RYERSON UNIVERSITY

DEPARTMENT OF ENGINEERING AND ARCHITECTURAL SCIENCE

FOR

CPS 125 SECTION 15

DIGITAL COMPUTATION AND PROGRAMMING

SEMESTER/ YEAR: Semester 2 (2020-2021)

INSTRUCTOR: Manar Alalfi

[redacted]

TITLE OF PROJECT: CPS 125 Final Project Report

AUTHORS OF THIS REPORT:

Stanley Tan - [redacted]

Stuti Patel - [redacted]

Syed Ammar Ali- [redacted]

Umaima Mehbuba [redacted]

Sarah Asmat

Performed: March 31-April 11, 2021

Submitted: April 11, 2021

Introduction:

The purpose of this assignment is to apply the skills learned during this semester within one cumulative project. This report analyses the methods used to tackle each individual problem. Using analysis skills to author code based on specified assigned tasks, the lessons learned within this course can be expressed. Problem 1 was completed by Umaima Mehbuba. Problem 2 was assigned to and completed by Syed Ammar Ali. Problem 3 was completed by Stanley Tan and Stuti Patel.

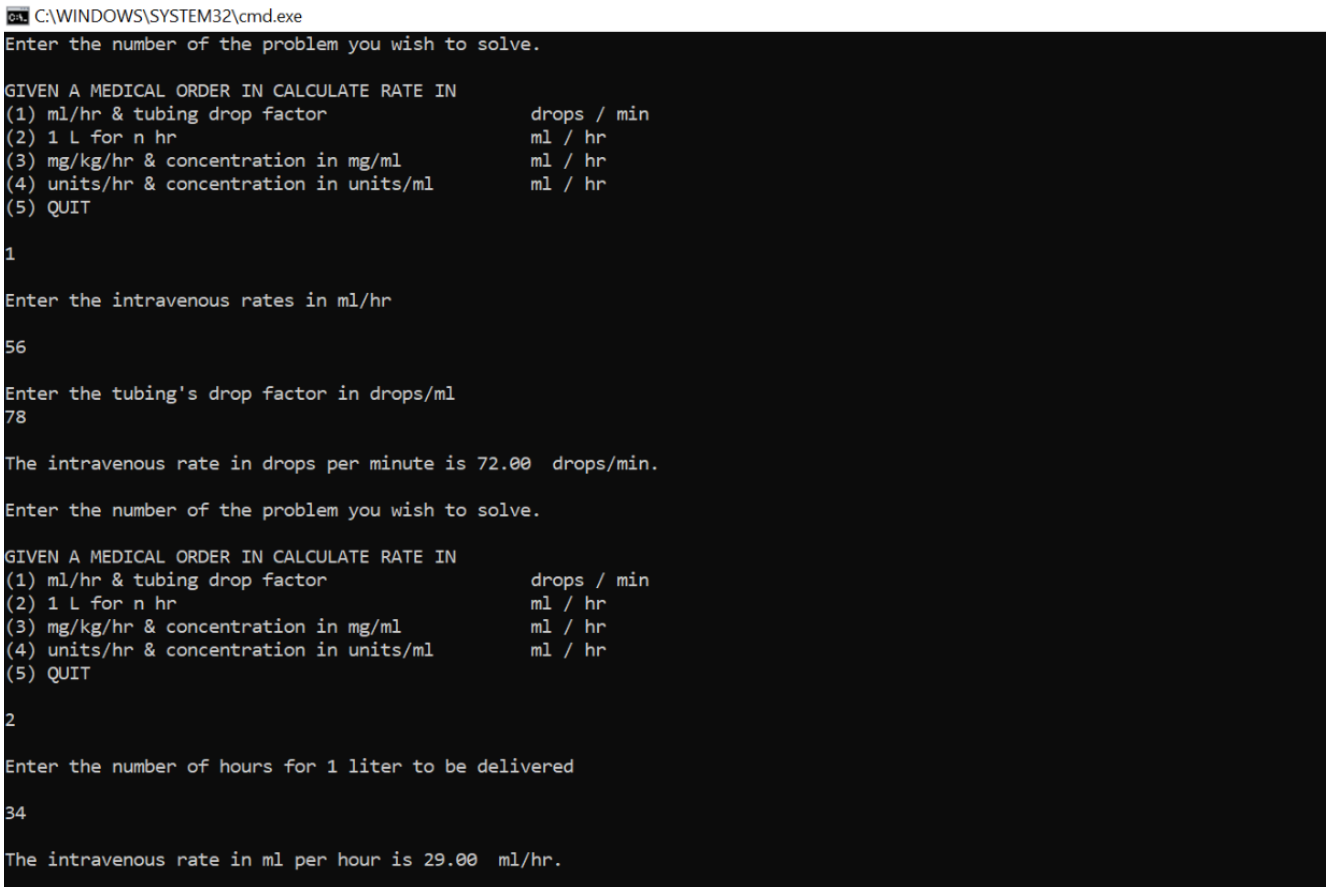
Problem 1:

