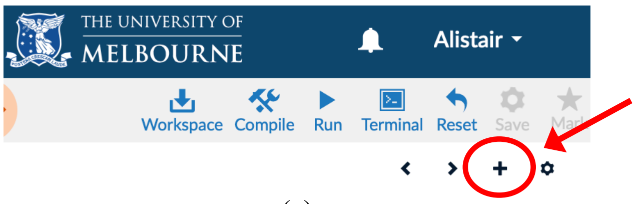
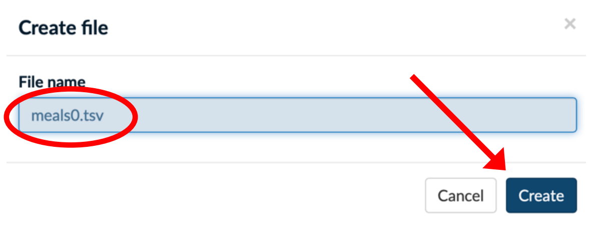
comp20005 (semester 1) / comp10002 (semester 2)
Running Programs in a Shell Within grok

To execute and test your assignment within grok, follow these steps:

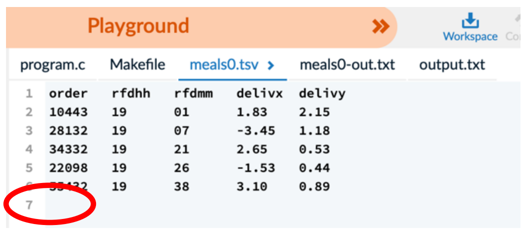
Zero: Create the correct data files and required output files in your grok environment, by (a) clicking the "+" button for each (b) file that you wish to create, making sure that (c) you end the file with a newline character (it will show as a blank line in the grok editor). You can either carefully type the data, or paste it from elsewhere.



(a)



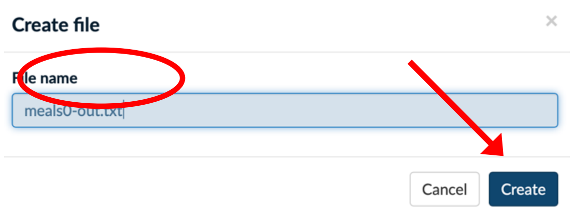
(b)



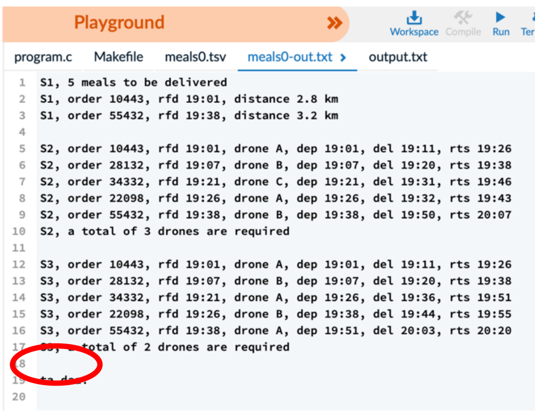
```
order	rfdhh	rfdmm	delivx	delivy
10443	19	01	1.83	2.15
28132	19	07	-3.45	1.18
34332	19	21	2.65	0.53
22098	19	26	-1.53	0.44
55432	19	38	3.10	0.89

```

(c)



(b)



```
S1, 5 meals to be delivered
S1, order 10443, rfd 19:01, distance 2.8 km
S1, order 55432, rfd 19:38, distance 3.2 km

S2, order 10443, rfd 19:01, drone A, dep 19:01, del 19:11, rts 19:26
S2, order 28132, rfd 19:07, drone B, dep 19:07, del 19:20, rts 19:38
S2, order 34332, rfd 19:21, drone C, dep 19:21, del 19:31, rts 19:46
S2, order 22098, rfd 19:26, drone A, dep 19:26, del 19:32, rts 19:43
S2, order 55432, rfd 19:38, drone B, dep 19:38, del 19:50, rts 20:07
S2, a total of 3 drones are required

S3, order 10443, rfd 19:01, drone A, dep 19:01, del 19:11, rts 19:26
S3, order 28132, rfd 19:07, drone B, dep 19:07, del 19:20, rts 19:38
S3, order 34332, rfd 19:21, drone A, dep 19:26, del 19:36, rts 19:51
S3, order 22098, rfd 19:26, drone B, dep 19:38, del 19:44, rts 19:55
S3, order 55432, rfd 19:38, drone A, dep 19:51, del 20:03, rts 20:20
S3, a total of 2 drones are required

ta daa.

