

## Assignment 2

### Due Date

*This assignment is to be completed by individuals.* The assignment is due at 3PM Wednesday May 3<sup>rd</sup> 2017.

### Background

Aircraft auto-pilots navigate using a *flight plan*. This is a collection of locations which must be flown over/to. These are called *waypoints*. For example flying from Hobart (international code “YMHB”) to Brisbane (“YBBN”) the plane flies over Merimbula (“YMER”) in New South Wales. In addition to a name, each waypoint also consists of a coordinate. Merimbula’s coordinate is 55H 758488 5911327. This has three parts: a grid reference (“55H”), an easting (758488), and a northing (5911327).

All the waypoints from the origin (for example Hobart) to the destination (for example Brisbane) would be collected together in an ordered list with a cursor which refers to the upcoming waypoint on the journey, the origin if the plane isn’t in flight yet, or the destination if the plane has already landed. There is no restriction on the number of included waypoints on a flight path and no need to do anything other than have the cursor advance/retreat from one waypoint to the next.

- a Which underlying data structure (*array* or *linked-list*) will you use as a basis to model the flight plan? In two–three sentences, justify your answer.
- b Which kind of abstract data type (*binary tree*, *general tree*, *array*, *stack*, *priority queue*, *double-ended queue*, *set*, *list*, etc.) would you use to model the flight plan? In two–three sentences, justify your answer by indicating the functionality required.

The types required to implement this include the following:

```
struct waypoint_int {
    char name[5];
    char grid[4];
    long int easting;
    long int northing;
};
typedef struct waypoint_int *waypoint;
```

You may assume the existence of the following functions which work on waypoints:

```
void init_waypoint(waypoint *wp, char *m, char *g, long
                  int e, int n);
```

```

char *get_name(waypoint w);
char *get_grid(waypoint w);
long int get_easting(waypoint w);
long int get_northing(waypoint w);
void set_name(waypoint w, char *m);
void set_grid(waypoint w, char *g);
void set_easting(waypoint w, long int e);
void set_northing(waypoint w, long int n);

```

You may further assume the existence of the following type declarations:

```

struct flight_path_int;
typedef struct flight_path_int *flight_path;

```

- c Define the `struct flight_path_int` type based upon your choice in (a) and define all other required types.
- d Define a function to create the flight path stub which consists only of the origin (which is the current location) and destination. The function should have the following function header:

```

void init_flight_path(flight_path *fp,
                     waypoint origin, waypoint destination);

```

- e Define a function to add a new waypoint to a flight path by inserting it into the list after the current location. If the given flight path is empty, an error message should be displayed and no addition should occur. Your definition should conform to the following function header:

```

void add_next(flight_path f, waypoint interim);

```

- f Define a function to search the whole list of waypoints for a particular name, printing and returning the UTM of the located waypoint as a string. Using the example from above, searching for "YMER" should show the grid, easting, and northing values and yield: "55H 5911327 758488". If the desired waypoint is not present in the list you should not print anything and return "". Your definition should conform to the following function header:

```

char *locate(flight_path f, char *name);

```

- g Write the function `heading()` which returns the name of the next waypoint in the list. `heading()` takes a Boolean variable which indicates whether the plane is required to go into a holding pattern over the current location. If this is the case — or if the plane is at the destination — then `heading()` should yield the name of the current location. An error message should be displayed if the flight path is empty and "" should be returned. `heading()` should return

the name of the upcoming/current waypoint and has the following function header:

```
char *heading(flight_path f, bool holding);
```

- h Write the function `remaining()` that traverses the list counting the number of waypoints from the named waypoint until an end of the journey (or `-1` if the waypoint name cannot be found in relevant portion of the list). The count should be inclusive of the waypoint. If the third parameter is `true` the count is forwards from the current location to the destination, if `false`, the count is from the current location backwards to the origin. `remaining()` has the following function header:

```
int remaining(flight_path f, char *name, bool
              onwards);
```

- i Write the function `skip()` which deletes the next waypoint in the flight plan. If the flight plan is empty or if the cursor is at the destination an error message should be displayed and no change made. `skip()` should have the following function header:

```
void skip(flight_path f);
```

- j Sometimes in flight an issue arises and the plane needs to return to where it has come from. For this to occur the flight path must be reversed. Write the `reverse()` function to invert an entire flight path. `reverse()` has the following function header:

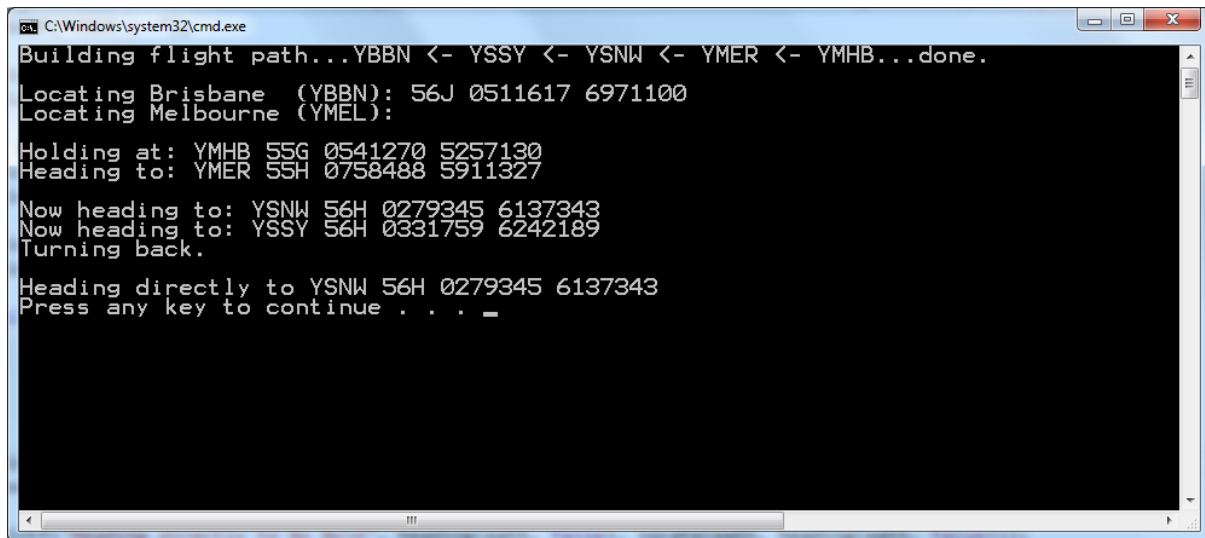
```
void reverse(flight_path f);
```

## Program specification

You must write one or more C programs to define the types and implement the functions discussed above. You should download a Visual Studio project as a starting point which comprises the following files:

- `flight_path.h` and `flight_path.c` — the *Flight Path* ADT as specified above. *You must complete `flight_path.c`*;
- `waypoint.h` and `waypoint.c` — the *Waypoint* ADT as specified above;
- `assig_two117.c` — the file which contains the `main()` function and other functions which implement the required task.

You should add program files for all other types that you need based upon your choice in (a) above. Your program is probably correct if you see the following output:



```
C:\Windows\system32\cmd.exe
Building flight path...YBBN <- YSSY <- YSNW <- YMER <- YMHB...done.
Locating Brisbane (YBBN): 56J 0511617 6971100
Locating Melbourne (YMEL):
Holding at: YMHB 55G 0541270 5257130
Heading to: YMER 55H 0758488 5911327
Now heading to: YSNW 56H 0279345 6137343
Now heading to: YSSY 56H 0331759 6242189
Turning back.
Heading directly to YSNW 56H 0279345 6137343
Press any key to continue . . . _
```

## Program Style

Your program should follow the following coding conventions:

- `const` variable identifiers should be used as much as possible, should be written all in upper case and should be declared before all other variables
- `#defined` symbols should be used for constant values if `const` is inappropriate
- variable identifiers should start with a lower case letter
- every `if` and `if-else` statement should have a block of code (i.e. collections of lines surrounded by `{` and `}`) for both the `if` part and the `else` part (if used)
- every `do`, `for`, and `while` loop should have a block of code (i.e. `{}`s)
- the keyword `continue` should not be used
- the keyword `break` should only be used as part of a `switch` statement
- opening and closing braces of a block should be aligned
- all code within a block should be aligned and indented 1 tab stop (or 4 spaces) from the braces marking this block
- global variables should be used sparingly with parameter lists used to pass information in and out of functions.
- commenting:
  - There should be a block of header comment which includes at least
    - file name
    - student name
    - student identity number
    - a statement of the purpose of the program
    - date
  - Each variable declaration should be commented
  - There should be comments that identify groups of statements that do various parts of the task
  - Comments should describe the strategy of the code and should not simply translate the C into English

## What and how to submit

### What to submit

#### *Paper submission*

- A paper cover page (blanks can be collected from the Information and Communications Technology Discipline Help Desk).
- Written answers to parts (a) and (b) above.
- A *landscape oriented* print-out of `flight_path.c` and any other program files you have added to the solution. *Your assignment will not be fully marked unless these are present.*

#### *Electronic submission*

- You should submit the entire Visual Studio project folder compressed as a ZIP file.

### How to submit

#### *Paper submission*

- Firmly staple together all of the required documents (with the signed cover page on top) and place them in the appropriate submissions box near the ICT Help Desk.

#### *Electronic submission*

- You should ZIP the Visual Studio project folder and then submit it to the “Assignment 2” assignment drop-box on *KIT107*’s MyLO site.

## Marking scheme

Task/Topic	Maximum mark
<i>Design</i>	
ADT chosen wisely	2
Data structure correct and justified	2
<i>Program operates as specified</i>	
<code>struct flight_path_int</code> correctly specified	2
<code>init_flight_path()</code> correctly completed	2
<code>add_next()</code> correctly completed	2
<code>locate()</code> correctly completed	2
<code>heading()</code> correctly completed	2
<code>remaining()</code> correctly completed	4
<code>skip()</code> correctly completed	3
<code>reverse()</code> correctly completed	3
<i>Program Style</i>	
Does not unnecessarily repeat tests or have other redundant/confusing code	3
Uses correctly the C naming conventions	3
Alignment of code and use of white space makes code readable	3
Always uses blocks in branch and loop constructs	3
Meaningful identifiers	3
Header comments for the program (author, purpose, date, filename etc)	3
Each variable declaration is commented	3
Comments within the code indicate the purpose of sections of code (but DO NOT just duplicate what the code says)	3

## **Plagiarism and Cheating:**

Practical assignments are used by the Discipline of Information and Communication Technology for students to both reinforce and demonstrate their understanding of material which has been presented in class. They have a role both for assessment and for learning. It is a requirement that work you hand in for assessment is substantially your own.

### **Working with others**

One effective way to grasp principles and concepts is to discuss the issues with your peers and/or friends. You are encouraged to do this. We also encourage you to discuss aspects of practical assignments with others. However, once you have clarified the principles, you must express them in writing or electronically entirely by yourself. In other words you must develop the algorithm to solve the problem and write the program which implements this algorithm yourself. You can discuss the question, but not the solution.

### **Cheating**

- Cheating occurs if you claim work as your own when it is substantially the work of someone else.
- Cheating is an offence under the [Ordinance of Student Discipline](#) within the University. Furthermore, the ICT profession has ethical standards in which cheating has no place.
- Cheating involves two or more parties.
  - If you allow written work, program print-outs, or electronic versions of your code to be borrowed or copied by another student you are an equal partner in the act of cheating.
  - You should be careful to ensure that your work is not left in a situation where it may be stolen by others.
- Where there is a reasonable cause to believe that a case of cheating has occurred, this will be brought to the attention of the unit lecturer. If the lecturer considers that there is evidence of cheating, then no marks will be given to any of the students involved. The case will be referred to the Head of School for consideration of further action.

Julian Dermoudy, April 9<sup>th</sup> 2017.