Code

```c
#include <stdio.h>
#define MINUTE 60 /* Number of minutes in an hour*/
#define M_LITER 1000
#define SENTINEL 5

int get_problem(void);
double get_n_hours(double num_hours);
void get_rate_drop_factor(double *, double *);
void get_kg_rate_conc(double *, double *, double *);
void get_units_conc(double *, double *);
double fig_drops_min(double, double);
double fig_ml_hr(double);
double by_weight(double, double, double);
double by_units(double, double);

int main(void){
int value; /* option chosen*/
     double answer; /* return value of functions*/
     double ml_hour; /* rate in ml/hr*/
     double drops_ml; /* tubing factor in drops/ml*/
     double mg_kg_hour; /* rate in mg/kg/hr*/
     double pat_weight; /* patient's weight in kg*/
     double mg_ml; /* concentration in mg/ml*/
     double units_hour; /* rate in units/hr*/
double units_ml; /* concentration in units/ml*/
     double num_hours; /* number of hours for 1 L to be
delivered*/


     value = get_problem();
     while( value!=5) {
     switch(value) {
          case 1:
          ml_hour=0;
          drops_ml=0;
          answer=fig_drops_min(ml_hour,drops_ml);
          printf("\nThe intravenous rate in drops per minute is
%.2lf  drops/min.\n\n",answer );
          break;
          case 2:
          num_hours=0;
          answer=fig_ml_hr(num_hours);
          printf("\nThe intravenous rate in ml per hour is %.2lf
ml/hr.\n\n" ,answer);
          break;
          case 3:
          mg_kg_hour=0;
          pat_weight=0;
          mg_ml=0;
          answer=by_weight(mg_kg_hour,pat_weight, mg_ml);
          printf("\nThe intravenous rate in ml per hour is %.2lf
ml/hr.\n\n",answer );
          break;
          case 4:
          units_hour=0;
          units_ml=0;
          answer=by_units( units_hour,units_ml);
          printf("\nThe intravenous rate in ml per hour is %.2lf
ml/hr.\n\n", answer);
          break;
          case 5:
          break;
          default:
          printf("Wrong input.\n\n");
     }
     value = get_problem();
}
}
int get_problem(void){
     int menu_number;
     printf("Enter the number of the problem you wish to
solve.\n\n");
     printf("GIVEN A MEDICAL ORDER IN CALCULATE RATE IN\n");
     printf("(1) ml/hr & tubing drop factor\t\t\t drops /
min\n");
     printf("(2) 1 L for n hr\t\t\t\t ml / hr\n");
     printf("(3) mg/kg/hr & concentration in mg/ml\t\t ml /
hr\n");
     printf("(4) units/hr & concentration in units/ml\t ml /
hr\n");
     printf("(5) QUIT \n\n");
     scanf("%d",&menu_number);
     return (menu_number);
}
double get_n_hours(double num_hours){
     printf("\nEnter the number of hours for 1 liter to be
delivered\n\n");
     scanf("%lf",&num_hours);
     return(num_hours);
}
void get_rate_drop_factor(double *ml_hour, double *drops_ml) {
     printf("\nEnter the intravenous rates in ml/hr\n");
     scanf("%lf",&*ml_hour);
     printf("\nEnter the tubing's drop factor in drops/ml\n");
     scanf("%lf", &*drops_ml);
}
void get_kg_rate_conc(double *mg_kg_hour, double *pat_weight,
double *mg_ml){
     printf("\nEnter the intravenous rates in mg/kg/hr\n");
     scanf("%lf",&*mg_kg_hour);
     printf("\nEnter the  patient's weight in kg\n");
     scanf("%lf", &*pat_weight);
     printf("\nEnter the  concentration of drug in mg/ml\n");
     scanf("%lf", &*mg_ml);
}
void get_units_conc(double *units_hour, double *units_ml){
     printf("\nEnter the intravenous rates in units per
hour\n");
     scanf("%lf",&*units_hour);
     printf("\nEnter the concentration of drug in units per
ml\n");
```

