ASSIGNMENT 1

HIT365 – C programming

This assignment is worth 5% of the assessment for this unit. Late submissions will not be accepted unless prior arrangements have been made.

Show all working and explanations. Remember to write sentences.

Successfully completion of tutorial exercises is worth 20% of your mark

QUESTION 1 (30 MARKS)

Fix or describe the error in each of the following pieces of code:

a)

float x = 19.23;

float xPrt = &x;

printf("%f\n",xPtr)

Response to Question 1 part a)

To make the code run correctly you would need to include pre-processing directives, a main function. Syntactically, you need to include a * in front of the xPrt variable and include a semi-colon on the end of the printf statement. The code written below executes correctly.

```
#include <stdio.h>

int main(void){

	float x = 19.23;
	float *xPrt = &x;
	printf("%f\n", *xPrt);

	return 0;
}
```

b)

int *x,y;

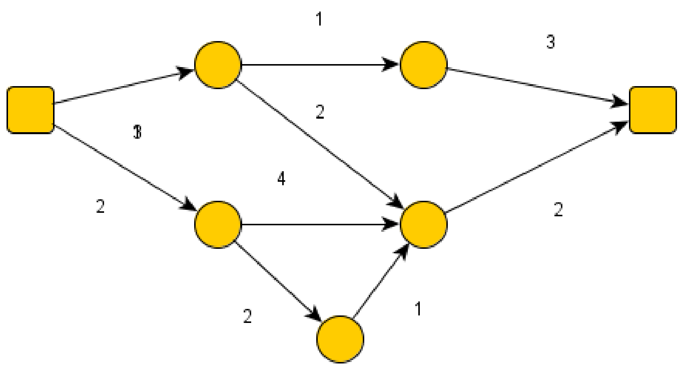
x=y;

Response to Question 2 part b)

To make the code run correctly you would need to include pre-processing directives, a main function. The following code executes successfully.

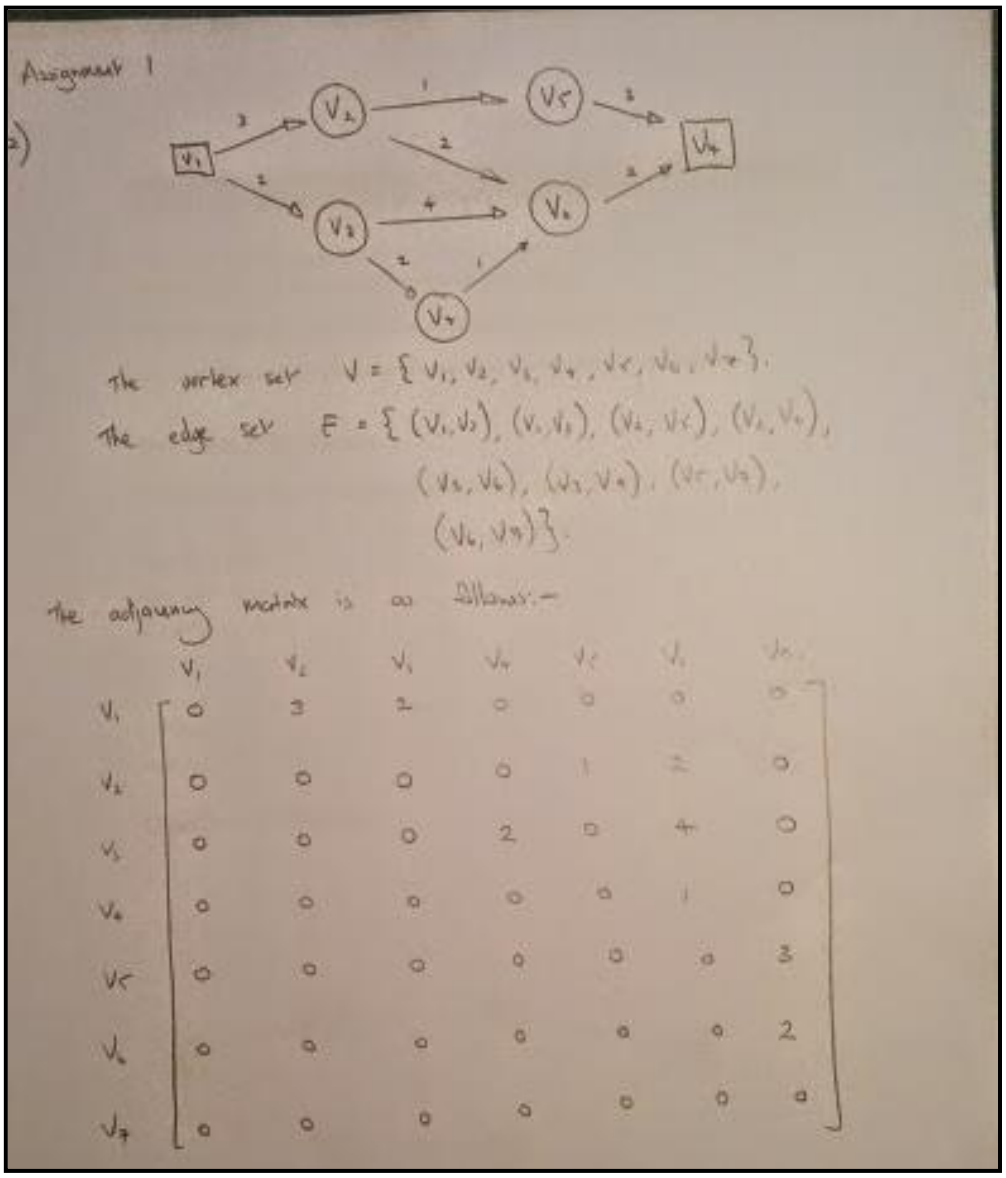
```
#include <stdio.h>

int main(void){
	int *x, y;
	*x = y;
	return 0;
}
```

QUESTION 2 (10 MARKS)



Write a matrix representation of the above digraph

Response



v_1
v_2
v_3
v_4
v_5
v_6
v_7
3
2
1
2
4
2
1
3
2

Assignment 1

2)

The vertex set $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$.

The edge set $E = \{(v_1,v_2), (v_1,v_3), (v_2,v_5), (v_2,v_6),$
$(v_3,v_6), (v_3,v_4), (v_5,v_7),$
$(v_6,v_7)\}$.

the adjacency matrix is as follows:-

$$\begin{array}{c|ccccccc}
 & v_1 & v_2 & v_3 & v_4 & v_5 & v_6 & v_7 \\
\hline
v_1 & 0 & 3 & 2 & 0 & 0 & 0 & 0 \\
v_2 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \\
v_3 & 0 & 0 & 0 & 2 & 0 & 4 & 0 \\
v_4 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
v_5 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\
v_6 & 0 & 0 & 0 & 0 & 0 & 0 & 2 \\
v_7 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

QUESTION 3 (30 MARKS)

Write a function that allows you to input a graph or takes a graph as input and calculates the following

a) The total number of edges

b) The degree of each of the nodes (vertices) in the graph

c) The minimum, maximum and total degree of the nodes in the graph

QUESTION 4 (30 MARKS)

a) Calculate the answer for the graph given in questions 1

b) Write another function that test the function you wrote for Question

c) Call these functions from you main function to test you program.

