HAND IN

Answers recorded on exam paper

Student number:								

Queen's University
Faculty of Applied Science

APSC 142 - Introduction to Computer Programming for Engineers

FINAL EXAMINATION – 19 April 2014, 9:00am

Instructors: Mr. Behnam Behinaein, Dr. Ying Zou, and Dr. Evelyn Morin

This examination is 3 hours in duration.

This is a closed book exam. NO AIDS ARE ALLOWED.

Please do no separate pages from this exam booklet; write your student number at the top of each page.

Read each question carefully, and answer in the space provided.

Please use scrap paper to draft your solutions, then copy your completed solutions, neatly, to the exam paper in the space provided for each question. (Do not hand in your rough work; it will not be marked. Any rough work that is handed in will be discarded.)

Comments in your C code are not expected and will not be considered in marking.

For full marks, C code must be reasonably efficient as well as correct. Global variables are NOT allowed on this exam.

Instructors may not be available during the exam. Proctors are unable to respond to queries about the interpretation of exam questions. Do your best to answer the exam questions as written.

Question 1 (12marks)		Question 4 (16 marks)	
Question 2 (14 marks)		Question 5 (16 marks)	
Question 3 (16 marks)		Question 6 (16 marks)	
		TOTAL (90 marks)	
		Marker Initials	

1. Program Comprehension (12 marks)

1.A (4 marks)

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

```
task main()
{
    int i;
    float x, y;
    float result1,result2;

    x=5.1;
    y=4.9;
    i=0.5;

    result1=1/2*x+y;
    result2=i*(x+y);
    nxtDisplayTextLine(0, "result1=%0.1f",result1);
    nxtDisplayTextLine(1, "result2=%0.1f",result2);
}
```

1.A

1.B (4 marks)

In the box on the next page, show the screen output that would be produced by the following program.

```
task main()
{
    int i;
    int arr[6];

    for (i=0; i < 6; i++)
    {
        arr[i]=i+i;
    }
    for (i=0; i < 6; i+=3)
    {
        nxtDisplayTextLine(i, "arr[%d]=%d",i,arr[i+1]);
    }
}
```

1.B

1.C (4 marks)

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

```
void function(int &val1, int &val2);
task main()
{
    int i,j;
    i=1;
    j=2;
    nxtDisplayTextLine(0, "i=%d,j=%d",i,j);
    function(i,j);
    nxtDisplayTextLine(1, "i=%d,j=%d",i,j);
}

void function(int &val1, int &val2)
{
    int var;
    var=val1;
    val2=val1;
    val1=var;
}
```

1.C

2. Computation and Output (14 marks)

Complete the following program to calculate the minimum and maximum values of y for the polynomial:

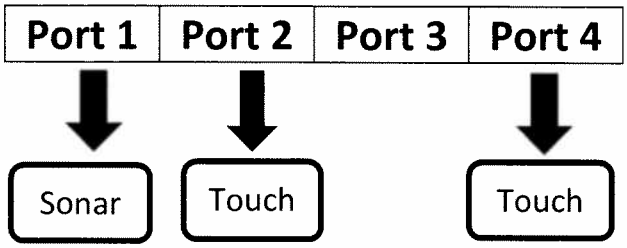
$y = 5x^3 + 2x^2 - 15x + 3$ at $x = 0.0, x = 0.2, x = 0.4, ..., x = 5.0$

That is, the values of x are separated by 0.2 between a start value of 0.0 and an end value of 5.0. The program should display the minimum and maximum values of y and the corresponding values of x that lead to these values.

```
task main()
{
// x is an input and y is an output of the polynomial
   float x, y;

// xMin is the value of x that leads to the minimum value of y (yMin)
   float xMin, yMin;

// xMax is the value of x that leads to the maximum value of y (yMax)
   float xMax, yMax;

// delta defines distance between two consecutive x values
   float delta = 0.2;

// complete the rest of the code
```

3. NXT Robot Operation (16 marks)

A robot is placed at point A, 200 cm away from point B. There is a wall at point B. The robot has 4 ports as shown below, with sensors attached as shown.



Complete the code on the following pages to do the following:

- Navigate the robot from point A towards point B at full speed.
- Using the sonar sensor, constantly display the distance from the wall.
- When the robot reaches a distance of 40 cm from the wall, reduce the speed to 30% of max speed.
- If either of the touch sensors bumps into the wall, stop the robot and display the image.

```
 #define FULL_SPEED 100
 #define ROWS 16  //number of rows in the image
 #define COLS 16  //number of columns in the image
 #define LEFT 40  //horizontal screen position at left edge of image
 #define TOP 40   //vertical screen position at top edge of image

// function prototypes
 int readTouchSensor();
 void displayImage();

 task main()
 {
    int bump=0;// stores the return value of  readTouchSensor function
    int distance;//stores the distance from the wall

    //initialize the sensor types;




// Continuously read the sonar and touch sensors (call the function
// touchSensor in the while loop and control the speed.

  while (                    )
  {









  }

// Stop the robot and call the function displayImage to display the image


}
```

```
/* This functions reads the touch sensors.
   If either sensor bumps into an object, this function will return 1.
   If neither touch sensor is pressed, this function will return 0.
*/

int readTouchSensor()
{









}
```

```
/* This function draws the image stored in the array "image" on the screen
*/

void displayImage()
{
    int i, j; //row and column
    int x, y; //screen x and y coordinates
    byte image[ROWS][COLS]=
                {   {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0},
                    {0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0},
                    {0,0,1,1,1,1,1,1,1,1,1,1,1,1,0,0},
                    {0,1,1,1,0,0,1,1,1,1,0,0,1,1,1,0},
                    {0,1,1,0,0,0,0,1,1,0,0,0,0,1,1,0},
                    {1,1,1,0,0,0,0,1,1,0,0,0,0,1,1,1},
                    {1,1,1,0,0,0,0,1,1,0,0,0,0,1,1,1},
                    {1,1,1,1,0,0,1,1,1,1,0,0,1,1,1,1},
                    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
                    {1,1,1,0,1,1,1,1,1,1,1,1,0,1,1,1},
                    {0,1,1,1,0,1,1,1,1,1,1,1,0,1,1,0},
                    {0,1,1,1,1,0,1,1,1,1,1,0,1,1,1,0},
                    {0,0,1,1,1,1,0,0,0,0,1,1,1,1,0,0},
                    {0,0,0,1,1,1,1,1,1,1,1,1,1,0,0,0},
                    {0,0,0,0,0,0,1,1,1,1,0,0,0,0,0,0} };

 for(                                  )
   {
      for(                                  )
        {
          x = LEFT+j; //set x screen position according to the column value
          y = TOP-i;  //set y screen position according to the row value

//Display pixels here









        }
        wait10Msec(2);
    }
}
```

4. Function and 1D Arrays (16 marks)

Daily maximum temperatures in degrees centigrade for February 2014 in Kingston were recorded, as follows:

1.9, 1.4, -4.1, -5.8, -7.2, -7, -8.3, -7, -8, -7.2, -12.6, -8.4, -4.3, -0.1, -1.8, -9.2, -11.1, -1.6, 3.7, 2.1, 4.8, 3.8, 0.7, -6.2, -7.4, -8.4, -8.2, -12.6

The values are in order by date from Feb. 1 to Feb. 28.

The temperatures are entered in a one-dimensional array, `max_temps`, in a C program. Complete the program to do the following:

a. calculate the average temperature, *avg*, in task main, where *avg* is obtained by summing over all temperature values (T_i, i= 0,...,N-1) and dividing by the total number of values (N):

b. $$avg = \frac{1}{N}\sum_{i=0}^{N-1} T_i$$

c. display the average value on the LCD screen

d. find the median temperature value: The median value of an array, is the value in the middle of the sorted array; for example the median of {3, 5, **6**, 9, 12} is 6 and the median of {1.2, 1.8, **3.6**, **4.2**, 4.9, 6.4} is the average of the two values in the middle: (3.6+4.2)/2 = 3.9. To find the median value of `max_temps`:
   - copy the array `max_temps` into a second array `sorted_temps`; pass `sorted_temps` to the function `sortArray` to sort the temperature values from minimum to maximum (use selection sort or bubble sort)
   - in `task main`, find the median value from the sorted array; include code to find the median if the number of values, N, in the array is odd or even, where:
     - if N is odd, find the middle index of the array and get the value at that position
     - if N is even, find the two middle indices and average the values in those positions

e. display the median temperature on the LCD screen

Complete the skeleton code given on the following two pages.

```
#define N 28
typedef float Float1D[31];
// add function prototypes



task main()
{
// variable declarations - declare any other variables used in the program
    float max_temps[] = {1.9, 1.4, -4.1, -5.8, -7.2, -7, -8.3, -7,
                          -8, -7.2, -12.6, -8.4, -4.3, -0.1, -1.8,
                        -9.2, -11.1, -1.6, 3.7, 2.1, 4.8, 3.8, 0.7,
                        -6.2, -7.4, -8.4, -8.2, -12.6};
    float sorted_temps[N];



// calculate and display the average temperature









// copy the values in max_temps into sorted_temps and call sortArray



// find the median value if N is even or odd and display the median value
// on the LCD screen

   if (              )



    else



}
```

```
/* complete the functions sortArray and swap to do a selection or bubble
   sort */

void sortArray (Float1D temps)
{
// declare two loop counters, (i,j) and variable to hold index of min value
    int i, j, minIndex;


// sort array from min to max









// call swap if minIndex!=i (outer loop counter)









}

void swap (int ind_1, int ind_2, Float1D list)
{









}
```

5. 2-D Arrays (16 marks)

A service for hosting on-line picture albums provides functionality that automatically recognizes an object (e.g., a person's face) in a digital picture. This functionality eases the searching for pictures. A two-dimensional (i.e., 2D) array is used to represent a digital picture in black and white. An example of a digital picture that shows a star shape is shown below.

```
byte pic[9][9] = {    {1,0,0,0,1,0,0,0,1},
                      {0,1,0,0,1,0,0,1,0},
                      {0,0,1,0,1,0,1,0,0},
                      {0,0,0,1,1,1,0,0,0},
                      {1,1,1,1,1,1,1,1,1},
                      {0,0,0,1,1,1,0,0,0},
                      {0,0,1,0,1,0,1,0,0},
                      {0,1,0,0,1,0,0,1,0},
                      {1,0,0,0,1,0,0,0,1}};
```

Each pixel of the picture is represented by one element in the 2D array. An element in the 2D array has two possible values: 1 sets a pixel to black, and 0 sets a pixel to white.

In this question, you will write a simplified program that checks if a picture contains a given image pattern. Two examples of the image patterns are given below:

```
byte pat1[5][5] = {                  byte pat2[5][5] = {
                    {0,1,1,1,0},                         {0,1,0,1,0},
                    {1,1,1,1,1},                         {1,1,1,1,1},
                    {0,1,1,1,0},                         {0,1,1,1,0},
                    {1,0,1,0,1},                         {1,0,1,0,1},
                    {0,0,1,0,0}};                        {0,0,1,0,0}};
```

`pat1` can be found in the star picture as shown in bold in the 2D array pic above. `pat2` cannot be found in the star picture. You will write a function **`findPattern()`** that searches for a pattern in a picture, and returns **1 (true)** when the **first** occurrence of the pattern is detected and returns **0 (false)** when the pattern cannot be found in the picture.

Below are constant declarations and function prototype definition:

```
#define WIDTH 9    //define the width of a picture which you want to
                   //search for a pattern
#define HEIGHT 9  //define the height of a picture which you want to
                  //search for a pattern
#define COLS 5    //define the number of the columns of a 2D array that
                  //is used to store a pattern
#define ROWS 5   //define the number of the rows of a 2D array that is
                 //used to store a pattern
typedef byte Picture[HEIGHT][WIDTH];  //define a new type for storing
                                      //a picture
typedef byte Pattern[ROWS][COLS]; // define a new type for storing a
                                  //pattern
byte findPattern(Picture, Pattern); // search a Pattern in a Picture.
```

