COMP1511 17s1 (http://www.cse.unsw.edu.au/~cs1511/17s1/)

Assignment 2 - Trader Bot

Introduction to Programming (http://www.cse.unsw.edu.au/~cs1511/17s1/)

Introduction

Your task is to write a bot in C. Your bot must successfully buy, transport and sell items in a virtual world. The details of the Trader Bot world are randomly generated for each game. Here is an example:

Name	Type	Commodity	Quantity	Price
CSE	*Start*			
Harvey-Norman	*Buyer*	Televisions	541	1307
MSY	*Buyer*	Computers	333	3033
Caltex	*Petrol station*	Fuel	684	215
Dump	*Dump*			
Coles	*Buyer*	Mars Bars	1401	348
O'Reilly	*Seller*	Books	9830	166
4 Pines	*Seller*	Beer	8050	90
LG	*Seller*	Televisions	2628	677
Lenovo	*Seller*	Computers	307	1666
Aldi	*Buyer*	Mars Bars	3994	343
Quadrangle	*Other*			
BP	*Petrol station*	Fuel	388	111
Mars	*Seller*	Mars Bars	5396	142
J&B Hifi	*Buyer*	Computers	305	3231
Physics Lawn	*Other*			
Apple	*Seller*	Computers	906	1640
Dell	*Seller*	Computers	711	2087
Good Guys	*Buyer*	Televisions	916	950
James Squires	*Seller*	Beer	9584	105
Regent Hotel	*Buyer*	Beer	10345	159
Sony	*Seller*	Televisions	360	1025
Batch Brewing	*Seller*	Beer	8298	85
Addison-Wesley	*Seller*	Books	6441	121
UNSW Bookshop	*Buyer*	Books	2890	342
Umart	*Buyer*	Computers	215	3395
Racecourse	*Other*			
IGA UNSW	*Buyer*	Mars Bars	4403	302
Prentice-Hall	*Seller*	Books	7337	127
Whitehouse	*Buyer*	Beer	6531	165

Your bot has a fixed number of turns in the *Trader Bot* world. Its only goal is to have as much cash as possible after the last turn.

Each turn, a bot is given a description of the world and asked to make a single action.

There are four types of actions: **Move**, **Buy**, **Sell** and *Dump*.

A bot makes money by buying a commodity from one location and selling to another location at a higher price.

For example, it might buy Computers from **Dell** at $2089 each at sell them to MSY at $3033 each.

A bot must move to a location before it can buy or sell from that location.

A bot making a **Move** action specifies the number of locations they wish to move. A bot can move backwards by specifying a negative number of locations. The Trader Bot world is circular so you can move from the last location to the first location and vice-versa.

For example, in the world above a bot at **MSY** which makes a **Move 3** action would go to **Coles**. If instead a bot at **MSY** a made **Move -5** action they world go to **IGA UNSW**.

There is also limit on the maximum number of locations a bot can move in one turn. This limit does not change during the game.

Each location a bot moves will use up 1 unit of fuel. A bot without fuel cannot move (it can perform other actions). A good bot will likely need to purchase fuel.

A bot attempting to make a move that would use up more than its available fuel or exceed the move limit is not penalized. It is moved the maximum possible distance.

Each location buys or sells at most one type of commodity. No location both buys and sells a commodity. Some locations do not buy or sell any commodity.

A bot making a **Buy** action must specify the number of items they wish to buy. A bot may receive less items than it requests. This will occur if:

- the quantity exceeds the number of items the seller has
- the bot has insufficient money
- the items would exceed the bot's weight
- the items would exceed the bot's volume limits.

A bot's cash is reduced by the price of the items it receives in a **Buy** action.

A bot will usually then use one or more **Move** actions to go to a location which buys this commodity. A bot may however choose to buy more items either of the same type or of another type.

A bot making a **Sell** indicates the number of items they wish to sell. A bot making a **Sell** action must be on a location which buys this type of item. The amount that is actually sold may be limited by the amount the buyer wishes to buy. This and the price the buyer will pay is indicated in the location's description.

A bot's cash will be increased by the total price of the items it sells in a **Sell** action.

Bot's are not penalised for attempting to buy or sell more items than is possible. The transaction will be carried out as much as is possible. For example, if a bot attempts to buy 10000 items from a location which only has 25 items, they will be sold 25 items (other restrictions permitting).