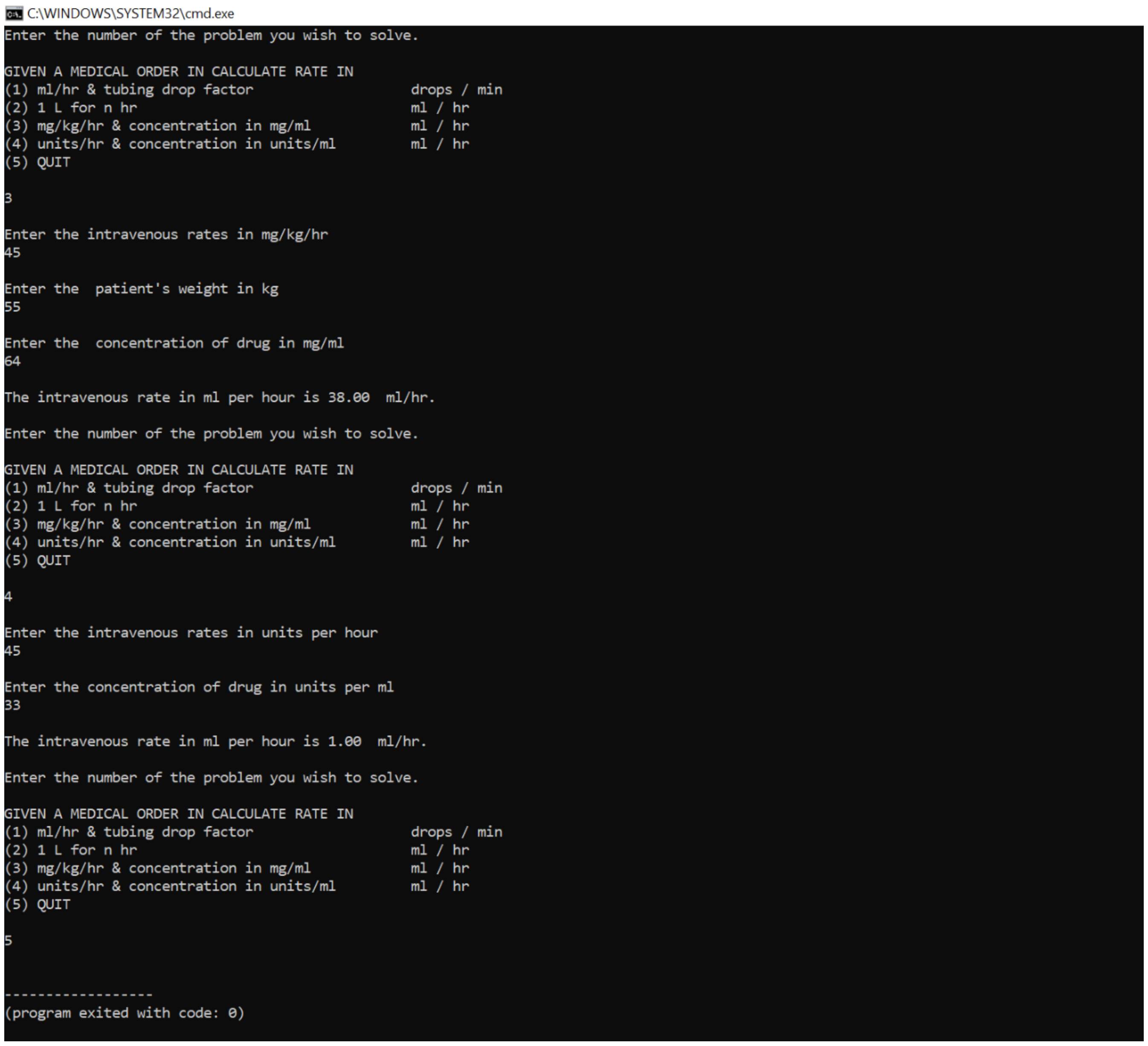
```
	scanf("%lf", &*units_ml);
}
double fig_drops_min(double ml_hour, double drops_ml) {
	get_rate_drop_factor(&ml_hour,&drops_ml);
	return((int)((ml_hour/MINUTE)*(drops_ml)));
}
double fig_ml_hr(double num_hours){
	return((int)(1000/get_n_hours(num_hours)));
}
double by_weight(double mg_kg_hr, double pat_weight, double
mg_ml){
	get_kg_rate_conc(&mg_kg_hr, &pat_weight, &mg_ml);
	return((int)(((mg_kg_hr)*(pat_weight))/(mg_ml)));
}
double by_units(double units_hour, double units_ml){
	get_units_conc(&units_hour,&units_ml);
	return((int)((units_hour)/(units_ml)));
}
```

