CAB202 Assignment 1, Exercise 1: Assignment 1

Return to Exercise List »

Results so far:

No submission has been made so far. (0%).

Assessed weight: 0%. You may continue to attempt this item until you reach 30 submissions. So far, you have made 0 submissions.

Requirements:

CAB202 Assignment 1

Due Date:	29 April 2019 23:59:59
Submission:	Attach your solution using the file attachment controls provided below.
Assessment Weight:	30%
Change Log:	The following changes have been made since the initial release of the assignment. • 28 Mar 2019 – Adjusted shape and dimensions of trash. • 28 Mar 2019 – Modified rule for dropping garbage. • 28 Mar 2019 – Clarification of battery definition. • 28 Mar 2019 – Added command to override battery level. • 28 Mar 2019 – Modified load display criterion. • 28 Mar 2019 – Allow up to 1000 pieces of dust. • 28 Mar 2019 – Removed Room 2 and Room 3. There is only one room in this simulation. • 28 Mar 2019 – Add controls to push the device one unit north, south, east, or west. • 29 Mar 2019 – Add help screen. • 04 Apr 2019 – Eliminated all references to furniture. Furniture was in Room 3. Room 3 no longer exists. Therefore furniture no longer exists. • 04 Apr 2019 – Tweak wording of charging station location.

Introduction

In this assignment you will design, implement, and test, a simulated Robotic Vacuum Cleaner, hereafter

referred to as *the device*. The simulation consists of a simple virtual environment within which the device moves under a rudimentary autonomous control algorithm, collecting particulate matter (also called *rubbish*) from the floor as it passes. Rubbish comes in three varieties: *dust* is made up of small particles; *slime* is made up of medium sized particles; *trash* is composed of large particles. When the device reaches its maximum load-carrying capacity, it travels to a charging station where it unloads all accumulated waste matter, and tops up its battery. Having done so, the device resumes its patrol, continuing until either the simulation is terminated via a command from the user, or the simulation exceeds a designated maximum time limit.

Listen Up!

1. The assignment will be graded by strict reference to the criteria listed below.
2. **This is not a group assignment.** While we encourage you to discuss the assignment and brain-storm with your associates, you must ensure that your submission is your own individual work.

   Share ideas, not code!

3. A high standard of academic integrity is required. Breaches of academic integrity, including plagiarism or any other action taken to subvert, circumvent, or distort the assessment process, will not be tolerated. QUT policy regarding academic conduct is available in the QUT MOPP Section C/5.3 Academic Integrity (http://www.mopp.qut.edu.au/C/C_05_03.jsp). In particular, under the provisions of MOPP statement C/5.3.7 (http://www.mopp.qut.edu.au/C/C_05_03.jsp#C_05_03.07.mdoc), Part (e), we reserve the right to require you to authenticate your learning. You may be required to show evidence of materials used in the production of the assignment such as notes, drafts, sketches or historical backups. You may also be required to undertake a viva or complete a supervised practical exercise.
4. Use of any third-party code library (other than the standard libraries supplied with GCC, ncurses, and the list of items labelled as Exceptions in the following paragraph) is strictly prohibited. Use of code downloaded from the internet, with or without correct attribution to the original author(s), is strictly prohibited. Submission of source code created by teaching staff of this unit in any previous semester is strictly prohibited. Subcontracting, outsourcing, off-shoring, purchasing, borrowing, stealing, copying, or obtaining source code by any means other than through an act of original creation by yourself, is strictly prohibited.

   Exceptions: you are strongly encouraged to call functions defined in the **current version** of the ZDK, as downloaded from CAB202 2019 Semester 1 Blackboard Learning Resources on or after 25 February 2019. You are strongly encouraged to download and submit copies – modified as appropriate – of the files `helpers.c`, `helpers.h`, and `zdj2.c`, from CAB202 2019 Semester 1 Blackboard Learning Resources Topic 4.

5. Do not post your solution in any form of online repository or file sharing service, or allow anybody else to obtain access to your solution in any way. Doing so will be classified as academic misconduct under the clauses pertaining to collusion, especially in the event that a copy of your source code obtained from such a service is submitted by another student, regardless of whether this has occurred without your knowledge and permission.
6. Abundant code samples, demonstrations, and exercises have been made available to support your effort toward this programming task. Written permission must be obtained from the Unit Coordinator if you

want to use technology other than the ZDK to implement your game. Permission will only be granted if there are compelling special circumstances that make it impossible for you to use the ZDK. Without this permission, a game implemented with some other graphical framework will receive a score of 0. Direct use of the `ncurses` library to render graphics or text is expressly prohibited.

Breaching any of the foregoing conditions will result in an immediate and final score of 0 for the Assignment.

Deliverables

Submit the following items:

- Implementation:
  - Submit all source files (`.c`) and header files (`.h`) required to compile and run your program.
  - In addition to source and header files, you are required to submit
    - A `bash` script or makefile which can be used to build your program in a typical CAB202 programming environment.
    - A `bash` script which embodies and automatically executes your test suite. The details of the contents of this script are set out under the heading Test Requirements, below.
  - Use the controls at the bottom of this page to attach source files, header files, and your bash test script.
- Architectural requirements:
  - Your program must implement the core high level architecture demonstrated in lectures. You are encouraged to use the implementation of the Zombie Dash Junior case study from Topic 4 as a launching point. A minimal skeleton for the program follows:

```
#include <math.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>
#include <cab202_graphics.h>
#include <cab202_timers.h>
// Insert other functions here, or include header files
void setup () {
    // Insert setup logic here
}
void loop() {
    // Insert loop logic here.
}
int main() {
    setup_screen();
    setup();
    while ( /* Insert termination conditions here */ ) {
        loop();
        timer_pause( /* Insert delay expression here. */ );
    }
    return 0;
}
```

Functional Specification Overview

The simulated machine will exhibit the following general capabilities:

- It will detect collision between itself and other fixed objects such as walls and charging station, and respond appropriately.
- It will have a direction sensor which it will use to navigate, by turning relative to its current direction, or by travelling towards a charging station.
- An in-built scale will enable the device to measure the weight of its current payload, and adjust its behaviour accordingly.
- A battery level sensor will enable the machine to monitor its current power status, to ensure that it

returns to its charging station when power runs low.

A basic graphical interface, rendered using the CAB202 ZDK graphics library, will show the progress of the robot vacuum cleaner as it navigates the virtual environment. The operator of the simulation will use the user interface to monitor the state of the device via a status panel, observe the effectiveness with which the machine cleans the floor, and verify that the machine generally conforms to specification as set out in detail in the remainder of this document.

The simulation will execute in two modes. When running in *interactive mode*, a human observer (e.g. your tutor) will control the simulation using instructions listed below. When running in *batch mode*, the simulation will be executed with a list of instructions provided via redirected standard input, using the same instruction set as those used by the human observer.

The instructions recognised by the simulator are listed below. Some of these instructions are atomic, but several require the input of one, two, or three numeric parameters, as elaborated in greater detail in the following sections. The Robot Vacuum Cleaner instruction set is listed below:

- `'b'` – tell the device to stop cleaning and return to base i.e. step repeatedly towards the charging dock.
- `'d'` – drop a piece of *dust* (small rubbish) on the floor.
- `'i'` – push the device one unit north (up on the terminal).
- `'j'` – push the device one unit west (left on the terminal).
- `'k'` – push the device one unit south (down on the terminal).
- `'l'` – push the device one unit east (right on the terminal).
- `'m'` – set the millisecond delay between successive invocations of the `loop` function.
- `'o'` – set the time-out period, that is, the total permitted run time, measured in seconds.
- `'p'` – cause the device to start moving (if it is stationary) or stop moving (if it is mobile).
- `'q'` – quit simulation.
- `'r'` – reset simulation.
- `'s'` – drop a piece of *slime* (medium rubbish) on the floor.
- `'t'` – drop a piece of *trash* (large rubbish) on the floor.
- `'v'` – move the device to a new location and specify a new direction.
- `'w'` – change the amount (weight) of the rubbish currently on board the device.
- `'y'` – change the battery level of the device.
- `'?'` – display help screen.

A video which demonstrates a near-complete version of the assignment is available for viewing in the Echo360 panel within Blackboard. The only functionality absent is pixel-level collision detection.

Functional Specification

1. **Terminal Set-up:** you may assume that the terminal will be quite large, for example, on the order of 150×50, or more. Tutors will not run the simulation in a tiny display.
2. **Status Display:** The status display is 5 characters high, located at the top of the terminal window, is entirely visible at all times (i.e. never occluded in any way) and displays the following information in a tabular format, laid out as two rows, each having three columns.

   i. Row 1, column 1: Student number, in the form `n9801154`.
   ii. Row 1, column 2: Robot direction, measured in degrees and expressed as a whole number between 0 and 359 inclusive.
   iii. Row 1, column 3: Remaining battery life, expressed as a percentage, using whole numbers.
   iv. Row 2, column 1: Elapsed time since last reset, measured as whole number minutes and seconds in the form `mm:ss`.
      - Use `get_current_time` to measure elapsed time since last reset, rather than attempting calculations based on the loop delay.
      - This must update in real time.
   v. Row 2, column 2: ~~Weight, expressed as a percentage of the nominal load-carrying capacity of the device, using whole numbers.~~Load weight, measured in grams.
   vi. Row 2, column 3: Rubbish available: three integers, separated by commas, showing the current amount of dust, slime, and trash, respectively, waiting to be collected from the floor.
3. A **command input area** occupies the bottom two rows the terminal window, remaining visible at all times (i.e. never occluded in any way by vacuum cleaners or rubbish).
   i. When not in use, this region will be empty.
   ii. During interactive command entry, the region is used to prompt for and echo character data obtained from standard input.
4. **Border lines:** The terminal window as a whole is outlined by a border, and the various parts of the display within are separated visually by horizontal and vertical lines.
   i. Vertical line segments, including the left and right edges of the room and the status display, are rendered using lines made of `'|'` (pipe) symbols.
   ii. Horizontal line segments, including the upper and lower boundaries of the, and horizontal rules in the status area, are rendered using lines made of `'-'` (minus) symbols.
   iii. Intersections between horizontal and vertical border lines should be marked by `'+'` (plus) symbols.
   iv. Border lines are visible at all times and never occluded by any object.
5. **Robotic Vacuum:** There is a Robotic Vacuum device.
   i. The device is represented by a distinctive icon which must be at least 9×9 characters in size.
   ii. The icon is approximately circular in shape.
      - Due to the highly pixellated display, a smooth circle is impossible.
      - Any reasonably smooth regular polygon with 8 or more sides will be satisfactory.
      - As the “pixels” are unlikely to be square, the aspect ratio will be distorted, so that circles will look more like ellipses. This is expected, and acceptable.
   iii. The outline of the icon is rendered with `'@'` (at) symbols.
      - You may use any pattern to fill the interior of the icon, however if space characters are used they must be opaque, so that objects pass out of view when the device moves over them.
   iv. Whenever the simulation is reset, including when it first starts, the icon is placed in the middle of the room.
      - This is a rigorous requirement. To be precise, the device is in the middle of the room if and only if the centre of the smallest possible axis parallel rectangle which encompasses all characters of the rendered image of the device (with visible pixels falling on or within its

perimeter) falls within one screen unit (horizontally and vertically) of the centre of the room.

v. Whenever the simulation is reset, including when it first starts, the device is stationary. It does not move until an explicit instruction is received to make it commence moving.