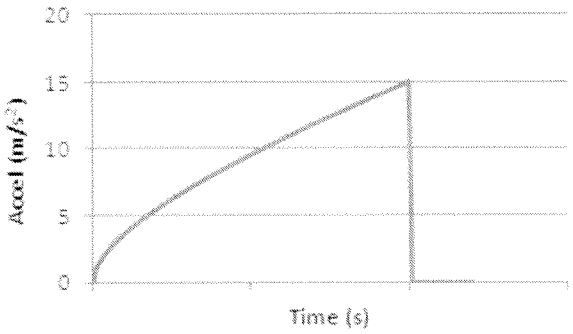
The code for `task main()` is given below. Complete the function `findPattern` on the following page.

```
#define WIDTH 9
#define HEIGHT 9
#define COLS 5
#define ROWS 5

typedef byte Picture[HEIGHT][WIDTH];
typedef byte Pattern[ROWS][COLS];
byte findPattern(Picture, Pattern);

task main()
{
      byte isFound;
      byte pic [9][9] = {
                        {1,0,0,0,1,0,0,0,1},
                        {0,1,0,0,1,0,0,1,0},
                        {0,0,1,0,1,0,1,0,0},
                        {0,0,0,1,1,1,0,0,0},
                        {1,1,1,1,1,1,1,1,1},
                        {0,0,0,1,1,1,0,0,0},
                        {0,0,1,0,1,0,1,0,0},
                        {0,1,0,0,1,0,0,1,0},
                        {1,0,0,0,1,0,0,0,1}
                        };
      byte pat[5][5]= {
                        {0,1,0,0,1},
                        {0,0,1,0,1},
                        {0,0,0,1,1},
                        {1,1,1,1,1},
                        {0,0,0,1,1} };

      isFound=findPattern(pic, pat);
      if (isFound == 1)
          nxtDisplayTextLine(2, "the pattern is found");
      else
          nxtDisplayTextLine(2, "the pattern is not found");
}

/* On the next page, complete the function to search for the pattern (pat)
   in the Picture (pic) */
```

```
byte findPattern(Picture pic, Pattern pat)
{




}
```

6. Computation and Numerical Methods (16 marks)

The acceleration of a small rocket was measured using an accelerometer. It was found that the acceleration could be modeled using the equation: $a(t) = \sqrt{t} + 0.05t$. When the rocket reached a maximum acceleration of $15 m/s^2$ the acceleration dropped to zero. The acceleration profile is shown below:



Complete the code on the following pages to:

a. find the time in seconds at which the rocket reaches maximum acceleration
b. determine the velocity of the rocket at 10 second intervals from t=0s to t = 120s, (i.e., velocity at 10s, 20s, ..., 120s) and store the values in an array rocketVelocity[n], where n is the number of values to be stored. To find the velocities calculate the definite integral of the acceleration function; use Riemann sum integration with a step size of 1 second. (Note: after the acceleration drops to zero, the rocket is travelling at constant velocity.)

```
task main()
{
// declare any other variables used in your program



    float accel, time, maxAccelTime;
    float rocketVelocity[12];
    time = 0;                          // set initial time to zero
    accel = 0;                         // initialize acceleration value

// find maxAccelTime = time at which acceleration equals 15 m/s*s









// display maxAccelTime on LCD screen



// integrate accel function using Reimann sum to get velocity up to t=120s
// store velocity value at each 10second interval in rocketVelocity[i]

    area = 0;          // initialize area to 0
    i = 0;             // rocketVelocity array index

















// display value of final velocity on LCD screen



}
```

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

Robot C commands and built-in functions:

motor[] SensorType[] SensorValue[] sensorTouch sensorSONAR sensorLightActive sensorSoundDBA	abs(float val) cos(float radians) sin(float radians) sgn(float x) sqrt(float x) exp(float x)	wait1Msec(int msecs) wait10Msec(int msecs) random(long range) log(float x) log10(float x)
PlayTone(int freq, byte 10msec_tics nxtSetPixel(int x, int y) nxtClearPixel(int x, int y) nxtDisplayTextLine(int line, " ", ...) nxtDisplayClearTextLine(line) nxtDisplayString(int line, string quoted_str) nxtDisplayStringAt(int x, int y, string quoted_str) eraseDisplay()		

Operators:

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

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; }

Repetition structures:

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

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 fund(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);