Sample Output:



C:\WINDOWS\SYSTEM32\cmd.exe

```
Enter the number of the problem you wish to solve.

GIVEN A MEDICAL ORDER IN CALCULATE RATE IN
(1) ml/hr & tubing drop factor                  drops / min
(2) 1 L for n hr                                ml / hr
(3) mg/kg/hr & concentration in mg/ml           ml / hr
(4) units/hr & concentration in units/ml        ml / hr
(5) QUIT

1

Enter the intravenous rates in ml/hr

56

Enter the tubing's drop factor in drops/ml
78

The intravenous rate in drops per minute is 72.00  drops/min.

Enter the number of the problem you wish to solve.

GIVEN A MEDICAL ORDER IN CALCULATE RATE IN
(1) ml/hr & tubing drop factor                  drops / min
(2) 1 L for n hr                                ml / hr
(3) mg/kg/hr & concentration in mg/ml           ml / hr
(4) units/hr & concentration in units/ml        ml / hr
(5) QUIT

2

Enter the number of hours for 1 liter to be delivered

34

The intravenous rate in ml per hour is 29.00  ml/hr.
```



```
C:\WINDOWS\SYSTEM32\cmd.exe
Enter the number of the problem you wish to solve.

GIVEN A MEDICAL ORDER IN CALCULATE RATE IN
(1) ml/hr & tubing drop factor                   drops / min
(2) 1 L for n hr                                 ml / hr
(3) mg/kg/hr & concentration in mg/ml            ml / hr
(4) units/hr & concentration in units/ml         ml / hr
(5) QUIT

3

Enter the intravenous rates in mg/kg/hr
45

Enter the  patient's weight in kg
55

Enter the  concentration of drug in mg/ml
64

The intravenous rate in ml per hour is 38.00  ml/hr.

Enter the number of the problem you wish to solve.

GIVEN A MEDICAL ORDER IN CALCULATE RATE IN
(1) ml/hr & tubing drop factor                   drops / min
(2) 1 L for n hr                                 ml / hr
(3) mg/kg/hr & concentration in mg/ml            ml / hr
(4) units/hr & concentration in units/ml         ml / hr
(5) QUIT

4

Enter the intravenous rates in units per hour
45

Enter the concentration of drug in units per ml
33

The intravenous rate in ml per hour is 1.00  ml/hr.

Enter the number of the problem you wish to solve.

GIVEN A MEDICAL ORDER IN CALCULATE RATE IN
(1) ml/hr & tubing drop factor                   drops / min
(2) 1 L for n hr                                 ml / hr
(3) mg/kg/hr & concentration in mg/ml            ml / hr
(4) units/hr & concentration in units/ml         ml / hr
(5) QUIT

5


------------------
(program exited with code: 0)
```

Description of Code:

This C Program is made to assist users with intravenous rate calculations. The program interacts with the user and prompts the user to select the number of the program they wish to solve and enter in the required values for the problem.

The program utilizes a set of functions to read in values and send the data to the calling module using output parameters. Another set of functions are used to call the previous functions by reference where the desired computations are made and returned as function values. A sentinel controlled loop using switch statement is used in the main program to display the menu and print the answers to the problem selected.

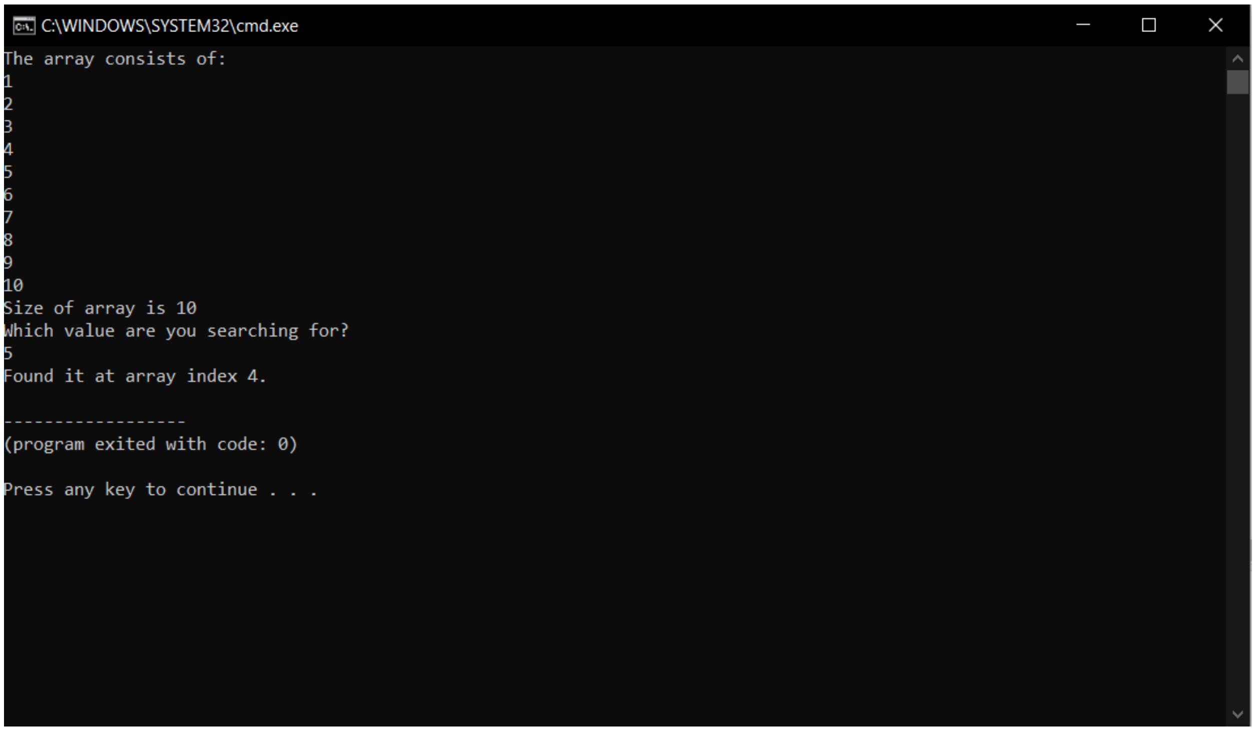
The program output begins by displaying the menu. Once a number is selected, the program will prompt the user to enter in required values. When ‘1’ is selected, the program prompts the user to enter in rate in ml/hr and the tubing's drop factor then displays intravenous rate in drops per minute. When ‘2’ is selected, the program prompts the user to enter the number of hours for 1 liter to be delivered and displays the intravenous rate in ml/hr. When ‘3’ is selected, the program prompts the user to enter rate in mg/kg/hr, patient's weight in kg, and concentration in mg/ml and displays the intravenous rate in ml/hr. When ‘4’ is selected, the program prompts the user to enter rate in units/hr and concentration in units/ml then displays the intravenous rate in ml/hr. Once a problem is solved, the program will loop and prompt the user to select another problem until ‘5’ is selected, and the program exits. If any other number is selected, the program will display “Wrong Input” and the menu will be displayed again to select another option.

Problem 2:

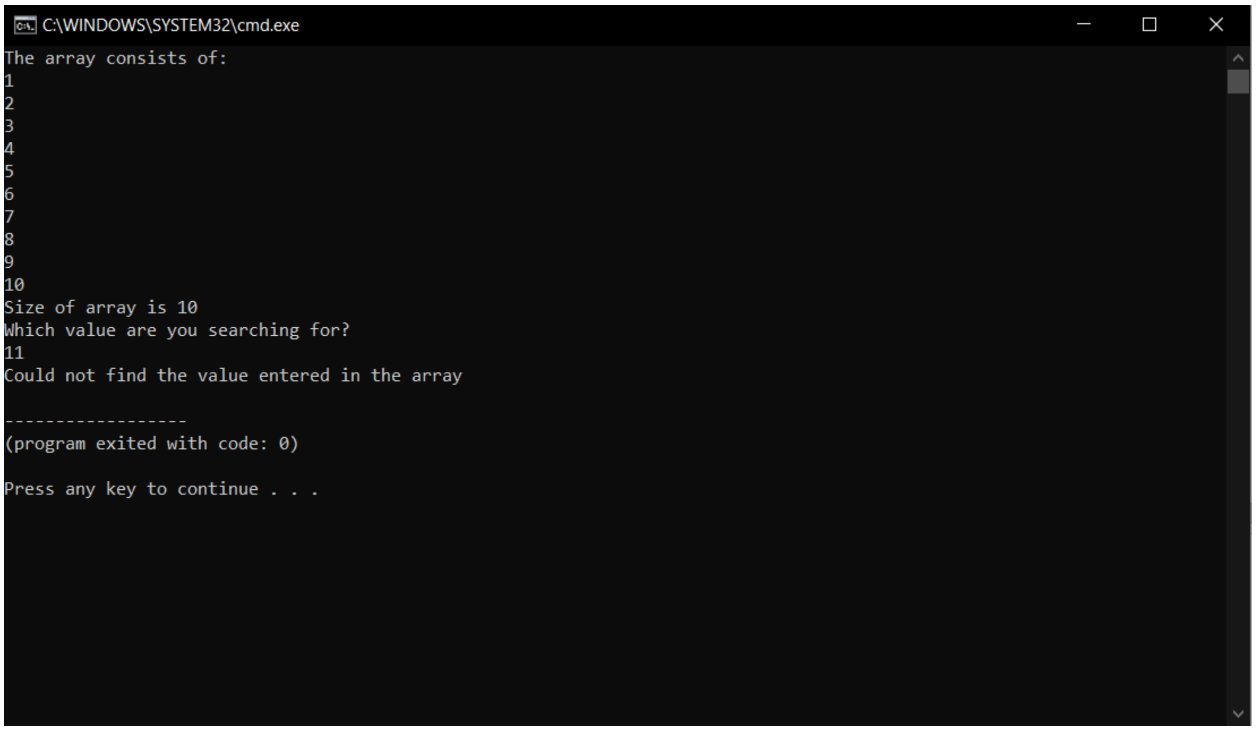
Code:

```
#include <stdio.h>
#include <math.h>
void binary_srch(const int search_array[], int target, int
size);
int main (void)
{
     int target;
     int array[]={1,2,3,4,5,6,7,8,9,10};
     printf("The array consists of:\n");
     for(int i=0;i<10;i++){
          printf("%d\n",array[i]);
     }
     printf("Size of array is
%d\n",(int)(sizeof(array)/sizeof(array[0])));//Get rid of this
     printf("Which value are you searching for?\n");
     scanf("%d",&target);
     binary_srch(array,target,sizeof(array)/sizeof(array[0]));
     return(0);
}

void binary_srch(const int search_array[], int target, int
size){
     int top, bottom, middle, index, found;
     bottom=0;
     top=size-1;
```

```
	found=0;
	while(!(bottom>top)&&found!=1){
		middle=(top+bottom)/2;
		if(search_array[middle]==target){
			found=1;
			index=middle;
		}
		else if(search_array[middle]>target){
			top=middle-1;
		}
		else{
			bottom=middle+1;
		}
	}
	if(bottom>top){
		found=-1;
	}
	if(found==1){
		printf("Found it at array index %d.", index);
	}
	else if(found==-1){
		printf("Could not find the value entered in the
array");
	}
}
```



```
C:\WINDOWS\SYSTEM32\cmd.exe
The array consists of:
1
2
3
4
5
6
7
8
9
10
Size of array is 10
Which value are you searching for?
5
Found it at array index 4.

------------------
(program exited with code: 0)

Press any key to continue . . .
```



```
C:\WINDOWS\SYSTEM32\cmd.exe
The array consists of:
1
2
3
4
5
6
7
8
9
10
Size of array is 10
Which value are you searching for?
11
Could not find the value entered in the array

------------------
(program exited with code: 0)

Press any key to continue . . .
```

Description of Code:

The Array being searched was declared and initialised to consist of integers from 1 to 10, sorted from the lowest to highest value. Therefore the value 1 exists at index 0, value 2 exists at index 1, and this trend repeats for the rest of the array. This allows for the algorithm to perform a binary search by comparing the target value to the middle value to determine if the target value exists near the top or bottom of the specified index range within the array. The process continues until the target value is found, or until the bottom index of the range(stored in the variable bottom) becomes greater than the top index of the array(stored in the variable top).

A boolean statement declared within the while loop in the function allows for the detection for when the entirety of the array has been searched, therefore ending the loop as the target value was not found within the array. This while loop consists of three if statements that reevaluate the position of the middle value within the range based on the comparison between the middle value within the specified range of indexes and the target value.

If the value of the middle matches the target then the loop is exited and the if statement following the while loop declares that the target value was found within the array, and at which index it was discovered. If the value is greater than the middle value it will choose the greater half of the array(excluding the already checked middle value), and then repeat the checking process until the target value is discovered or until the entire array has been searched. The if statement also checks if the target value is lower than the middle and searches the array accordingly. If the entire array is searched but the target value is not found, an if statement will check the value of the found variable and will declare that the target value could not be found. It should be noted that this algorithm will work with any integer array as long as it is sorted from lowest to highest.

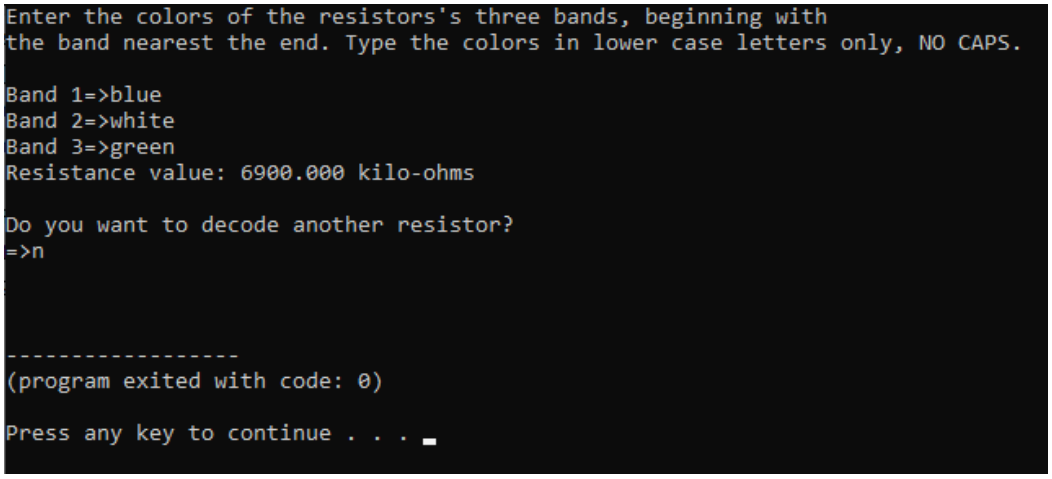
Problem 3:

CODE

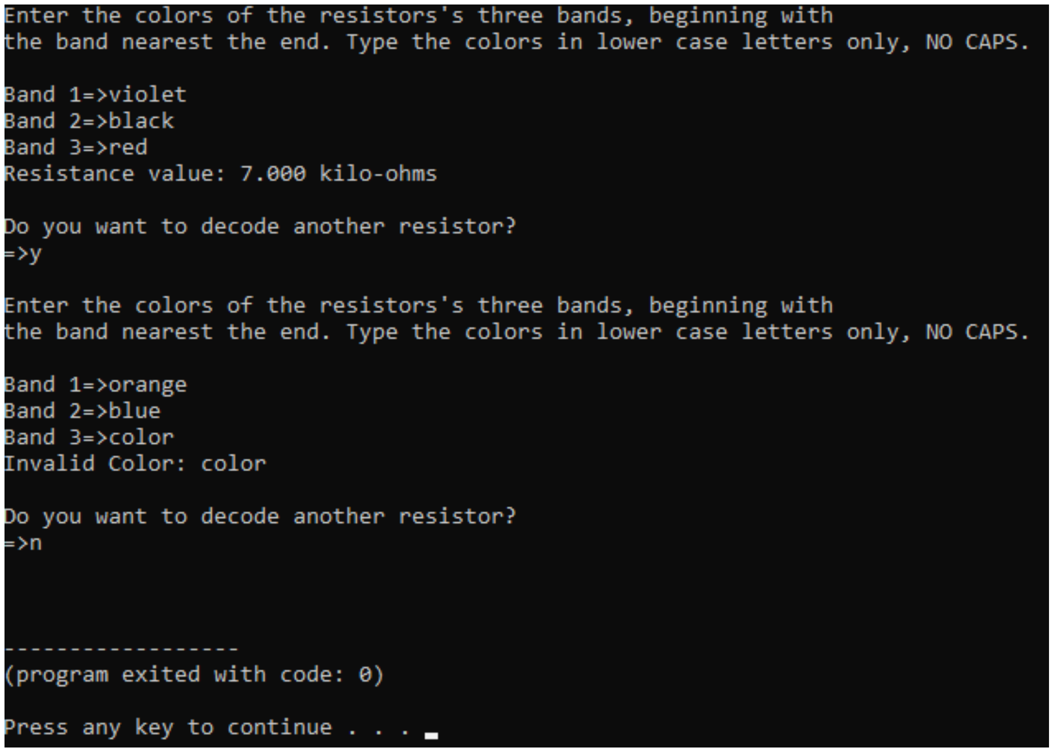
```
#include <stdio.h>
#include <math.h> /* for pow*/
#include <ctype.h> /* for toupper*/
#include <string.h> /* for strlen*/
#define NOT_FOUND -1 /* constants */
#define SUB_1 10
#define SUB_2 7

int search(const char[][SUB_2], const char[], int);

int main (void){
     char reply,
     char_left;
     int i;
     int counter;
     int value;
     double answer = 0.0;
     int no_error = 1;
     char COLOR_CODES[SUB_1][SUB_2] =
{"black","brown","red","orange","yellow","green",
"blue","violet","gray","white"};
char target[SUB_2];

     do{
          printf("Enter the colors of the resistors's three
bands, beginning with \n");
          printf("the band nearest the end. Type the colors in
lower case letters only, ");
          printf("NO CAPS.\n\n");

          for(counter = 1; counter <= 3; counter++){
               printf("Band %d=>", counter);
               scanf("%s", target);

               value = search(COLOR_CODES,target,SUB_1);

               if(value != NOT_FOUND){
                    switch(counter){
                         case 1:
                                   answer = value*10;
                                   break;
                         case 2:
                                   answer += value;
                                   break;
                         case 3:
                                   if (value>3)
                                        answer*=pow(10,value-3);
                                   else
                                        for (i=0; i < 3-value ;
i++)
                                        answer /= 10;
}
                         }
                     else
                         no_error = 0;
          }
          if (no_error == 1)
               printf("Resistance value: %.3f
kilo-ohms\n\n",answer);
```

```
          else
               printf("Invalid Color: %s\n\n", target);

          printf("Do you want to decode another resistor?\n=>");
          scanf("%c%c", &char_left, &reply);
          printf("\n");
          }while(reply == 'y');
     }

int search(const char COLOR_CODES[][SUB_2], const char target[],
int size)
{
     int i, j;
     int length, counter = 0;
     int where = 0 ;

     length = strlen(target);
     for (i = 0; i < size ; i++){
                    for (j = 0; j < length; j++)
                              if(COLOR_CODES[i][j] == target[j])
                                        counter++ ;
                    if(counter == length){
                              return i;
                    } else
                              counter = 0;
     }
     --i;
     return where = NOT_FOUND;
}
```

