COMP0002 C Coursework Marksheet

The criteria for the mark ranges used for the overall coursework mark:

Inadequate (0-39)	Failed to clearly demonstrate a minimal understanding of programming in C. There are fundamental errors, code will not compile, or nothing of significance has been achieved. (F)
Just Adequate (40-49)	Shows a basic understanding, sufficient to achieve a basic pass, but still has serious shortcomings. Code will compile but doesn't work properly. (D)
Satisfactory (50-59)	Reasonable understanding but with some deficiencies. The code compiles and runs. This is the default range for a straightforward answer to at least the basic stages. (C)
Good (60-69)	A good understanding, maybe some minor issues, but otherwise satisfactory. The code compiles, runs and demonstrates reasonable design practice. Most expectations have been met and most stages completed. (B)
Very Good (70-79)	A very good understanding comfortably above the average expectations, demonstrating a clear proficiency in design and programming. All stages completed. (A)
Excellent (80-100)	Excellent design and programming at a level well above normal expectations, demonstrating deep understanding in all aspects. All stages completed very proficiently. This level is used sparingly only where it is fully justified. (A+, A++)

The rubrics table below shows your relative level of achievement for various criteria. In each box with ticked with a 'X' *one or more* of the items apply to your coursework (not necessarily all the comments). To see how to improve look at the items to the right on the same row.
Note: the 'X's are indicative only. They do *not* add up to generate your mark.

Feature	Inadequate (<40)	Just Adequate (40-49)	Satisfactory (50-59)	Good (60-69)	Very Good/Excellent (70+)
Formatting and presentation of source code	Inconsistent or poor formatting in much of the code. Badly presented. Little or no organisation. □	Formatting is careless and has inconsistencies. Not presented that well. Presentation is haphazard in places. □	The code is mostly properly formatted. Reasonable presentation. Code is organised well enough. □	The code is consistently properly formatted. Clear structure and presentation. Generally good practice. □	The code is entirely properly formatted. Good, clear structure and presentation. Clearly demonstrates very good practice. □
Use of comments	Any comments present are random and unhelpful. Comments that are present are unnecessary, adding nothing of value. The use of comments has not been understood. No comments present but they are needed to make better sense of what the code is meant to be doing. The code is poorly written and lacks readability. □	Weak use of comments. Some comments may be inconsistent, incorrect, out of date, or not relevant. Some comments simply repeat what the code itself expresses or add little or no value to understanding the code. No comments but the code itself needs significant improvement to make it more readable. □	Mostly acceptable comments but some may be unnecessary. Not all comments are clear. No comments, the readability of the code is just about good enough but could be improved with comments added in places. □	Reasonable use of comments but some may still be unnecessary or unclear. The role of comments is understood quite well. No comments, as they are largely not needed, and the readability of the code is quite good. Adding carefully chosen comments in a few places might be helpful. □	Good commenting, kept brief but relevant, adding information to the source code. The role of comments is very well understood. No comments but they are not needed as the code is very readable and easy to understand. □

Use of the programming language	A clear lack of understanding, as the code either doesn't compile and/or run. Language constructs are being misused. □	A basic understanding, enough to get a program that at least compiles and partly runs. Not using the language that well. □	Reasonable use of the language. Shows satisfactory use of the language features. □	Good use of the language. A quite good understanding of how to use it properly. Language generally used correctly. □	Very good understanding of the language. Always used correctly. □
Use of functions	Functions used minimally or not at all, undermining the structure of the program. The role of functions is not understood. Most code in one function. □	Not used enough functions to provide adequate structure. Some functions are too long, not cohesive, or poor abstractions. Some or all functions don't have suitable parameters. □	Reasonable use of functions. Mostly short and cohesive, but some may be too long and not good abstractions. Parameters generally used properly, but review carefully. □	Good use of functions. Mostly short and cohesive, and satisfactory abstractions. Parameters generally used appropriately. □	Very good use of functions. Use of functions fully understood, good parameters Function length, cohesiveness, and good use of abstraction has been achieved. □
Use of variables	Variable used in an ad hoc way with no clear organisation or scope. Poor naming. Incorrect types used. All at file/global scope. Local and parameter variables not understood. □	Variables used just about adequately. Not always paying attention to scope, good naming or use of types. Too many variables defined in file/global scope, some should be local. □	Reasonable use of variables. Scope, names and types mostly valid but review carefully. □	Good use of variables. Scope, names and types generally valid. □	Very good use of variables. Valid names and scopes. Types well understood. □

Pointers and memory allocation if used. No 'X' in this row simply means these features are not used.	Not understood and misused. Dynamic memory not correctly allocated or freed. Incorrect pointer arithmetic. Segmentation faults occur. □	Pointers used but with some mistakes. Pointers not needed in some cases. Dynamic memory allocation not always valid or freed correctly. Pointer arithmetic may not be correct. One or more segmentation faults caused by incorrect pointer use may occur. □	Pointers use generally valid but with some issues or confusion over the use of pointer operators. Pointers used but not really always necessary. Dynamic memory generally allocated and freed correctly, but possibly not always. Pointer arithmetic is valid. No segmentation faults. □	Pointers generally used correctly. Dynamic memory generally allocated and freed correctly. Pointer arithmetic is used correctly. No segmentation faults. □	Pointers understood and used very well. Dynamic memory always allocated and freed correctly. Pointer arithmetic is always valid. No segmentation faults. Pointers and dynamic memory used in ways that enhance the code design and efficiency. □
Arrays.	Not understood and misused. Incorrect indexing. Not used for anything sensible. □	Arrays used but with some issues. One or more indexing problems. Trying to access arrays via pointers but with significant issues. □	Array use generally valid but quite straightforward. Arrays used somewhat clumsily in places. Indexing valid, some index expressions might be improved. Using pointers with arrays, mostly valid but with some issues. □	Arrays generally used correctly. Indexing correct. Using pointers with arrays correctly, valid dynamic memory management. Use of arrays improves the code. □	Arrays used properly and are well understood. Indexing always correct. Combining array access with pointers and dynamic memory management properly. □

Quality of Coursework Answer	Very poor, not done enough to justify a pass mark. Arena or robot not displayed properly. Robot does not move correctly. Lacks understanding of how to design a program properly. □	Adequate, an arena is displayed and a robot shown moving. Uses a pre-determined route or basic movement algorithm. Design is basic and needs significant improvement. □	An arena is displayed, robot moves around. Adequate progress on finding an algorithm for the robot to navigate the arena and find markers. The design is satisfactory. □	Good, an arena is displayed and the robot moves correctly. Uses a reasonable algorithm to find markers. The design and coding are quite good. □	Very good, an arena can be generated and displayed effectively. A robust algorithm moves the robot around to find markers. The design and coding are very good. □
Overall	Some serious deficiencies, not submitted a working answer. You need to spend significant time practicing C programming. It is recommended that you read at least one textbook (see reading list on Moodle). □	Understood well enough to write some basic working code. Still significant gaps in your knowledge. Invest time in more C programming practice and find a good textbook to read. □	You can write some reasonable working code. There are some gaps in your knowledge and understanding, but overall you have an adequate understanding of C programming. Keep practicing! □	You can write good working code. You've got a good understanding of C programming. Keep practicing! □	You are a very good programmer. Writing good quality code. As always, it is well worth the time to continue reviewing and improving your coding. □

Additional Feedback:

Mark: