Started on	Friday, 16 August 2019, 2:55 PM
State	Finished
Completed on	Friday, 30 August 2019, 12:00 AM
Time taken	13 days 9 hours
Grade	**10.00** out of 10.00 (**100**%)

Information

About this quiz

This is the second part of the C-programming assignment superquiz also known as ENCE260 Assignment 1. The due date and time is nominally 11:55pm, Friday 23rd August but submissions will be accepted without penalty until 11:55pm on Thursday 29th August.

The quiz contributes 6% to your total ENCE260 course grade and is in two parts. The first part relates to lab 5, on structures. It's an excellent exercise for helping you to come to grips with strings, pointers and structures, so you'll be ready for the mid-semester test. There are two questions in this part, worth 15% and 35% of the quiz respectively.

The second part involves processing a CSV file of temperature data to extract some statistics and key data. It too is broken into two questions, worth 10% and 40% of the marks for the quiz, respectively.

All questions can be done with just the knowledge from labs 1 through 5.

Style penalties of up to 20% will be applied for submissions that do not conform to the ENCE260 style guidelines. Style will be assessed manually, after the final close date.

As usual, duplicate or near-duplicate assignments will receive zero marks. Attempts to beat the system with code that isn't in the spirit of the question (e.g. printing known answers directly) will also receive zero marks.

Information

ProcessStudents, take #2

In this second part of the superquiz, you are provided with a version of *structexample3.c* called *prog.c* that has a modified *student* struct. In the first two questions you are to make the following changes to that program.

1. You must insert the missing body of the *readOneStudent* function so that the program behaves as intended, reading the students from the specified input file and printing a list of them to standard out.
2. You must then modify the program so that it maintains the list of students in alphabetical order of name. The program is also required to take the name of the file to be processed as a command line argument, rather than being hard-coded into the source file.

The requirements are documented in more detail in the two questions concerned.

Note on the Standard Error Stream

So far all our program output has been via functions like *printf*, *puts* or *putchar*, all of which write to what is called the *standard output* stream, often referred to as "stdout". By default standard output goes to the terminal but can be redirected by means of the bash '>' symbol to a file. The standard output stream can be explicitly written to using the file variable *stdout* (declared in <stdio.h>) and a set of functions like *fprintf*, *fputs, fputc* etc which take the file variable (often called a "stream") as an explicit parameter. For example

```
fprintf(stdout, "Hello %s\n", name);
```

is **exactly** equivalent to

```
printf("Hello %s\n", name);
```

Linux programs can also send output to an alternative built-in output stream called the *standard error* stream. The built-in variable *stderr* refers to this stream, which is conventionally used only for error messages. [Aside: the odd program like *valgrind* uses it for its informational output so that this can be kept separate from the output of the program being debugged.]

For this superquiz, all you need to know is that you can print output to the standard error stream using lines like

```
fprintf(stderr, "Bad name: %s\n", name);
```

The program *prog.c* already contains an example of such a line.

Question **1**

Correct

Mark 1.50 out of 1.50

readOneStudent

The supplied file *prog.c* is similar to structexample3.c except that:

1. The student struct now contains a first name, a last name and a student ID instead of a name and an age. StudentID is an int.
2. The input data file is again a CSV file but the fields are firstname, lastname and studentId (an int). Here is an example file.
3. The body of the *readOneStudent* function, which needs to be different from that in *structexample3.c* in order to handle the different *Student* struct, appears to have gone missing.

The (mostly empty) *readOneStudent* function has been preloaded into the answer box. You task in this question is to re-constitute the missing body. The answer box should contain only the function *readOneStudent,* which will be plugged back into *prog.c* (with minor variations to the main function) for testing. Thus, you can and should use the *newStudent* function in your implementation of *readOneStudent*.

For example:

Test	Result

`FILE* inputFile = fopen("studlist.asst.txt", "r");`
`Student* studentList = readStudents(inputFile);`
`printStudents(studentList);`
	`Fred Nurk (12345678)`
	`Richard Lobb (9876543)`
	`Richard Lobby (981349)`
	`JingWu Wang (2913244)`
	`Tweedle Dum (16813486)`
	`Tweedle Dee (98234532)`
	`Agnes McGurkinshaw (987612)`

Answer: (penalty regime: 0, 5, 10, 20, ... %)

Reset answer

Ace editor not ready. Perhaps reload page?

Falling back to raw text area.

```
// Read a single student from a csv input file with student first name in first column,
// second name in the second column and studentId in the last (third) column.
// Returns: A pointer to a Student record, or NULL if EOF occurs or if
// a line with fewer than 2 commas is read.
Student* readOneStudent(FILE* file)
{
    //i take in the POINTER to the first line of output from the csv file which is
already a student struct?
    Student* student = NULL;       // Pointer to a student record from the pool
    char buffer[MAX_LINE_LENGTH];
    char* commaPos1 = NULL;
    char* commaPos2 = NULL;
    char* line = fgets(buffer, MAX_LINE_LENGTH, file);
    if ( line == NULL) {
        return NULL;
    }
    commaPos1 = strchr(buffer, ',');
    if (commaPos1 != NULL) {
        commaPos2 = strchr(commaPos1 + 1,',');
    }

    if ((commaPos1 == NULL) || (commaPos2 == NULL)) {
        return NULL;
    }
    int identification = atoi(commaPos2 + 1);
    *commaPos1 = '\0';
    *commaPos2 = '\0';

    student= newStudent(buffer, commaPos1 + 1, identification);
    return student;
}
```

	Test	Expected	Got	

✔	`FILE* inputFile = fopen("studlist.asst.txt", "r"); Student* studentList = readStudents(inputFile); printStudents(studentList);`	`Fred Nurk (12345678)`
		`Richard Lobb (9876543)`
		`Richard Lobby (981349)`
		`JingWu Wang (2913244)`
		`Tweedle Dum (16813486)`
		`Tweedle Dee (98234532)`
		`Agnes McGurkinshaw (987612)`
			`Fred Nurk (12345678)`
			`Richard Lobb (9876543)`
			`Richard Lobby (981349)`
			`JingWu Wang (2913244)`
			`Tweedle Dum (16813486)`
			`Tweedle Dee (98234532)`
			`Agnes McGurkinshaw (987612)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.50/1.50.

Question **2**

Correct

Mark 3.50 out of 3.50

Modifying the main program

After the last question you should have finished up with a working *prog.c*. In this question you are to modify that program as follows

1. The program is to be run with a bash command of the form:

   ```
   ./prog filename
   ```

   where *filename* is the name of the file to be processed. For example:

   ```
   ./prog studlist2.txt
   ```

2. As they are read from the file, students are inserted into the list in alphabetical order of their first name, rather than at the end as in the example or the beginning as in the lab quiz. If two or more students have the same first name, they must be inserted in increasing order of their second name. It is guaranteed that in the test data no two students have the same first and last names.

If the command is not run with a single filename argument you should print to the *stderr* stream the output

```
Usage: prog filename
```

and the program should immediately exit. The introduction explains what the *stderr* stream is.

If the given filename cannot be opened, you should print to *stderr* an error message of the form

```
File 'blah.txt' not found
```

where the actual file name given in the command should appear between the apostophes. Again the program should then immediately exit. Please realise that code in the existing *prog.c* needs to be modified to print an error message including the actual filename from the command line, not the fixed filename *studlist.asst.txt*.

Alphabetical sort ordering is defined by the ordinal values of the characters, i.e. their positions in the ASCII table. So 'A' < 'B' < ... < 'Z' .... < 'a' < 'b' < ... 'z'. This is the ordering that *strcmp* implements. If the first characters compare equal, the ordering is defined by the second characters and so on until a difference is found.

Non-functional Requirements

The program must be a straightfoward variation on that provided in *prog.c*. Specifically, each student must be represented by a struct and students must be inserted into a singly-linked list, maintained in alphabetical order at all times.

Hints

1. To get the filename from the command line, see the section *Command-line arguments* in part 5 of the lecture notes and video Part 5:3 Arrays of strings .
2. To help with constructing a sorted list, you are strongly advised to write two support functions as follows:
   - *bool precedes(const Student* student1, const Student* student2)* returns true if and only if student *student1* should precede (i.e., is "less than") student *student2* in the list. This is a pretty trivial 4 line function, but without it, the next function is much harder.
   - *Student* insert(Student* student, Student* list*) inserts the student *student* into the list of students that has *list* as its first element. The return value is the new list head; it will be either *student* if the new student went at the head of the list or *list* otherwise. My version of this is under 20 lines long, and of course uses *precedes.*
3. *valgrind* is used for many of the tests. You should check your program with valgrind before submitting in case it has non-obvious errors, such as conditions that depend on uninitialised variables.

For example:

Test	Result

`./prog studlist.asst.txt`	`Agnes McGurkinshaw (987612)`
	`Fred Nurk (12345678)`
	`JingWu Wang (2913244)`
	`Richard Lobb (9876543)`
	`Richard Lobby (981349)`
	`Tweedle Dee (98234532)`
	`Tweedle Dum (16813486)`

Answer: (penalty regime: 0, 5, 10, 20, ... %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
/* prog.c
 * A variant of structexample3.c from lab 5 with the following changes:
 *
 * 1. The student struct now contains a firstname, a lastname and a student ID
 *    instead of a name and an age.
 * 2. The input data file is again a CSV file but the fields are firstname
```

```
 *  2. The input data file is again a CSV file but the fields are firstname,
 *     lastname and student ID (an int).
 *
 * Richard Lobb, August 2019.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

#define MAX_LINE_LENGTH 80      // The longest line this program will accept
#define MAX_NUM_STUDENTS 500    // The maximum number of students this program can
handle
#define MAX_NAME_SIZE 50        // The maximum allowable name length

// The declaration of the student record (or struct). Note that
// the struct contains the first and last names as arrays of characters.

typedef struct student_s Student;

struct student_s {
    char firstname[MAX_NAME_SIZE];
    char lastname[MAX_NAME_SIZE];
    int studentId;
    Student* next;              // Pointer to next student in a list
};

// Create a pool of student records to be allocated on demand

Student studentPool[MAX_NUM_STUDENTS];  // The student pool
int firstFree = 0;

// Return a pointer to a new student record from the pool, after
// filling in the provided first and last name and student ID fields.
// Returns NULL if the student pool is exhausted.
Student* newStudent(const char* firstname, const char* lastname, int studentId)
{
    Student* student = NULL;
    if (firstFree < MAX_NUM_STUDENTS) {
        student = &studentPool[firstFree];
        firstFree += 1;
        strncpy(student->firstname, firstname, MAX_NAME_SIZE);
        student->firstname[MAX_NAME_SIZE - 1] = '\0';
        strncpy(student->lastname, lastname, MAX_NAME_SIZE);
        student->lastname[MAX_NAME_SIZE - 1] = '\0';
        student->studentId = studentId;
        student->next = NULL;
    }
    return student;
}



bool precedes(const Student* student1, const Student* student2)
//return 1 if it student1 is smaller than student 2 return 0 if student2 is bigger
{
    //compare first name return 0 if student 1 is smaller
    // if first name is the same, compare second name and return 1 if student 1 is
smaller and 0 if student 2 is smaller
    int result = strcmp(student1->firstname, student2->firstname);
    if (result == 0) {
        result = strcmp(student1->lastname, student2->lastname);
        if (result > 0) {
            return 1;
        } else {
            return 0;
        }
```

```
    } else {
        if (result > 0) {
            return 1;
        } else {
            return 0;
        }
    }
}


Student* insert(Student* student, Student* first)
//Propagating the linked list using precedes function.
//five main cases to be covered
//list only 1 item OK
//current->next = NULL OK
//traversing
//item becomes new first  OK
//insert item beween prev and current OK
{
    Student* Head = first;
    Student* Current = first;
    Student* Previous = NULL;
    while (precedes(student, Current) == 1)
        //while student is greater than current
    {
        if (Current->next == NULL) {
            //this means i am the largest
            Current->next = student;
            return Head;
        }
        Previous = Current;
        Current = Current->next;
        //Traversing
    }
    //outside while loop means that student is smaller than current
    if (Previous == NULL) {
        //it means that the list has only 1 item and i am the smalleset
        student->next = Current;
        Head = student;
    } else {
        //i am inserting item between prev and current
        Previous->next = student;
        student->next = Current;
    }

    return Head;
}


// Read a single student from a csv input file with student first name in first column,
// second name in the second column and studentId in the last (third) column.
// Returns: A pointer to a Student record, or NULL if EOF occurs or if
// a line with fewer than 2 commas is read.
Student* readOneStudent(FILE* file)
{
    //i take in the POINTER to the first line of output from the csv file which is
already a student struct?
    Student* student = NULL;
    char buffer[MAX_LINE_LENGTH];
    char* commaPos1 = NULL;
    char* commaPos2 = NULL;
    // if line is empty
    char* line = fgets(buffer, MAX_LINE_LENGTH, file);
    if ( line == NULL) {
        return NULL;
    }
    commaPos1 = strchr(buffer, ',');
    if (commaPos1 != NULL) {
        commaPos2 = strchr(commaPos1 + 1,',');
```

```
    }

    if ((commaPos1 == NULL) || (commaPos2 == NULL)) {
        return NULL;
    }
    int identification = atoi(commaPos2 + 1);
    *commaPos1 = '\0';
    *commaPos2 = '\0';
    student= newStudent(buffer, commaPos1 + 1, identification);
    return student;
}

// Reads a list of students from a given file. Input stops when
// a blank line is read, or an EOF occurs, or an illegal input
// line is encountered.
// Returns a pointer to the first student in the list or NULL if no
// valid student records could be read.
Student* readStudents(FILE *file)
{
    Student* first = NULL;     // Pointer to the first student in the list
    Student* last = NULL;      // Pointer to the last student in the list
    Student* student = readOneStudent(file);
    while (student != NULL) {
        if (first == NULL) {
            first = last = student;   // Empty list case
        } else {
            //initiallise new list to store sorted students
            first = insert(student, first);
        }
        student = readOneStudent(file);
    }

    return first;
}

// printOneStudent: prints a single student, passed by value
void printOneStudent(Student student)
{
    printf("%s %s (%d)\n", student.firstname, student.lastname, student.studentId);
}


// printStudents: print all students in a list of students, passed
// by reference
void printStudents(const Student* student)
{
    while (student != NULL) {
        printOneStudent(*student);
        student = student->next;
    }
}

// Main program. Read a linked list of students from a csv file, then display
// the contents of that list.
int main(int argc, char* argv[])
{
    FILE* inputFile = fopen(argv[1], "r");
    if(argc != 2) {
        fprintf(stderr, "Usage: prog filename\n");
        return EXIT_FAILURE;
    } else if (inputFile == NULL) {
        fprintf(stderr, "File '%s' not found\n", argv[1]);
        return EXIT_FAILURE;
    }

    else {
        Student* studentList = readStudents(inputFile);
        printStudents(studentList);
```

```
        // The program could now do various things that make use of
        // the linked list, like deleting students and adding new ones,
        // but the program is already quite long enough!
    }
}
```

	Test	Expected	Got	

✔	`./prog studlist.asst.txt`	`Agnes McGurkinshaw (987612)`
		`Fred Nurk (12345678)`
		`JingWu Wang (2913244)`
		`Richard Lobb (9876543)`
		`Richard Lobby (981349)`
		`Tweedle Dee (98234532)`
		`Tweedle Dum (16813486)`
			`Agnes McGurkinshaw (987612)`
			`Fred Nurk (12345678)`
			`JingWu Wang (2913244)`
			`Richard Lobb (9876543)`
			`Richard Lobby (981349)`
			`Tweedle Dee (98234532)`
			`Tweedle Dum (16813486)`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 3.50/3.50.

Information

Introduction to Superquiz 2, Part 2

Congratulations on landing a job with ThermoDudes Ltd, a company that markets heat pumps around the South Island. They want statistics on temperature variations in different areas of the South Island to help them with their marketing. They can get temperature and other climate data from NIWA, via the web interface at https://cliflo.niwa.co.nz/ and now need a programmer to help them process the data. They are delighted to discover that you're a C programmer, so they pass the job to you on your first day in the office.

Question **3**

Correct

Mark 1.00 out of 1.00

strchrn - a warm-up exercise

To extract a particular column of interest you need to be able to locate its the start and end. So your task in this question is to write a function to find a single delimiter:

```
char* strchrn(char* s, int c, int n)
```

This is an extension of the library *strchr* and *strrchr* functions. While *strchr* finds the first occurrence of a given character in a string and strrchr finds the last, your new function *strchrn* finds the *n*th occurrence, where n = 1 is the first occurrence. The function returns a pointer to that character in *s*, or NULL if n <= 0 or no such occurrence of the character exists.

Your answer should not use any of the standard C string functions like *strchr* etc (not that they would help much, anyway).

For example:

Test	Result

`char* line = "This is a string";`
`for (int n = 1; n <= 5; n++) {`
`  char* ptr = strchrn(line, 'i', n);`
`  if (ptr != NULL) {`
`  int index = ptr - line;`
`  printf("Occurrence %d of 'i' found at index = %d\n", n, index);`
`  } else {`
`  printf("Occurrence %d of 'i' not found\n", n);`
`  }`
`}`
	`Occurrence 1 of 'i' found at index = 2`
	`Occurrence 2 of 'i' found at index = 5`
	`Occurrence 3 of 'i' found at index = 13`
	`Occurrence 4 of 'i' not found`
	`Occurrence 5 of 'i' not found`

Answer: (penalty regime: 0, 10, ... %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>

char* strchrn(char* s, int c, int n)
{
    //takes in char* pointer , letter to search for, c  and  which instance of c to return
    int occurrence = 0;
    if (n <= 0) {
        return NULL;
    }
    while (occurrence != n) {
        //edge cases: i reach end of list not enough occurrences
        if (*s == '\0') {
            return NULL;
        } else {
            //if letter im searching for matches letter in pointer
            if (*s == c) {
                occurrence++;
            }
        }
        s++;
    }
    //return the pointer to the nth iteration of c
    return s - 1;


}
```

	Test	Expected	Got	

	Test	Expected	Got	

✔	`char* line = "This is a string";`
	`for (int n = 1; n <= 5; n++) {`
	`  char* ptr = strchrn(line, 'i', n);`
	`  if (ptr != NULL) {`
	`  int index = ptr - line;`
	`  printf("Occurrence %d of 'i'`
	`found at index = %d\n", n, index);`
	`  } else {`
	`  printf("Occurrence %d of 'i' not`
	`found\n", n);`
	`  }`
	`}`
		`Occurrence 1 of 'i'`
		`found at index = 2`
		`Occurrence 2 of 'i'`
		`found at index = 5`
		`Occurrence 3 of 'i'`
		`found at index = 13`
		`Occurrence 4 of 'i'`
		`not found`
		`Occurrence 5 of 'i'`
		`not found`
			`Occurrence 1 of 'i'`
			`found at index = 2`
			`Occurrence 2 of 'i'`
			`found at index = 5`
			`Occurrence 3 of 'i'`
			`found at index = 13`
			`Occurrence 4 of 'i'`
			`not found`
			`Occurrence 5 of 'i'`
			`not found`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 1.00/1.00.

Question **4**

Correct

Mark 4.00 out of 4.00

processtemps.c

In this question you get to write the temperature-file processing program, *processtemps.c,* which processes a file of temperature data. The function *strchrn* should be useful.

The input data file

The supplied file dailyminmaxtempschch2017.txt is a typical temperature data file. If you open it in a text editor you'll see that it contains three distinct parts, separated by a single blank line. Firstly there's a section of station data, then a section of daily temperature recordings for every day of the year 2017. Lastly there's a brief epilogue of download information.

The second line of the temperature recordings section is a header line that describes what each column contains. The only columns of relevance to us here are Date (NZST) and Tmax(C). You may assume that they are always in columns 2 and 3 respectively and you may ignore the first two lines of that section. Tmax is the maximum temperature recorded throughout the 24 hour period.

You can open the sample temperature data file with Geany or any halfway decent text editor (i.e., anything other than Notepad).

Your task

Your are to write a program called *processtemps.c* that reads such a file and prints out a header containing the site name, a blank line and another header line for the daily recordings. This is followed by a list of all dates and maximum daily temperatures that equalled or exceeded a given threshold.

The program is run from the command line and must take two arguments: the filename and the threshold (a floating point number). For example:

```
processtemps dailyminmaxtempschch2017.txt 30.0
```

The example table below shows the expected output from that command.

The main header and table header lines should be printed regardless of whether or not there are any recordings exceeding (or equalling) the given threshold temperature.

If the command line does not have exactly the two expected arguments (filename and threshold) the program should print the following message to *stderr*.

```
Usage: processtemps filename threshold
```

If the specified file cannot be opened, the function should instead print to the *stderr* stream an error message exactly in the following format (where the file specified in the error message should of course be the same as the one passed in as a parameter). Note the quotes around the filename.

```
File 'dailyminmaxtempschch2017.txtt' not found
```

Notes:

1. You need to extract the site name from the header section. It will not necessarily be *Christchurch Aero.*
2. You may assume that commas are used to separate fields throughout.
3. You may assume that the date field always consists of a 4-character year, a 2-character month and a 2-character day followed by a timestamp (':0800') that can be ignored. Note however that the output date is printed differently, in the form *day/month/year* with 2-digit day and month values.
4. You may assume that the maximum length of an input line is 500 characters and the maximum length station name is 30 characters.
5. The output must be in exactly the format shown in the sample output. The date-and-temperature lines should be printed with a format of
   ```
   %02d/%02d/%d%9.1lf C\n
   ```
6. You may assume that line 3 of the file contains the site name in the first field.
7. valgrind is used in most of the tests except the first one, so you would be well advised to test your own code with valgrind.
8. Functions are at most 30 lines in length, excluding blank lines and comments.
9. You'll need to use *atof* to convert the string representation of temperatures in the input file and in the command line to numeric values. You may assume that only valid floating point strings will be encountered when testing.
10. Note that *atof* takes a pointer to a string representation of a floating point number, possibly with leading spaces, and converts characters until an illegal character is encountered. At that point the function returns the value obtained so far, with no error. Thus a call like *atof(" 23.567,1.234")* is perfectly valid and returns the first number, 23.567 (as a double).

Tests that you pass, except the first example one (like the above), are hidden. Different files will be used for testing, but the data in the test files will be in the same format - only the field values (i.e., the text and/or numbers between the commas) and the number of lines in the recordings sections of the file will change.

Remember that this assignment is being human-marked for style, so please put some effort into choosing nice identifiers, good structure and good comments.

For example:

Test	Result

`./processtemps dailyminmaxtempschch2017.txt 30.0`	`Dates when 30.0 C was reached at Christchurch Aero`
	
	`   Date       MaxTemp`
	`12/01/2017     30.2 C`
	`01/02/2017     31.0 C`
	`07/02/2017     30.1 C`
	`03/12/2017     30.4 C`
	`06/12/2017     30.7 C`
	`09/12/2017     32.5 C`
	`10/12/2017     31.0 C`
	`11/12/2017     30.5 C`
	`26/12/2017     30.9 C`

Answer: (penalty regime: 0, 10, 20, ... %)

Ace editor not ready. Perhaps reload page?
Falling back to raw text area.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>



#define MAX_INPUT_LINE_LENGTH 500
#define MAX_LENGTH_STATION_NAME 30

typedef struct daily_weather_s WeatherReport;

struct daily_weather_s {
    //A weather struct that holds characters of Day, Month, Year, a double Temperature
and a pointer to another WeatherReported if provided
    //Allocated 1 more memory space so as to hold the string terminatior condition
    char Year[5];
    char Month[3];
    char Day[3];

    double Temperature;
    WeatherReport* next;
};

WeatherReport ReportSpace[MAX_INPUT_LINE_LENGTH];
int firstFree = 0;



char* strchrn(char* s, int c, int n)
{
    //takes in char* pointer , letter to search for, c  and  which instance of c to
return, n
    int occurrence = 0;
    if (n <= 0) {
        return NULL;
    }
    while (occurrence != n) {
        //edge cases: i reach end of list not enough occurrences
        if (*s == '\0') {
            return NULL;
        } else {
            //if letter im searching for matches letter in pointer
            if (*s == c) {
                occurrence++;
            }
        }
        s++;
    }
    //return the pointer to the nth iteration of c
    return s - 1;
```

```
}

WeatherReport* GenerateWeatherReport(const char* Year, const char* Month, const char*
Day, const double Temp)
{
    //Makes a WeatherReport struct from the data given and returns it.
    WeatherReport* Report = NULL;
    Report = &ReportSpace[firstFree];
    firstFree += 1;
    strncpy(Report->Year, Year, 20);
    Report->Year[4] = '\0';
    strncpy(Report->Month, Month, 20);
    Report->Month[2] = '\0';
    strncpy(Report->Day, Day, 20);
    Report->Day[2] = '\0';
    Report->Temperature = Temp;
    Report->next = NULL;
    return Report;
}

WeatherReport* readOneValue(FILE* file)
//processing file here
//read one line of file
//extract date, month, year and Temperature
//sends all the generated data to GenerateWeatherReport function to make the
weatherstruct
//returns weather struct with date month year and Temp already initialized
{
    WeatherReport* OneWeatherInstance = 0;
    // if line is empty
    //return
    char buffer[MAX_INPUT_LINE_LENGTH];

    char* line;
    line = fgets(buffer, MAX_INPUT_LINE_LENGTH, file);
    if (line == NULL || *line == '\n') {

        return NULL;
    }
    char* startofdate = strchrn(line, ',', 1) + 1;
    char* startoftemp = strchrn(line, ',', 2) + 1;
    char Year[5] = {0};
    memcpy(Year, startofdate, 4);
    startofdate += 4;
    char Month[3] = {0};
    memcpy(Month, startofdate, 2);
    startofdate += 2;
    char Day[3] = {0};
    memcpy(Day, startofdate, 2);
    startofdate += 2;
    char unconvertedTemp[5];
    memcpy(unconvertedTemp, startoftemp, 4);
    unconvertedTemp[4] = '\0';
    double Temp = atof(unconvertedTemp);
    OneWeatherInstance = GenerateWeatherReport(Year, Month, Day, Temp);
    return OneWeatherInstance;
}


WeatherReport* readWeather(FILE *file, double threshhold)
{
    //i generate a linked list here which will return a pointer to first item in my
list
    //this will then be printed using the printreport function
    WeatherReport* first = NULL;
    WeatherReport* Head = NULL;
    WeatherReport* weatherInstance = readOneValue(file);
    while (weatherInstance != NULL) {
        if ((weatherInstance->Temperature) < threshhold) {
```

```
        } else {
            if (first == NULL) {
                first = Head = weatherInstance;
                first->next = Head;
                //This line is to cover the edge case where there is only one item in
the list.
                Head->next = NULL;
            } else {
                Head->next = weatherInstance;
                Head = Head->next;
            }
        }
        weatherInstance = readOneValue(file);
    }
    return first;
}

char* get_line(char* outputPointer, int linetoread, FILE*inputFile)
{
    //a function that just uses fgets to get to the linetoread specified from the file
and returns
    //a pointer to it.

    for (int counter = 0; counter != (linetoread - 1); counter++) {
        char useless[MAX_INPUT_LINE_LENGTH];
        fgets(useless, MAX_INPUT_LINE_LENGTH, inputFile);


    }
    fgets(outputPointer, MAX_INPUT_LINE_LENGTH, inputFile);
    return outputPointer;
}

void printReport(WeatherReport* weatherSummary, char* sitename, double maxTemp)
//takes in the all the required inputs weatherSummary, name of the place ,
maxtemperature provided to format it nicely into the output required.
{
    int i = 0;
    printf("Dates when %0.1f C was reached at ", maxTemp);
    //The printing of the name was hardcoded as i didnt remember that i could have just
inserted an EOF to the replace the ',' sorry
    for (; sitename[i] != ','; i++) {
        printf("%c", sitename[i]);
    }
    puts("\n");
    puts("   Date       MaxTemp");
    while (weatherSummary != NULL) {
        printf("%s/%s/%s     %0.1f C\n", weatherSummary->Day,  weatherSummary->Month,
weatherSummary->Year,  weatherSummary->Temperature);
        weatherSummary = weatherSummary->next;
    }




}

int main(int argc, char* argv[])
{
    //Takes in 3 arguments and generates a weather report based on the file
provided(arg 2) and the maximum temperature(arg 3)
    FILE* inputFile = fopen(argv[1], "r");
    if(argc != 3) {
        fprintf(stderr, "Usage: processtemps filename threshold\n");
        return EXIT_FAILURE;
    } else if (inputFile == NULL) {
        fprintf(stderr, "File '%s' not found\n", argv[1]);
        return EXIT_FAILURE;
```

```
} else {
    double threshhold = atof(argv[2]);
    char sitename[MAX_LENGTH_STATION_NAME] = {0};
    get_line(sitename, 3,inputFile);
    char datastart[MAX_INPUT_LINE_LENGTH] = {0};
    get_line(datastart, 6, inputFile);
    WeatherReport* weatherSummary = readWeather(inputFile, threshhold);
    printReport(weatherSummary, sitename, threshhold);
}
```

	Test	Expected	Got	

✔	`./processtemps dailyminmaxtempschch2017.txt 30.0`	`Dates when 30.0 C was reached at Christchurch Aero`
		
		`Date MaxTemp`
		`12/01/2017 30.2 C`
		`01/02/2017 31.0 C`
		`07/02/2017 30.1 C`
		`03/12/2017 30.4 C`
		`06/12/2017 30.7 C`
		`09/12/2017 32.5 C`
		`10/12/2017 31.0 C`
		`11/12/2017 30.5 C`
		`26/12/2017 30.9 C`
			`Dates when 30.0 C was reached at Christchurch Aero`
			
			`Date MaxTemp`
			`12/01/2017 30.2 C`
			`01/02/2017 31.0 C`
			`07/02/2017 30.1 C`
			`03/12/2017 30.4 C`
			`06/12/2017 30.7 C`
			`09/12/2017 32.5 C`
			`10/12/2017 31.0 C`
			`11/12/2017 30.5 C`
			`26/12/2017 30.9 C`
				✔

Passed all tests! ✔

Correct

Marks for this submission: 4.00/4.00.

◄ Superquiz 1 Jump to... Quiz 1: Linux and C Basics (Practice copy) ►