HAND IN

Answers recorded on exam paper

Student number								

QUEEN'S UNIVERSITY FINAL EXAMINATION

FACULTY OF ENGINEERING & APPLIED SCIENCE

APSC 142 – Introduction to Computer Programming for Engineers

Dr. E. Morin (Sec. 100); Mr. M. Mohamad (Sec. 101); Dr. B. Lynch (Sec. 102)

April 16, 2015, 2:00 pm

INSTRUCTIONS TO STUDENTS:

1. This examination is 3 HOURS in length. **NO AIDS ARE ALLOWED.**
2. Please read all questions carefully and answer in the space provided on the examination paper.
3. Do not separate pages from this exam paper; **write your student number in the space provided at the top of each page**.
4. Please use scrap paper to draft your solutions, then copy your completed solutions neatly to the space provided on the exam paper. You may hand in your scrap paper, but it will not be marked.
5. Comments in your C code are not expected and will not be marked.
6. For full marks, your C code must be reasonably efficient as well as correct. Global variables are not permitted on this exam.
7. Except on questions for which robot C is indicated, write your code in standard C.

GOOD LUCK!

PLEASE NOTE: "Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer exam questions as written.

Question 1 (12 marks)		Question 4 (10 marks)	
Question 2 (12 marks)		Question 5 (15 marks)	
Question 3 (20 marks)		Question 6 (15 marks)	
Marker Initials		TOTAL (84 marks)	

Student number:								

1. Program Comprehension (12 marks)

1.A (2 marks)

In the box below, show the screen output that would be produced by the following program:

```
#include <stdio.h>
void main()
{
      int i = 2;
      float x,y;
      x = i/4.0 + 0.5 + (int)0.5;
      y = 6 + 1/2 + 5%3;

      printf("x = %.1f\n",x);
      printf("y = %.2f",y);
}
```

1.A

Student number:								

1.B (5 marks)

In the box below, show the screen output that would be produced by the following program:

```
int function (int &x, int &y, int z);

task main()
{
     int x = 5, y = 5, z=5;
     int f = function(x,y,z);
     displayTextLine(1,"(x,y,z)=(%d,%d,%d)",x,y,z);
     displayTextLine(2,"f=%d",f);

}

int function (int &x, int &y, int z)
{
     int i;
     i = x;
     i = 10;
     y = 10;
     z = 10;
     return (x+y+z);
}
```

1.B

Student number:								

1.C (5 marks)

In the box below, show the screen output that would be produced by the following program. You can use the chart provided below to trace your code:

```
#include<stdio.h>
void function (int a[], int N);

void main()
{
      int i;
      int a[5] = {1,2,3,4,5};
      function (a,5);

      for (i = 0; i < 4; i++)
            printf("%d,",a[i]);
      printf("%d",a[4]);
}

void function (int a[], int N)
{
      int i;
      int tmp1 = a[N-2];
      int tmp2 = a[N-1];

      for (i = N-1; i >= 2; i--)
            a[i] = a[i-2];

      a[0] = tmp1;
      a[1] = tmp2;
}
```

Before Loop				
a[]				
0	1	2	3	4
1	2	3	4	5

During Loop					
i	a[]				
	0	1	2	3	4

1.C

Student number:								

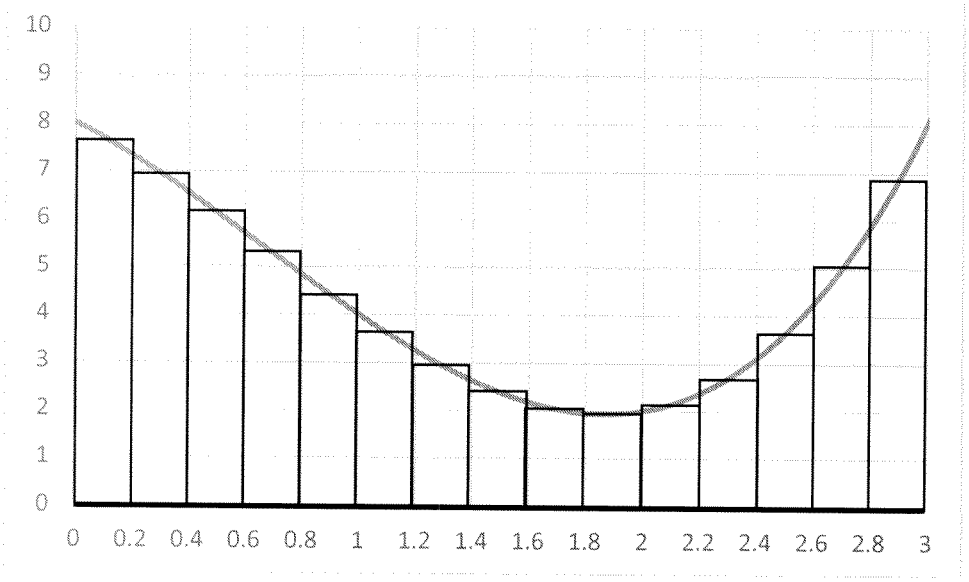
2. Computation and Output (12 marks)