6. **Default Device Behaviour:** The default behaviour of the Robot Vacuum device is as follows:
   i. When initially set in motion by pressing a `'p'` command, it moves directly downwards to the bottom border, at a speed of 0.2 screen units per time slice. This direction may be displayed as 0 degrees, or it may be displayed as 90 degrees, depending on the coordinate system you have chosen to use.
   ii. Whenever the device hits a wall or other obstacle such as the charging station, it will pause for one time slice, and in that time it will change direction by swivelling either left or right by a random angle between 30 and 60 degrees.
      - Depending on the angles involved, this may mean that the device pauses for several time slices. However, it will (with very high probability) eventually move away from the wall.

   iii. The device must never be seen to cross over or intersect the wall in any way. Nor may it: leave the terminal window in the normal course of running the simulation; invade the status area; or traverse into the area set aside for input.
   iv. If the device is moving and a `'p'` command is encountered, it stops moving. If it is not moving and a `'p'` command is encountered, it starts moving. Apart from stopping and starting, there should be no other side effects: all other aspects of the state of the simulation must remain unchanged.
   v. The device must never move more than 1 character distance in any direction in a single step, unless repositioned via the `'v'` instruction.
7. **Battery Life:** The device has an initial battery life of 100% which decreases by 1% per second.
   i. Use `get_current_time` to measure passing time as the battery discharges, rather than attempting calculations based on the loop delay.
   ii. If the battery life reaches zero, the device becomes immobile as it has no power reserve. See further notes below.
   iii. The battery life must never be observed to have a value less than zero, or greater than 100%.
8. **Charging station:** The device docks with a charging station to recharge its battery and unload any rubbish it has collected since the last visit.
   i. The charging station is represented by a distinctive image at least 9 characters wide and 3 characters high.
   ii. The outline of the charging station must be rendered with `'#'` (hash) symbols.
   iii. The charging station must be positioned in the middle of the northern edge of the room, immediately below the top border.
   iv. If the device is not in *Return to Base* mode, then the charging station should be treated as a wall.
   v. The Robot Vacuum must never be seen to visibly overlap the charging station.
9. **Return To Base:** This mode is activated via the `'b'` command, when the battery level drops below 25%, or when the total current weight of on-board rubbish exceeds 45g.
   i. In *Return to Base* mode, the device constantly monitors its location relative to a charging station, and changes direction to ensure that each move will take it toward the charging station by the

most direct route possible.

ii. While returning to base, the device passes over the top of rubbish without collecting anything, even if it has capacity to collect more rubbish.

iii. If the device collides with the charging station while in *Return to Base* mode, then it should dock by stopping adjacent to the station.

iv. While the device is docked:

- The word “Docked” must be displayed in the command input area.
- The Battery Life indicator should show that the devices is recharging.
- Charge should accumulate at a rate equivalent to a full charge (100%) in 3 seconds.
- Use `get_current_time` to measure passing time during recharge, rather than attempting calculations based on the loop delay.
- Once docked, the device remains docked until it is fully charged.
- When the Robot Vacuum is fully charged it should leave the dock by reverting to its default movement regime. This should allow it to make its way away from the charging station as a result of default collision processing.

10. **Battery Fully Discharged:** If the Robot Vacuum's charge reaches 0% it should stop moving and a Simulation Over message should be displayed.

i. The message should be displayed in the middle of the room, in the centre of a clear rectangle which permits it to be easily seen and deciphered.

ii. Leave existing images in the room, apart from those covered by the clear rectangle.

iii. The message should give the user the option of pressing 'q' to quit the simulation gracefully.

iv. The message should give the user the option of pressing 'r' to reset the simulation to its initial state.

v. Pending reset, the status indicators should remain visible, frozen in their state at the time the charge hit 0%.

