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

Dec 17, 2015, 9:00 am

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.

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 (12 marks)	
Question 2 (12 marks)		Question 5 (12 marks)	
Question 3 (12 marks)		Question 6 (12 marks)	
		Question 7 (6 marks)	
Marker Initials		TOTAL (90 marks)	

Student number:								

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()
{
     float f = 2.6;
     int i;

     i = f;
     displayTextLine(1, "%d, %0.2f", i, f);

     f = i * f;
     i = i * i;
     displayTextLine(1, "%d, %0.1f", i, f);
}
```

1.B (4 marks)

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

```
task main()
{
     int i = 0, j = 1;

     do {
        displayTextLine(i, "%d", j );
        i++;
        j = j + i;
     } while (j < 11) ;
}
```

Student number:								

1.C (4 marks)

The function below is supposed to arrange the two passed-in integer variables, swapping them if necessary so that upon return, val1 is not greater than val2. If the values were swapped, then the function returns a 1; otherwise, it returns a 0.

In the box below, write the lines of C code that will complete the function.

```
int swap(int &val1, int &val2)
{

        return swapped;
}
```

Student number:								

2. Computation and Output (12 marks)

Write a program that calculates values of the polynomial $2x^3 - 100x^2 + 200x - 35$ at x=0, x=0.1, x=0.2, ... , x=3.0 (that is, for values of x spaced apart by 0.1 between a start value of 0 and an end value of 3.0). Your program should then display the maximum value of the polynomial in that same range [0, 3], as well as the value of x at which the maximum occurs.

```
task main()
{











}
```

Student number:								

3. Robot Operation (12 marks)

Write a program in RobotC that spins the EV3Bot in place while measuring distance to obstacles it sees around it using its SONAR sensor on port index 1. The Bot should eventually complete ***one full 360 rotation*** by rotating in increments of 10 degrees, measuring SONAR distance after each 10-degree rotation. If the measured sonar distance **d** is below 40 centimetres, the Bot should emit a 50-millisecond duration tone whose frequency **f** is given by this expression **f = 1600-20(d)**.

When the 360-degree rotation is completed, the Bot should report if ***any*** distance it measured was less than 40 cm by reporting either "TOO CLOSE" or "OK".

Your program **MUST use** the following pre-defined function by calling it as needed to accomplish your task:

void Rotate(int degrees) – rotates at fixed speed for the specified number of degrees

NOTES:

1) **DO NOT define any new functions of your own creation, you must use only the predefined one, and any built-in RobotC functions needed.**
2) **DO NOT define the given Rotate() function – assume it is done by someone else.**

Student number:								

Question 3: Use the space in the following box to write the rest of the program to carry out the task described on the preceding page.

```
void Rotate(int degrees);
task main() {

     // GIVEN DECLARATIONS
     int distance;

     //ADD ANY EXTRA DECLARATIONS YOU MIGHT NEED NEXT


     //DO SONAR SENSOR TYPE INITIALIZATION HERE


    //SPIN A TOTAL OF 360 DEGREES IN INCREMENTS OF 10 DEGREES
    //EMITTING A TONE EACH TIME IF THE DISTANCE IS BELOW 40 CM


     //REPORT EITHER "TOO CLOSE" OR "OK"


 }
```

Student number:								

4. Functions and 1-D Arrays (12 marks)

An experimental robot-arm's data acquisition system records up to 1024 integer X-position and Y-position measurements into a file, which are then read by a **main()** program into two arrays (**x[]** and **y[]**). You have to compute the distance from the origin for each point and store in another floating point array **d[]**.

x[0] and **y[0]** is the position at time 0, and you have to compute distance **d[0]**.
x[1] and **y[1]** is the position at time 1, and you have to compute distance **d[1]**.

-
-
-

x[1] and **y[1]** is the position at time 1, and you have to compute distance **d[1]**.

The distance is from the origin is given by the formula $d = \sqrt{x^2 + y^2}$. The main program computes the distance by calling a **distances()** function to which it will pass the **x[]**, **y[]** and **d[]** array. N (the length of the arrays) is a defined constant at the top of the program (value is 1024). The **distances()** function does not have a return value.

On the next page, fill in the boxes provided to:

1) Write a function call from **main()** to the **distances()** function.
2) Write the definition (i.e. body) of the **distances()** function.

Assume that the following lines exist at the top of **main()**:

typedef int INT1D[1024];

typedef float FLOAT1D[1024];

Student number:								

Write a one line function call from **main()** to **distances()** in this box:

Write the definition (i.e. body) of the **distances()** function in this box:

Student number:								

5. 2-D Arrays (12 marks)

You are part of a landmine clearing agency whose job is to remove mines from previous wars to make the area safe for civilians. You have a map of the area which is represented as a two dimensional array. Each element of the array gives the number of mines in that area of the map. For example, the value 5 at row 4, column 6 means that there are 5 mines in that area that have to be removed. However, a map has been discovered that is also a map of the same minefield. It is suspected that one area has a different number of mines. Your job is to write a program that will compare both maps and find the difference between the maps (if there is a difference).

The map array types are declared using the typedef statement:

typedef int Int2D[255][255];

Your main program will call the following function:

int findMapDifference(Int2D map1, Int2D map2, int & row, int & col);

If there is an element of one array that differs from the corresponding element of the other array (that is, if the maps are different), then the function returns the value true (that is, 1), and the row and column of the different elements are passed back through the parameters **row** and **col**. If all elements are the same (i.e. the maps are the same), then the function returns the value false (i.e. 0), and the **row** and **col** parameters are not changed.

Note: there will be at most one element in the two arrays that are different. (that is, there is only one area that is different between the two maps).

Complete the **main()** program and **findMapDifference()** function on the following page. The **main()** program must call the function and then report the result by prining to the screen the (row,column) position of the difference if found. Otherwise it should report "NO DIFFERENCE".

Student number:								

Complete **main()** in this box:

```
typedef int Int2D[255][255];
int findMapDifference(Int2D map1, Int2D map2, int & row, int &col);
task main()
{
   int ourMap[255][255];
   int foundMap[255][255];
   int result, row, col;
   // Assume the map arrays are filled by predefined functions here
   fillOurMap(ourMap);
   fillFoundMap(foundMap);
   // Call the function and report the results









}
```

Complete the definition (body) of **findMapDifference()** in thix box

```
int findMapDifference (Int2D map1, Int2D map2, int &row, int &col)
{





















}
```

Student number:								

6. Simulating a Physical Problem (12 marks)

A rubber ball is dropped off the observation deck of the CN tower on a windless day. Accounting for air resistance and gravity, the ball undergoes a **downward** vertical acceleration given by $a = (9.8 - 0.017v^2)$ where the ball's **downward** velocity v is in units of meters per second.

On the following page, write a main program that models the flight of the rubber ball until it reaches the ground. Your program should use the small-time-step simulation approach where you advance in samm time increments of $\Delta t = \mathbf{0.01}$ seconds, and do repeated computations to find the new downward velocity and vertical position at the end of each time **step until the ball's height drops below 0 (or equals 0).**

For each time step, your program should therefore:

1) compute the corresponding velocity increase $\Delta v = (\mathbf{9.8 - 0,017}v^2) * \Delta t$
2) Compute the final velocity at the end of the time step, $v = vprev + \Delta v$
3) Compute the average velocity vavg = (vprev + v)/2 over the time step
4) Compute the height change $\Delta h = vavg * \Delta t$, and the new height **(h-Δh)** at the end of the time step

The final step of the program is to display the time that the ground was struck.

DO NOT use any functions in your solution.

Student number:								

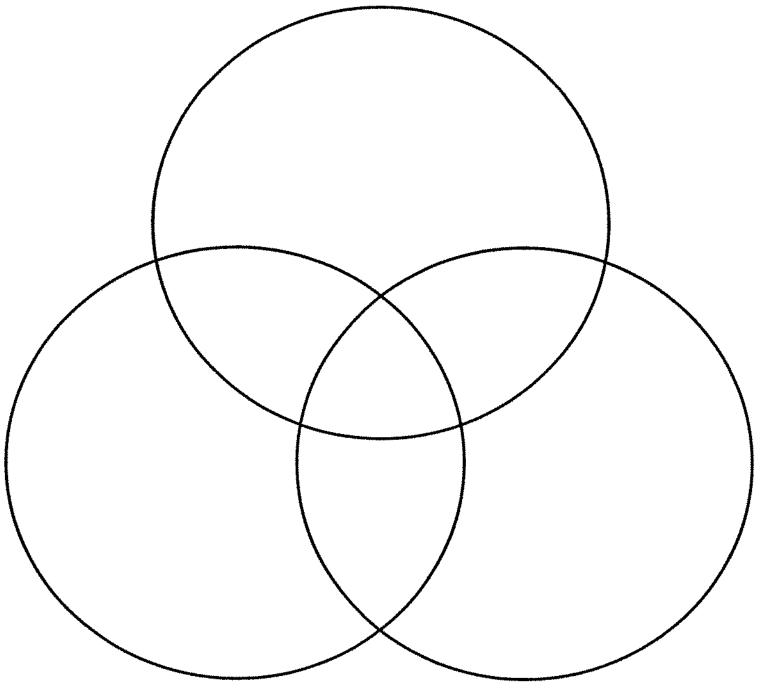
Complete your solution to question 6 in this box:

```
#define HINIT 346 // CN observation deck height (meters)
#define DELTAT 0.01 // time step delta (secs)

task main()
{
    // YOUR DECLARATIONS




    // INITIALIZATION





    // FLIGHT LOOP











   // OUTPUT




}
```

Student number:								

7. Simulating a Physical Problem (6 marks)

In design thinking, there are three main areas to be considered, often represented as a venn diagram. Complete the venn diagram below with the three areas of design thinking, and in the box below the venn diagram, give a short explanation of each of the three areas.



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)`
		`abs(float val)`
		`cos(float radians)`
		`sin(float radians)`
		`sgn(float x)`
		`sqrt(float x)`
		`exp(float x)`
		`log(float x)`
		`log10(float x)`

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

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

Repetition structures:

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

Selection structures:

if (condition)	if (condition_1)	switch(variable)
{ statements when	{ block 1}	case val_1:
condition is true;	else if(condition_2)	{ statements;
}	{ block 2 }	break; }
else	.	case val_2:
{ statementswhen	.	{ statements;
condition is false;	else if(condition_n)	break; }
}	{block n }	.
	else	.
	{block_of_code}	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: int func(int num)
- pass by reference: 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:
    - o void functionName (typeName, other parameters);