Write a program that uses the Riemann sum method to find the following definite integral:

$$\int_0^3 f(x)dx = \int_0^3 (x^3 - 2x^2 - 3x + 8)dx$$

To do this, use rectangles of width 0.2, from 0 to 3, as shown in the graph below. Calculate the values of $f(x) = x^3 - 2x^2 - 3x+8$ at the mid-points of the rectangles (that is for $x = 0.1, 0.3, \ldots, 2.9$) and store the computed values in an array `f_vals[N]`. As you are doing this, also find the minimum value of $f(x)$ and the x-value at which the minimum value occurs. Then, using the array values, find the definite integral. Print the following output to the screen:

```
The minimum value of XX.XX occurs at x = X.X
The area under f(x) from x=0 to x=3 is: XX.XXX
```



Enter your code on the following page.

Student number:								

```
#include <stdio.h>
#define N _________

void main()
{









}
```

Student number:								

3. Robot Operation (20 marks)

Write a program for the EV3 robot that will navigate a path and react to coloured tiles and obstacles. You must use one function, `CheckColour()`, to measure the tile colour and set the LED output, and another function, `CheckDistance()`, to measure the distance to an obstacle and set the LED output. The function `CheckDistance()` must also return `1` to indicate that the robot should stop under a specific condition.

The left motor is plugged into motor port A and the right motor is plugged into motor port B. The following table lists the connected sensors:

Port	Sensor Name	SensorType
S1	Colour	`sensorEV3_Color`
S2	Ultrasonic	`sensorEV3_Ultrasonic`
S3	Touch	`sensorEV3_Touch`
S4	Touch	`sensorEV3_Touch`

The program must do the following:

1. Keep checking the two front touch sensors until <u>both</u> of them are pressed, then display "Starting!" in big text on the screen for 2 seconds, erase the screen, and move on to the next command.
2. Command the robot to start driving straight for 3600 encoder ticks at a speed of 30, and set the LED colour to orange.
3. While the robot is driving, keep measuring the colour and distance using the functions `CheckColour()` and `CheckDistance()`. Write the two functions so that the robot achieves the following:
    a. If the measured colour is green, set the LED colour to green, otherwise keep it set to orange.
    b. If the measured distance is less than 50 cm and greater than or equal to 30 cm, set the LED colour to red.
    c. If the measured distance is less than 30 cm, set the LED colour to red, and return `1` from `CheckDistance()` to indicate that the robot should stop (otherwise return `0`).
4. After finishing the straight drive (or stopping due to an obstacle), display "Finished!" in big text on the screen for 2 seconds and end the program.

Enter your code on the followng 2 pages.

Student number:								

```
void CheckColour();
int CheckDistance();  // returns 1 if distance less than 30 cm

task main()
{



      // set sensor types (fill in the rest):
      SensorType[S1] = sensorEV3_Color;



      SensorMode[S1] = modeEV3Color_Color;
      // wait for touch sensor input before starting

















      while (                                                    )
      {

















      }


}
```

Student number:								

```
// functions:
```

Student number:								

4. Functions and 1-D Arrays (10 marks)

You have been approached by a biologist who would like you to write a program to discover patterns in DNA sequences. A DNA sequence is represented as an array of characters where each character represents one of the four nucleic bases that form the DNA sequence. The four nucleic bases are Adenine, Cytosine, Guanine, and Thymine, represented with the characters **A**, **C**, **G**, **T**. An example of a DNA sequence is shown below:

```
char DNA[10] = {'A','C','C','G','A','G','T','G','A','G'}
```

The input to your program will be the DNA sequence and a pattern to find within this DNA sequence; the pattern is a smaller DNA sequence. The program has to determine the number of occurrences of the pattern within the DNA sequence. For example:

```
char pat1[2] = {'A','G'}     char pat2[4] = {'C','C','G','G'}
```

It can be seen that `pat1` exists twice within the DNA sequence. However, `pat2` has zero occurrences within the DNA sequence. You will write a function **`findPattern()`** which returns the number of times a pattern appears in a DNA sequence (0 is returned if the pattern does not exist within the DNA sequence).

Below is the function prototype declaration.

```
int findPattern(int DNA[], int n, int pat[], int m);

// DNA[] is the DNA sequence being searched
// n is the size of the DNA sequence
// pat[] is the pattern you are searching for
// m is the size of the pattern
```

The code for the `main()` function is given on the following page. Complete the `findPattern()` function.

Student number:								

```
#include <stdio.h>
int findPattern(char DNA[], int n, char pat[], int m);

void main()
{
      int times_found;
      char DNA[10] = { 'A','C','C','G','A','G','T','G','A','G'};
      char pat[3] = {'A','C','G'};

      times_found = findPattern(DNA,10,pat,2);
      if (times_found > 0)
            printf("Pattern found %d times",times_found);
      else
            printf("Pattern not found");
}
```

```
int findPattern(int DNA[], int n, int pat[], int m)

{









}
```

Student number:								

5. 2-D Arrays (15 marks)

Write a program that uses three functions that each take an input array, manipulate it, and store the result in a new array:

1. `flipHorizontal(int input[][N], int output[][N])`
   This function flips the array horizontally, reversing the order of the columns.
2. `flipVertical(int input[][N], int output[][N])`
   This function flips the array vertically, reversing the order of the rows.
3. `transposeArray(int input[][N], int output[][N])`
   This function flips the array along the diagonal, turning each row into a column.