Trader Bot worlds will always contain one or more locations of type *LOCATION_DUMP*. At these locations, a bot can make a **Dump** action and all the items it has on board will be removed. The bot receives no cash for these items. A **Dump** action is only useful if a bot has items that can't be sold and wishes to make room for other items.

Trader Bot worlds will always contain one or more locations of type *LOCATION_PETROL_STATION* which sell fuel. Fuel is placed in a bot's fuel tank, it is not included in the bot's cargo. Fuel does not affect a bot's weight and volume limit - there is a separate limit on the amount of fuel a bot can carry. Fuel cannot be sold by a bot. Bots always start with the maximum amount of fuel they can carry.

Trader Bot worlds will always contain one location of type *LOCATION_START*. Bots start at this location.

The initial state of the world is randomly determined at the beginning of each game. The details of the world including prices do not change during the game. The only exception is that the amount of items buyers and sellers are willing to trade reduces as bots buy and sell items.

Multi-bot Worlds

Your bot will be tested in Trader Bot worlds where it is the only bot present.
It will also be tested in Trader Bot worlds where other bots are present.

All bots start at the same location, with the same amount of fuel & cash and have the same number of turns. All bots have the same limits on the amount of fuel they can carry, how far they can move and the maximum volume and weight they can carry.

When multiple bots are present in a world each turn all bots' **get_action** functions are called to request an action. The actions then occur simultaneously.

Multiple bots may be present on one location. If multiple bots on the one location make buy or sell requests which cannot all be satisfied the trade is divided fairly between the bots. Items will be left unsold if it not possible to divide trade fairly.

If, for example, there are 7 items for sale at a location, and 3 bots simultaneously request to buy all 7 items each bot will be sold 2 items and 1 item will be left unsold.

Getting Started

The week 10 lab exercises take you through getting started.
You do not provide a main function for this assignment.

You instead submit two functions with these prototypes:

```
char *get_bot_name(void);

void get_action(struct bot *b, int *action, int *n);
```

These function will be called when your bot needs tomake an action.
get_bot_name is called once at the start of every game to get your bot's name. This may be any string you choose. Do not use offensive or obscene words. A suffix will be added to your name if necessary to make it unique.

get_action is called once every turn to get your bot's move.

Your **get_action** function should assign values appropriate values to `*action` and to `*n` .

These two values whould indicate the action your bot wishes to make this turn.

Here is very simple bot which just moves forward 1 location every turn:

```
#include "trader_bot.h"

char *get_bot_name(void) {
    return "Snail Bot";
}

void get_action(struct bot *b, int *action, int *n) {
    *action = ACTION_MOVE;
    *n = 1;
}
```

The script `~cs1511/bin/bot_test` will automatically test your program on a random world.
It provides a main function which creates the structs describing the world and calls your **get_action** to get your bot's action every turn. For example:

```
$ ~cs1511/bin/bot_test mybot.c|more
dcc mybot.c /home/cs1511/public_html/assignments/trader_bot/trader_bot_r
*** Trader Bot Parameters ***
cash=127483
fuel_tank_capacity=45
maximum_cargo_volume=1808962
maximum_cargo_weight=226238
maximum_move=7
n_bots=1
turns_left=13

*** Trader Bot Commodities ***
Mobiles volume=230 weight=936

Turn 0
CSE: start
Shell: Petrol station 16 units of available fuel for $154
J&B Hifi: will buy 164 units of Mobiles for $443
Coogee Beech: other
Samsung: will sell 344 units of Mobiles for $274
Dump: dump
Coogee Beech: other

"Mr Robot" is at CSE with $127483 and 45 fuel
```

When **get_action** is call to get your action its given a pointer to a struct describing the current state of your bot.
Do not change this struct or the structs it points to.

A full description of the Trader Bot world can be obtained by following pointers starting with this struct.

A significant component of this assignment is understanding the representation of the Trader Bot world.

The constants and types used in this representation are described in this file: **trader_bot.h** (trader_bot.h)

You will need to #include **trader_bot.h** in your code.