Response to Question 3 & 4

```
/*
 * Assignment1.c
 * Created on: 08/04/2014
 * Author: Shane
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define SIZE 7 /* SIZE needs to be the size of the square array */

/* Prototype functions */
void printgraphAnalysis(int graph[][SIZE], int rowSize, int colSize);
int getEdges(int graph[][SIZE], int rowSize, int colSize);
int getDegree(int graph[][SIZE], int rowSize, int colSize, int i);
int getMinDegree(int graph[][SIZE], int rowSize, int colSize);
int getMaxDegree(int graph[][SIZE], int rowSize, int colSize);
int totalDegree(int graph[][SIZE], int rowSize, int colSize);
void testFunc(int graph[][SIZE], int rowSize, int colSize);

/* Main function specifies the graph in 2D array form and also
 * calls printgraphAnalysis and testFunc */
int main(void){

	int matrix[SIZE][SIZE] = {{0,3,2,0,0,0,0},
				   {0,0,0,0,1,2,0},
				   {0,0,0,2,0,4,0},
				   {0,0,0,0,0,1,0},
				   {0,0,0,0,0,0,3},
				   {0,0,0,0,0,0,2},
				   {0,0,0,0,0,0,0}};

	int k = sizeof(matrix) / sizeof(matrix[0]);
```

```c
        /* This function doesn't return any values, rather, it simply
         * prints an analysis of the graph */
        printgraphAnalysis(matrix, k, k);

        /* A call to this function will let you know which of the above
         * functions pass and which fail */
        testFunc(matrix, k, k);

        /* Return statement in the main */
        return 0;
}

/* Function prints analysis of the graph as specified by the assignment */
void printgraphAnalysis(int graph[][SIZE], int rowSize, int colSize){

        /* Initialise the variables to be used in the function */
        int i;

        /* Prints information on the number of edges in the graph */
        printf("\nThe number of edges in your graph are: %d\n",
                                   getEdges(graph, rowSize, colSize));

        /* Prints the headers for the degrees for each node table */
        printf("\n\n%s%7s\n\n", "Node", "Degree");

        /* Prints information on the number of degrees for each node */
        for (i = 0; i <= SIZE-1; i++){
                printf("V%d%7d\n", i+1, getDegree(graph, rowSize, colSize, i));
        }

        /* Prints information on the node with the minimum number of degrees */
        printf("\nThe minimum number of degrees on a node is: %d\n",
                                   getMinDegree(graph, rowSize, colSize));

        /* Prints information on the node with the minimum number of degrees */
        printf("\nThe maximum number of degrees on a node is: %d\n",
                                   getMaxDegree(graph, rowSize, colSize));

        /* Prints information on the total number of degrees in the graph */
        printf("\nThe total number of degrees in the graph are: %d\n",
                                   totalDegree(graph, rowSize, colSize));
}


/* This function, given a 2D array input of a graph and the
 * row and col sizes, will get the number of edges present
 * in the graph */
int getEdges(int graph[][SIZE], int rowSize, int colSize){
        int edgeCounter = 0;
        int i, j;

        for (i = 0; i <= rowSize-1; i++){

                        /* The code, for a given row, iterates through the columns */
                        for (j = i; j <= colSize-1; j++){

                                /* This piece of code tests to see if there is a
                                 * non-zero entry in the graph matrix for the [ij]th
                                 * entry to the graph matrix. If there is, then it counts
                                 * this as an edge */
                                if (graph[i][j] != 0){
                                        edgeCounter += 1; /* Counter increase by one */
                                }
```

```c
			}
		}
		return edgeCounter; /* Returns the desired edge value */
}


/* This function will return the number of degrees for a specified node - it takes a 2D array,
 * row and col sizes of the array. Further, it will take an index number i in reference to
 * a specific node. It outputs the number of degrees in a node */
int getDegree(int graph[][SIZE], int rowSize, int colSize, int i){

		int degreeCounter = 0;
		int j;

		/* The code, for a given row, iterates through the columns */
		for (j = 0; j <= colSize-1; j++){

			/* This piece of code tests to see if there is a non-zero
			 * entry in the graph matrix for the [ij]th entry to the graph
			 * matrix. If there is, then it counts this as an edge */
			if (graph[i][j] != 0){
				degreeCounter += 1; /* Degree counter is increase by one if edge present */
			}
		}

		return degreeCounter;
}


/* This function will call the getDegree function for each node in the graph taken as input
 * and will determine the minimum degrees on a node */
int getMinDegree(int graph[][SIZE], int rowSize, int colSize){

		int i, degreeNew;
		int minDegree = SIZE-1;

		/* For loop iterates through each node */
		for (i = 0; i <= colSize-1; i++){

			degreeNew = getDegree(graph, rowSize, colSize, i);

			/* Decision to update a new min degree */
			if (degreeNew < minDegree){
				minDegree = degreeNew;
			}
		}

		return minDegree;
}

/* This function operates exactly the same as the getMinDegree(), however,
 * it will find the maximum degree for all nodes in the graph */
int getMaxDegree(int graph[][SIZE], int rowSize, int colSize){

		int i, degreeNew;
		int maxDegree = 0;

		/* For loop iterates through each node */
		for (i = 0; i <= colSize-1; i++){

			degreeNew = getDegree(graph, rowSize, colSize, i);

			/* Decision to update a new max degree */
		if (degreeNew > maxDegree){
			maxDegree = degreeNew;
		}
	}

	return maxDegree;
}

/* This function finds the total number of degrees present on all nodes */
int totalDegree(int graph[][SIZE], int rowSize, int colSize){

	int i;
	int totalDegree = 0;

	for (i = 0; i <= colSize-1; i++){
		totalDegree += getDegree(graph, rowSize, colSize, i);
	}

	return totalDegree;
}

/* Once all the correct values are specified for a given graph, this function will test
 * whether or not the above functions are performing their desired function accurately */
void testFunc(int graph[][SIZE], int rowSize, int colSize){

	/* Actual correct data value */
	int actNumEdges = 9;
	int actNumDegree[] = {2, 2, 2, 1, 1, 1, 0};
	int actMinDegree = 0;
	int actMaxDegree = 2;
	int actTotalDegree = 9;
	int indicator = 0;
	int i;

	/* Test if the function getEdges works correctly */
	if(actNumEdges == getEdges(graph, rowSize, colSize)){
		printf("\nTest getEdges: Pass\n");
	} else {
		printf("\nTest getEdges: Fail\n");
	}

	/* Test if the function getDegree works correctly */
	for (i = 0; i <= SIZE-1; i++){
		/* For loop iterates through each node and the */
		if(actNumDegree[i] != getDegree(graph, rowSize, colSize, i)){
			printf("\nTest getDegree: Fail\n");
			indicator = 1;
			break;
		}
	}

	/* This is part of the test for correct operation of
	 * getDegree - if the indicator is still set to 0 then the
	 * function passes the test */
	if (indicator == 0){
		printf("\nTest getDegree: Pass\n");
	}

	/* Test if the function getMinDegree works correctly */
	if (actMinDegree == getMinDegree(graph, rowSize, colSize)){
		printf("\nTest getMinDegree: Pass\n");
	} else {
		printf("\nTest getMinDegree: Fail\n");
```

```
        }

        /* Test if the function getMaxDegree works correctly */
        if (actMaxDegree == getMaxDegree(graph, rowSize, colSize)){
                printf("\nTest getMinDegree: Pass\n");
        } else {
                printf("\nTest getMinDegree: Fail\n");
        }

        /* Test if the function totalDegree works correctly */
        if (actTotalDegree == totalDegree(graph, rowSize, colSize)){
                printf("\nTest totalDegree: Pass\n");
        } else {
                printf("\nTest totalDegree: Fail\n");
        }
}
```