11. **Load Carrying Capacity:** The device is capable of carrying rubbish with a total mass of 65g.

i. The current load of the Robotic Vacuum is reassigned to 0 when the simulation is reset (including when it first starts).

ii. To ensure that the load never exceeds this figure, the device is programmed to turn off its vacuum pump as soon as the total current load exceeds 45g, and transition to **Return to Base** mode.

iii. As long as the current total load is equal to or less than 45g, the device will continue to collect rubbish.

12. **Rubbish:** the simulation allows a number of items of rubbish to be distributed onto the floor. As noted elsewhere, there are three categories of rubbish: *dust*, which is small; *slime*, which is medium in size; and *trash*, which is large.

i. There are three categories of rubbish:

a. **Dust:** has a mass of 1g, and at any given time there may be up to ~~20~~ 1000 pieces of dust on the floor. Dust is represented by an image consisting of a `'.'` (point or full stop) symbol.

b. **Slime:** has a mass of 5g, and at any given time there may be up to 10 patches of slime on the floor. Slime is represented by an approximately elliptical 5×5 image consisting of `'~'` (tilde) symbols.

c. **Trash:** has a mass of 20g, and at any given time there may be up to 5 piles of trash on the

floor. ~~Trash is represented by an approximately equilateral triangle, with sides 7 units long,~~ Trash is represented by an isosceles triangle with base at least 11 units long, and height at least 6 units, made of `'&'` (ampersand) symbols.

ii. When the simulation is reset (or when the simulation starts for the first time), the system should use the command input area to obtain the number of pieces of dust, slime, and trash to spread on the floor, up to and including the limits listed above.

iii. Rubbish should not spawn on (or under, or overlapping) the Robotic Vacuum, overlapping with any other item of rubbish, overlapping the charging station, or overlapping with the outlines of the room.

iv. At all times, the current number of dust, slime, and trash, waiting of the floor to be cleaned up is displayed in the status area.

13. **Rubbish Collection:** Rubbish is collected by the device when it is not in **Return to Base** mode, and the icon of the Vacuum collides with the icon of the rubbish.
    i. The rubbish should disappear from the floor immediately. This is a very strong vacuum cleaner.
    ii. The weight of the device should be updated in the status display immediately to reflect the added payload.
    iii. The available rubbish indicator should update immediately, to reflect the removal of the rubbish.
14. **Collision Detection:** Two levels of attainment are possible:
    i. Bounding box collision detection will allow credit for most of the functionality to be obtained.
    ii. Pixel level collision detection will yield a higher score, as it will enable more accurate collision between the device and other objects such as rubbish or the charging station.
15. **Other simulation controls:** In addition to those already covered, the simulation should respond to the following commands.
    i. `'d', 's', 't'` – use the input command region to read an (x,y) coordinate pair, and move a piece of rubbish of the appropriate type into that location.
        - ~~The rubbish moved must already be on the floor waiting for collection.~~
        - ~~If there is no rubbish waiting for collection of the specified type, then the simulator should ignore the command.~~
        - This operation drops a new piece of rubbish on the floor.
        - If the maximum number of permitted items of the selected type are already present, then the simulator should ignore the command.
    ii. `'i', 'j', 'k', 'l'` – push the device one unit towards the north (up), west (left), south (down), east (right), respectively. All other constraints must continue to be satisfied. If the device is engaged in normal movement and it is pushed, then it should continue its previous movement after being pushed. If the device is returning to base when it is pushed, it should re-orient and continue to return to base after being pushed.
    iii. `'m'` – Ask the user for a new delay period for use in the main event loop. Refer to Topic 4 lecture notes.
    iv. `'o'` – set the time-out period, that is, the total permitted run time, measured in seconds. Refer to Topic 4 lecture notes.
    v. `'q'` – Display a farewell message, wait for user input, then exit the simulation
    vi. `'r'` – Reset simulation. The Robot Vacuum, rubbish, and status display are all restored to their

initial state as described above.

vii. `'v'` – Ask the user to supply new x, y, and angle settings for the device. Assign these values into the state of the object, and resume. If the devices is operating in *Return to Base* mode at the time it is moved, then it must automatically realign on a charging station and resume its journey back from the new location.

viii. `'w'` – Ask the user to enter a new, positive value for the weight. The current payload weight is updated to the new value. This allows the behaviour of the device to be efficiently tested under a range of conditions (empty, full, in between) to verify that all trigger criteria are detected properly.

ix. `'y'` – Ask the user to enter a new battery level, between 0 and 100 inclusive. The battery level of the device is updated to the new value. This allows the behaviour of the device to be efficiently tested under a range of conditions (low charge, full charge, and near the cut-off for *Return to Base* behaviour) to verify that all trigger criteria are detected properly.

x. `'?'` – Suspend all operations in the simulation and display a simple help screen which lists all available commands with a brief synopsis. The screen should prompt the user to press a key to resume, then wait until the user presses a key, or until the execution time-out limit is reached. When the help screen closes, the simulation should appear exactly as it was, apart from the elapsed time indicator, which should include the time spent viewing the help screen.

Program structure, implementation quality

The program must be implemented with a view to ease of comprehension and maintainability. To this end:

1. The program should be subdivided into loosely coupled functions. Wherever possible and practical, data is transferred between functions via parameters and return values rather than shared global variables.
2. No function may contain more than 25 executable C language instructions. Note that comma separated expressions with side effects will be classified as executable instructions.
3. No function may be substantially identical to any other function.
4. All functions have carefully and meaningfully narrated documentation comments.
5. All source code must be formatted in a professional manner, making sensible use of meaningful identifiers, and adhering to a consistent coding standard of your choice.
6. Groups of related functions should be sectioned out into separate modules. Use of global variables is restricted to the compilation unit within which it is declared.
7. Template shell scripts and makefiles will be provided as tutorial resources during Week 5.
8. Evidence of correct use of language support for modularity must be provided via the shell script or makefile which you submit with your implementation. the existence of multiple source files by itself is not sufficient to obtain full marks under this criterion.

Test Requirements

The program must be rigorously and methodically tested, using an automated test suite which you will implement using a shell script.

- The script should be able to run without problem in a standard Linux or Cygwin terminal, such as MinTTY using the bash command interpreter.

- Each test must be documented in the shell script, using in-line comments to address:
    - What item of functionality (from the numbered list above) the test exercises.
    - What the expected outcome from running the test should be.
    - What the behaviour observed in your program is.
    - An executable command which includes the explicit input sequence required to run the test, and which executes the test automatically when the shell script runs.
- Template shell scripts for this purpose will be provided as tutorial resources during Week 6.
- Failure to provide an executable test suite following the format set out in the template will result in a score of 0 for this section.

Marking

The assignment is worth 30% of your total grade in this subject, and it is marked out of 30. A detailed marking rubric will be published at least 14 days before the submission deadline. The approximate breakdown of marks will be:

- Functionality: 12 marks.
- Testing: 12 marks.
- Program structure, implementation quality: 6 marks.

The following points should be noted about marking:

1. **If your code does not compile when submitted to AMS, the mark awarded will be 0.**
    - Give yourself plenty of time to get the basics right.
    - Submit Early. Submit Often.
2. **If the program has been implemented via a framework other than the ZDK without prior written permission from the Unit Coordinator, the mark awarded will be 0.**
3. If the program fails with segmentation faults or other fatal errors, marks will be awarded for the features that were observed prior to the crash.
    - No effort will be made to work around the crash, nor will we debug your code to make it compile or run.
    - Your program must compile and run well on any modern desktop machine equipped with a Unix-like environment.
    - **To this end, part of your job is to test the program in a variety of environments, including (as a minimum) QUT lab machines using the CAB202 portable Cygwin environment.**
    - “It runs fine on my computer!” may be true, but it is largely irrelevant if the program crashes when executed on another machine. Such an excuse will not be accepted.
4. It is your responsibility to implement each feature sufficiently well that it is readily detected through normal operation of the game. Any feature that is not apparent through normal play will be deemed to be unimplemented.
5. We require tutors to adhere to a strict time limit of 3 minutes run time when marking each submission. Any feature that cannot be assessed in that time will be deemed to be unimplemented. Therefore you must avoid defects in game dynamics such as extremely fast motion, extremely slow motion, inconsistent control settings, premature program termination, or other properties that render the game

unfit for use.

