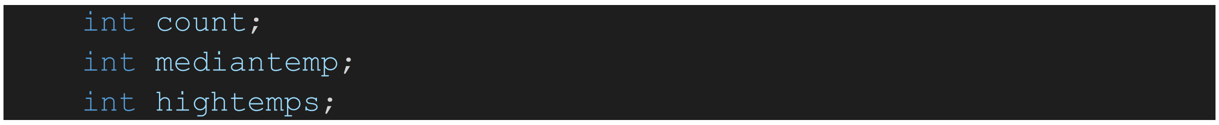
The essential type model

MISRA C Rule 10.4 – Arithmetic conversions of different operands have been omitted. This ensures that all arithmetic operations remain valid when executed.

Expressions

MISRA C Rule 12.1 – The precedence of operators within expressions have been made explicit with the inclusion of parentheses "( )" where applicable. This provides clarity over the order of operations in which calculations are performed, preventing possibly incorrect calculations and enhancing user readability (MISRA, 2019).

MISRA C Rule 12.3 – The comma operator has been eliminated in variable declarations. This is to improve code readability and prevent declarations being visually missed.



```
    int count;
    int mediantemp;
    int hightemps;
```

(screenshot obtained via firelarm.c)

Side effects

MISRA C Rule 13.3 – Increment (++) and decrement (--) operators have been declared outside of lengthy functions/assignments for clarity. This prevents accidentally unwanted results.

```
                count++;
                sorttemp[count] = t->temperature;
```
(screenshot obtained via firelarm.c)

Control statement expressions

MISRA C Rule 14.4 – Controlling expressions have been declared as boolean variables and/or compared with the "== true" operand. As such, a Boolean value has been implemented for adherence.

```
bool alarm_active = false;
```

```
        if (alarm_active == true) {
```

Control flow

MISRA C Rule 15.1, NASA Rule 1, ISO 26262-6:2018 1(i) – The goto statement has been eliminated. The goto statement can create difficulty in code readability which can further contribute to undesired control flow if not implemented correctly. As such, a more logical control flow structure has been implemented as a replacement. This ensures the remaining logic flows sequentially.

MISRA C Rule 15.5, ISO 26262-6:2018 1(a) – Early returns have been eliminated from functions, to incorporate a single point of exit. Return statements that are executed prior to the end of the function may result in unintentional logic, and as such the code has been reduced to one single point of exit.

```
        return 0;
```

MISRA C Rule 15.6 – Compound-statements have been implemented by including "{ }" within functions. The omission of "{ }" increases the difficulty of code readability and may also cause undesired results. As a result, their include provides more clarity to the reader and ensures that all contents within their declaration are performed.

```
                if (t->temperature >= 58) {
                    hightemps++;
                }
```

Functions

MISRA C Rule 17.2, NASA Rule 8, ISO 26262-6:2018 1(j) – Recursion should be excluded, as its repetition may occur in an unexpected and unwanted amount of executions.

NASA Rule 4 – Function exceeds page limit. This rule states that a function should not be longer than a single sheet of paper, which equates to approximately 60 lines of code. As such, the scale of main() has been reduces as it was initially 63 lines in length.

MISRA C Rule 17.7, NASA Rule 7 – Return values from functions have been checked, and provided with an alternative in the event the desired outcome is not occurred.

```
            if(pthread_create(threads + i, NULL, tempmonitor, &i)
!=0) {
                perror("Entrance thread not created");
                return 1;
            }
```

Pointers and arrays

MISRA C Rule 18.4 – Removing +, -, += and -= operators applied to an expression of pointer type is considered ‘Advisory’. As such, it’s implementation has been noted however not replaced as it is not regarded as ‘Required’.

NASA Rule 9 – Function pointers not permitted, however unfortunately there was no way for us to rectify this occurrence, however its safety critical impact has been noted.

Standard libraries

MISRA C Rule 21.3, NASA Rule 3 – Dynamic memory allocation such as malloc() may not be used, however unfortunately there was no way for us to rectify this occurrence, however its safety critical impact has been noted.

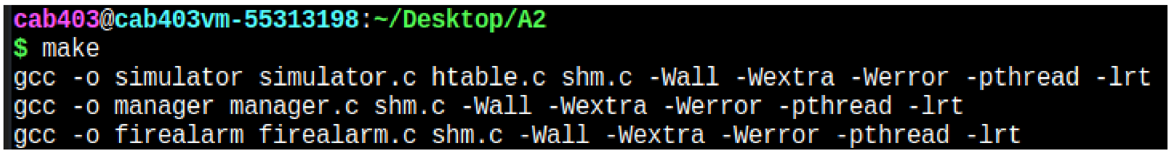
MISRA C Rule 21.6 – The inclusion of <stdio.h> has not been omitted from our project, despite this rule being noted. The justification for its usage is that is was noted within the unit as being permitted.

MISRA C Rule 21.9 – The function for “qsort” should be excluded. This could be replaced with the algorithm for qsort hardcoded as opposed to relying on a function from an external library.

MISRA C Rule 21.10 - The Standard Library <time.h> has been excluded to adhere to safety critical standards, as time and date functions can result in undefined values.

Compilation of all files

NASA Rule 10 – Code has been complied with all warning, inclusive of "-Wall, -Wextra, and -Werror", prior to execution of code. The inclusion of these flags ensures that all errors and warnings have been identified and rectified if required. As such, the final implementation results in the following:



```
cab403@cab403vm-55313198:~/Desktop/A2
$ make
gcc -o simulator simulator.c htable.c shm.c -Wall -Wextra -Werror -pthread -lrt
gcc -o manager manager.c shm.c -Wall -Wextra -Werror -pthread -lrt
gcc -o firealarm firealarm.c shm.c -Wall -Wextra -Werror -pthread -lrt
```

Video

See attached “CAB403.mp4” embedded within submission or alternatively, view via the following URL (Access granted to QUT email addresses):

https://connectqutedu-my.sharepoint.com/:f:/r/personal/n10731300_qut_edu_au/Documents/CAB403?csf=1&web=1&e=AegPbM

Reference List

ISO. (2018). Road vehicles – Functional safety – Part 6: Product development at the software level. *ISO*. https://www.synopsys.com/content/dam/synopsys/sig-assets/whitepapers/meeting-iso-26262-software-standards.pdf

MISRA. (2019). MISRA C:2012. Guidelines for the user of the C language in critical systems. *MISA*. https://misra.org.uk

NASA. (2006). The Power of 10: Rules for Developing Safety-Critical Code. *MISA*. http://www.cs.otago.ac.nz/cosc345/resources/NASA-10-rules.pdf

QUT. (2022). CAB403 Systems Programming - Assignment 2. *QUT*. https://blackboard.qut.edu.au