To help you understand **trader_bot.h** look at this **draw.io diagram** (https://www.draw.io/?lightbox=1&highlight=0000ff&edit=_blank&layers=1&nav=1&title=Single%20Bot%20Trade%20Bot%20World#R7V1dc9u4kv01ftkqpQiAn4%2BTTG showing the structs and pointers representing a Trader Bot World.

The Trader Bot world is the same one shown in the above example of running **bot_test** but at turn 8 when the state of the world is this:

```
Turn 8

CSE: start
Shell: Petrol station 16 units of available fuel for $154
J&B Hifi: will buy 164 units of Mobiles for $443
Coogee Beech: other
Samsung: will sell 113 units of Mobiles for $274
Dump: dump
Coogee Beech: other

"Mr Robot" is at Shell with $61449 and 26 fuel carrying: 241 Mobiles
```

Note pointer **b** in the diagram represents the pointer that would be passed to **get_action** to get the bot's move for Turn 8.

Improving your Bot

If given the option **--valgrind** option bot_test will compile your bot with **dcc --valgrind**. This is highly recommended to check you don't have uninitialized variables.

```
$ ~cs1511/bin/bot_test --valgrind dodgy_bot.c
```

You can also check for memory leaks with the option **--leak-check**. This will compile your bot with **dcc --leak-check**.

```
$ ~cs1511/bin/bot_test --leak-check dodgy_bot.c
```

Once you have your bot basically working, there are 3 pre-supplied worlds: **medium_world.txt**, **large_world.txt**, **larger_world.txt**, you test it in a world with the **-w** option for example:

```
$ ~cs1511/bin/bot_test -w medium_world.txt mybot.c|less -R
```

It is possible to construct your own *Trader Bot* world. Use the the medium world specification (medium_world.txt) as starting point e.g:

```
$ cp /home/cs1511/public_html/assignments/trader_bot/medium_world.txt myworld.txt
$ gedit myworld.txt &
$ ~cs1511/bin/bot_test -w myworld.txt great_bot.c|less -R
```

You can use the **-n** option to run multiple copies of your bot in the same world.

```
$ ~cs1511/bin/bot_test -n 5 -w medium_world.txt great_bot.c|less -R
```

You can test different versions of your bot against each other by placing all the source files for that version in a separate directory. For example, if you have 3 versions of your bot in the directories *bot_v1*, *bot_v2* and *bot_v3*, you can test them against each other with this command:

```
$ ~cs1511/bin/bot_test bot_v1 bot_v2 bot_v3|less -R
```

You can also re-use a particular seed with the **-s** option.

```
$ ~cs1511/bin/bot_test -s 308 bot.c|less -R
```

At the request of students the *bot_test* uses ANSI sequences to displayed parts of its output in colour on terminals. You can disable this with the **--no_colorize** option.

Trader Bot Tournaments

Tournaments will be run on all bots submitted with *give*, starting Tue May 16 with the results displayed on the class web page. This will allow you to see how your bot performs in a world with many other bots present.

Submitted bots will be first tested in a single-bot-world. They will only be be added to the tournament if they reach a qualifying-level of performance in a single-bot world.

Assumptions/Restrictions/Clarifications

You should follow discussion about the assignment in the class forums. Questions about the assignment should be posted there so all students can see the answer.
You may submit multiple **.c** files containing your Trader Bot code.

You may use any name for the source files of your program except **trader_bot_main.c**.

Your source files must not include a **main** function.

Your source files must contain two functions with exactly these prototypes:

```
char *get_bot_name(void);

void get_action(struct bot *b, int *action, int *n);
```

Your C files **get_action** should not change the structs representing the Trader Bot world.

Your source files may contain other functions.

DO NOT CHANGE trader_bot.h.

It is not necessary to submit **trader_bot.h**. If you do submit **trader_bot.h** its contents will be replaced by this file.

You may submit other **.h** files.

You may submit data files with the suffix .txt to be used by your program. You submitted code must be C only. You may not submit code in other languages. You may not use *system* or other C functions to run external programs.

You may call functions from the standard C libraries (e.g. the functions from `stdio.h`, `stdlib.h`, `string.h`) and the maths library (`math.h`).

You may not use functions from other C libraries (libraries that require the -l flag for dcc). In other words as long as dcc compiles your program without the -l flag you are fine.

Your **get_action** function must take at most 10 seconds to return a move on a CSE computer when compiled with dcc --valgrind.