The arrays contain `N` number of rows and `N` number of columns (a global constant initialized with `#define`), and are `int` types.

The main program should initialize an input array with the values below and three empty arrays of the same size. The functions should then be called to fill the three empty arrays with the horizontally flipped input array, vertically flipped input array, and transposed input array (the results are shown for clarity).

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Original array

4	3	2	1
8	7	6	5
12	11	10	9
16	15	14	13

Horizontally flipped array

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

Vertically flipped array

1	5	9	13
2	6	10	14
3	7	11	15
4	8	12	16

Transposed array

Enter your code on the following 2 pages.

Student number:								

```
#define N 4



int main()
{









}
```

Student number:								

```
// functions
```

Student number:								

6. Algorithms: Sorting and Searching (15 marks)

Nine students are registered in a class. The three-digit student id numbers are stored in the array, `class_ids[]`. The professor wants to sort the array in ascending order for marks entry. Subsequently, another professor asks if students with id numbers 445 and 530 are taking the class.

Write a function to use either selection sort, or bubble sort, to sort `class_ids[]` in ascending order. Write a second function which uses binary search to search for a specific id number; if the id number is found, return a value of 1 (true), if not found, return a value of 0 (false). The search function should be called twice, once to search for #445 and once to search for #530. After each search, print "Student XXX is in the class" or "Student XXX is not in the class" depending on the result of the search.

```
#include <stdio.h>
#define N 9
//function prototypes




void main()
{
  int class_ids[]={321, 445, 585, 782, 246, 229, 644, 392, 512};









}
//Enter functions on next page
```

Student number:								

```
//functions
```

APSC 142 – Final Examination Information Sheets
C Functions and Program Operators and Structures

Robot C commands and built-in functions:

SensorType[] SensorMode[] sensorEV3_Color sensorEV3_Touch sensorEV3_Ultrasonic sensorEV3_Gyro sensorSound_DBA	abs(float val) cos(float radians) sin(float radians) sgn(float x) sqrt(float x) exp(float x) log(float x) log10(float x)	sleep(int msecs) wait1Msec(int msecs) random(long range)

```
setMotorSpeed(int motor, int speed)
getMotorRunning(int motor)
getMotorEncoder(int motor)
moveMotorTarget(int motor, int ticks, int speed)
waitUntilMotorStop(int motor)
getUSDistance(int port)
getColorName(int port)
getColorReflected(int port)
getTouchValue(int port)
playTone(int freq, byte 10msec_tics)
setPixel(int x, int y)
clearPixel(int x, int y)
displayTextLine(int line, " ", ...)
displayStringAt(int line, string " ", ...)
eraseDisplay()
```

Built-in colour codes and LED settings:

colorNone	colorYellow	ledOff	ledOrangeFlash
colorBlack	colorRed	ledRed	ledOrangePulse
colorBlue	colorWhite	ledRedFlash	ledGreen
colorGreen	colorBrown	ledRedPulse	ledGreenFlash
		ledOrange	ledGreenPulse

Operators:

- parentheses: ( ) - innermost first
- unary: + - ++ -- !
- binary: * / % + -
- relational binary: < <= > >= == !=
- logical binary: && ||
- assignment: = += -= *= /=

Repetition structures:

```
while (condition)
{
   ... statements;
}
```

```
do
{
  ... statements;
} while (condition);
```

```
for (exp_1; exp_2; exp_3)
{
  ... statements;
}
```

Selection structures:

```
if (condition)
{ ...  statments
when
  condition is true;
}
else
{ ... statements
when
  condition is
false;
}
```

```
if (condition_1)
{ block 1}
else if
(condition_2)
{ block 2}
.
.
else if
(condition_n)
{block n}
else
{block_of_code}
```

```
switch( variable )
case val_1:
{ statements;
  break; }
case val_2:
{ statements;
  break; }
.
.
case val_n:
{ statements;
  break; }
```

Arrays:

```
1D array: vals[N] = {vals[0], vals[1], ... vals[N-1]};
2D array: vals[N][M] = {{vals[0][0], vals[0][1], ... vals[0][M-1]},
                        {vals[1][0], vals[1][1], ... vals[1][M-1]},
                         ...
                        {vals[N][0], vals[N][1], ... vals[N][M-1]}};
```

Functions:

- function types - byte, int, float, void
- prototype: functionType functionName (parameters);
- function:

```
functionType functionName (parameters)
{
    block_of_code
      ...
      return variable; //not included for void function
}
```

- pass by value: e.g., int func(int num)
- pass by reference: e.g., void func(int &num)

Functions and arrays:

- typedef - 1D array: typedef type typeName[N];
- typedef - 2D array: typedef type typeName[ROWS][COLS];
- function prototype with array:
  - void functionName (typeName, other parameters);