6. It is better to implement some of the features extremely well than to try to do everything and deliver a substandard product.
7. Penalties will be applied if the code exhibits general defects or undesirable behaviour not otherwise covered in this document. This includes but is not limited to such things as:
    - Errors in object motion, such as objects jumping more than one character position per frame;
    - Objects moving outside their permitted area;
    - Failure to clear the screen appropriately between updates;
    - Collisions involving invisible or non-existent objects;
    - Disturbing or flashing display;
    - Sluggish response times, excessively fast response times, or other performance defects which render the simulation difficult to operate according to specification.
    - Gratuitous errors in program structure and organisation, including but not limited to: inappropriate use of recursion; use of inappropriate data structures such as linked lists or binary search trees; using `#include` to insert the contents of source files rather than header files; uploading of multiple versions of the same source file.

Notes:

- *For purposes of AMS assessment, this activity has been classified as non-assessed. This does not mean the assignment is non-assessable – it means that the system is not able to assess the assignment automatically. AMS will not enforce a hard close-down for submissions, however in line with QUT policy, late submissions without an approved extension will not be marked. If special circumstances prevent you from meeting the assessment due date, you can apply for an extension (https://www.student.qut.edu.au/studying/assessment/late-assignments-and-extensions)* ***before the assignment due date****. If you don't have an approved extension you should submit the work you have completed by the due date and it will be marked against the assessment criteria.*
- Use "Attach file" to attach *a single version* of each source file.
- Files with names ending with a pattern other than `.c`, `.h`, or `.bash`, *will be rejected*. Please ensure that your naming convention takes this limitation into account.
- A maximum of 30 files may be included in each submission. This limit is fixed and there is no valid reason to request that the system be re-engineered to accept a greater number of attachments.
- When you have attached all required files, press the "Submit" button.
- Source files for each submission will be placed in a single distinct folder on the server, and compiled with the following command:

```
gcc *.c -std=gnu99 -Wall -Werror -I../ZDK -L../ZDK -lzdk -lncurses -lm
-o a1_n9801154
```

  Note that the wild-card pattern `*.c` will compile *all* files with a name that ends with the extension `.c` in the present working directory.

Therefore, you should organise your files in a folder alongside the ZDK folder and use this command to build your solution. Make sure there are no additional files (such as old versions of your program) in your work folder which may cause unpredictable build errors and/or run-time behaviour.

It is your responsibility to use the command set out above to build the project on your own computer to verify that it will compile without problems.

If you are using the `#include` directive to connect multiple C source files, the `#included` files should be renamed to have an extension of `.h` to avoid errors arising from multiple definition of functions. However, in general you should not include anything other than `.h` files in the first place.

- If compilation is successful, AMS will verify that your program has compiled successfully, and then return. Your program will *not* be executed because there is no meaningful test that can be performed automatically on a program such as this. The score allocated by AMS for this task is not your mark for the assignment.

Submitted files:

Attach file:

Browse... No file selected.

Use the file chooser to attach a source file.

Declaration and submission

By submitting this form, I certify that:

- I have read, and understand, QUT Manual Of Policy and Procedures, Section C/5.3, Academic Integrity; and
- This submission is in full compliance with all provisions of QUT Manual of Policy and Procedures, Section C/5.3, Academic Integrity; and
- With the exception of support libraries provided to the class by the CAB202 teaching team, I am the sole author of all source code and attachments included in this submission.

Agree to these conditions: ☐

Submit

Transcript:

© 2019 - Queensland University of Technology