There is no way for your bot to pass information from one turn to the next. Your bot is not permitted to create files.

Sellers do not generate more of the items they sell during the game. The number of items they have for sale is reduced when bots buy items from them. The number of items they have for sale does not otherwise change.

This is also true for buyers. The number of items a buyer is willing to buy reduces when bots sell items to them. The number of items they are willing to buy does not otherwise change during the game.

trader_bot.h will not be changed when your bot is compiled for marking.

Submission of Work

You are required to submit intermediate versions of your assignment.
Every time you work on the assignment you should copy your work to your CSE account and submit it using the give command below.

It is fine if intermediate versions do not compile or otherwise fail submission tests.

Only the final submitted version of your assignment will be marked.

All these intermediate versions of your work will be placed in a git repo and made available to you via a web interface at this URL, replace *z5555555* with your zpass. `https://gitlab.cse.unsw.edu.au/z5555555/17s1-comp1511-ass2/commits/master`

This will allow you to retrieve earlier versions of your code if needed.

You submit your work like this:

```
$ give cs1511 ass2  C files
```

Blogging

You must blog every time you work on this assignment, recording how much time you spent working on the assignment and what this time was spent doing (reading, designing, coding, testing, debugging, ...).
You must blog about all significant bugs in your assignment including what test found the bug, how the bug was tracked down and fixed, how long this took and any lessons learnt.

You may create one big blog post and edit each time, or multiple small blog posts for the assignment.

Attribution of Work

This is an individual assignment. The work you submit must be your own work and only your work apart from these exceptions.
Joint work is not permitted.
You may use code written with your lab partner for the functions in the week 10 lab. This code should include comments indicating the joint-authorship with your lab partner.

You may use code from the sample solutions for the week 10 lab. You should add comments indicating the source of this code.

You may use small amounts (< 10 lines) of general purpose code (not specific to the assignment) obtained from site such as Stack Overflow or other publically available resources. You should attribute clearly the source of this code in a comment with it.

You are not permitted to request help with the assignment apart from in the course forum or from course staff.

Do not provide or show your assignment work to any other person (including by posting it on the forum) apart from the teaching staff of COMP1511.

The work you submit must otherwise be entirely your own work. Submission of work partially or completely derived from any other person or jointly written with any other person is not permitted. The penalties for such an offence may include negative marks, automatic failure of the course and possibly other academic discipline. Assignment submissions will be examined both automatically and manually for such submissions.

Relevant scholarship authorities will be informed if students holding scholarships are involved in an incident of plagiarism or other misconduct. If you knowingly provide or show your assignment work to another person for any reason, and work derived from it is submitted you may be penalized, even if the work was submitted without your knowledge or consent. This may apply even if your work is submitted by a third party unknown to you.

Note, you will not be penalized if your work is taken without your consent or knowledge.

Assessment

This assignment will contribute 15% to your final mark.

The assessment for the assignment recognizes the difficulty of the task.

5% of the marks for assignment 2 will come from your tutor reading your blog. They will assess generously any genuine attempts to record the information described above.

15% of the marks for assignment 2 will come from hand marking of the readability of the C you have written. These marks will be awarded on the basis of clarity, commenting, elegance and style. In other words, your tutor will assess how easy it is for a human to read and understand your program.

Here is an indicative marking scheme. It assumes you do a good job blogging.

85+%	An elegant readable program which competes well against other players.
75%	A well written program which makes good profits in a single player situation.
65%	A well written program which reliably makes a non-trivial profit in a single player situation.
55%	A working program which often makes a profit.
-70%	Knowingly providing your work to anyone and it is subsequently submitted (by anyone).
-70%	Submitting any other person's work. This includes joint work.
0 FL for COMP1511	Paying another person to complete work. Submitting another person's work without their consent.

The lecturer may vary the assessment scheme after inspecting the assignment submissions but it will remain broadly similar to the description above.

Due Date

This assignment is due at 23:59pm Sunday June 4.
If your assignment is submitted after this date, each hour it is late reduces the maximum mark it can achieve by 5%. For example if an assignment worth 74% was submitted 5 hours late, the late submission would have no effect. If the same assignment was submitted 15 hours late it would be awarded 25%, the maximum mark it can achieve at that time.