SAMPLE RUNS



```
Enter the colors of the resistors's three bands, beginning with
the band nearest the end. Type the colors in lower case letters only, NO CAPS.

Band 1=>blue
Band 2=>white
Band 3=>green
Resistance value: 6900.000 kilo-ohms

Do you want to decode another resistor?
=>n



------------------
(program exited with code: 0)

Press any key to continue . . .
```



```
Enter the colors of the resistors's three bands, beginning with
the band nearest the end. Type the colors in lower case letters only, NO CAPS.

Band 1=>violet
Band 2=>black
Band 3=>red
Resistance value: 7.000 kilo-ohms

Do you want to decode another resistor?
=>y

Enter the colors of the resistors's three bands, beginning with
the band nearest the end. Type the colors in lower case letters only, NO CAPS.

Band 1=>orange
Band 2=>blue
Band 3=>color
Invalid Color: color

Do you want to decode another resistor?
=>n



------------------
(program exited with code: 0)

Press any key to continue . . .
```

Description of Code:

This specific code simply calculates the value of a 3 band resistor. The following program utilizes mainly arrays, functions, and strings. By inputting the three colour bands on the resistor, the main code can output the value. The main code consists of a do while loop which calculates the value. The first two colour bands of the resistor are also the first two numbers of the value, and the final colour band represents the 10 multiplier. This can be seen in the sample code above.

In order to find the value of the colour band, a helper function, "Search", will be used which finds the index of the input string that matches the 2D array with all the colours. All these colours are matched with their index, which is the equivalent of their colour value. This helper function matches each character of the input string and the 2D array with all the colours to find the colour index. This index value will be inputed, however if the input string does not match any of the strings in the 2D array, the function will result NOT_FOUND, which is the equivalent of -1.

This program repeats itself in the do while loop until the user decides not to continue to decode another resistor.

Conclusion

For Problem 1: This project was a great learning experience. Doing this project helped me understand the functions and function calls a lot better. However, a few challenges were encountered along the way. It was difficult to figure out the passing of function values from one function to another instead of directly passing to the main program. If I were to do this project again, I would simply avoid using so many functions, and instead directly passing values from the void functions to the main function. This would definitely be a more efficient way to code the program.

Conclusion for Problem 2: In conclusion, while working on Problem 2, very few issues were encountered while coding. The only issue that was encountered was how to approach the output for each situation. When choosing the value for the found variable. The value of 0 appeared to serve no purpose, therefore it was decided that the if statements should only consider the value of found when found is equal to 1 or -1. This issue was quickly resolved and the rest of the problem, including authoring the search algorithm, was quite simple.

For Problem 3: During my experiment, I have found it very frustrating as the code did not proceed to run the way I intended it to. In order to fix this problem, I have modified the code so the desired final result was produced. If I were to do this again, I would have simply just done a while loop and try to shorten the code in order to find a more efficient way to run the program.