```

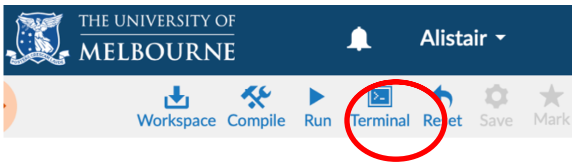
(c)

Then click on "Save" so that the new files become part of the workspace.

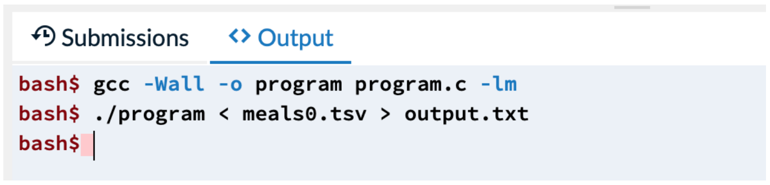
One: Iteratively construct/extend "program.c" using the built-in grok editor, in the usual way.

Two: Compile your program using the "Compile" button. Fix all syntax errors and warnings.

Three: To execute your program and test it, start the inbuilt grok shell by clicking on “Terminal”.

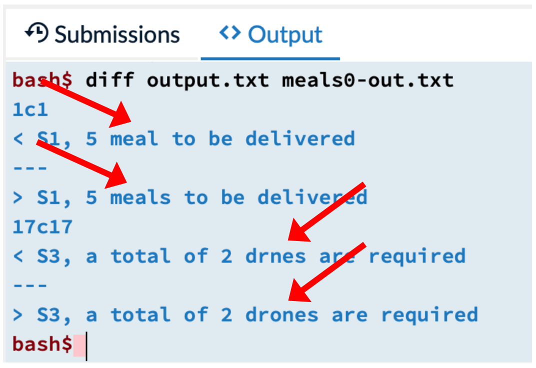


Four: Wait for the “bash$” prompt. Then execute your program as shown to create a file “output.txt”. You can also compile it in the shell if you wish, as shown. You will need “-lm” if you use the math library.



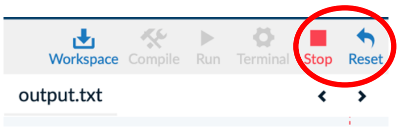
```
bash$ gcc -Wall -o program program.c -lm
bash$ ./program < meals0.tsv > output.txt
bash$
```

Five: Check whether your output is correct by using “diff”, which compares files. In this example (comp20005 in 2021s1) there are two lines that are detected as different because of spelling mistakes. Your goal is to get no differences detected. The “<” lines in the output show what is in “output.txt” (the first argument to “diff”), and the “>” lines show the corresponding lines of “meals0-out.txt”, the second argument.



```
bash$ diff output.txt meals0-out.txt
1c1
< S1, 5 meal to be delivered
---
> S1, 5 meals to be delivered
17c17
< S3, a total of 2 drnes are required
---
> S3, a total of 2 drones are required
bash$
```

Six: Stop the terminal shell by clicking the “Stop” button. You will then be able to edit your program again, to correct any problems. While editing you can also view the file “output.txt” that was created by selecting it in the grok edit menu if you need to check it again.



Seven: Go back to Step One, and iterate until your program obtains the required output (OR until it is five minutes before the submission deadline!) Make new test data too, to cover different scenarios. If you have made a tricky test file that you would like to share, email it to ammoffat@unimelb.edu.au, and I’ll publish your test file and the output my solution program generates.

Note: We are not using grok for assignment submissions. **Final assignment submissions must always be made via the LMS**.