CS 2213 Advanced Programming Recitation - Exercise

 (Graphs – graph copy function)

You are given the below code that we implemented in slides to create/read/print a graph. First copy/paste it into a file say **graph.c** and compile/run it.

> ➢ **gcc graph.c –o graph**
> ➢ **graph**

It reads the input graph from stdin, but you can use redirections and read the graph from a file. For this again copy/paste the below graph data into a file **undirectedgraph1.txt** and then run your program as

> ➢ **graph < undirectedgraph1.txt**

Then you are asked to to implement **copy_graph_list(..)** function and test your program… Also before ending the main function, make sure you free the graphs…

As always, make sure you release (free) the dynamically allocated memories if you allocate any memory in your programs. So, before submitting your program, run it with valgrind to see if there is any memory leakage…

Also if you need to debug your program, compile your programs with –g option and then run it with gdb and/or ddd.

undirectedgraph1.txt

```
6 8
1 2 3
1 3 6
2 3 1
2 4 5
3 5 2
4 5 3
4 6 6
5 6 1
```

graph.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef enum {FALSE, TRUE} bool;

#define MAXV 100

typedef struct edgenode {
   int y;
   int weight;
   struct edgenode *next;
} edgenodeT;
```

```c
typedef struct {
   edgenodeT *edges[MAXV+1];
    int degree[MAXV+1];
    int nvertices;
    int nedges;
    bool directed;
} graphT;


main()
{
   graphT    myg1, *myg2=NULL;

   initialize_graph(&myg1, FALSE);
   read_graph(&myg1, FALSE);
   print_graph(&myg1);

   copy_graph_list(&myg1, &myg2);
   print_graph(myg2);
}

copy_graph_list(graphT *g, graphT **newg)
{
   graphT *tmpG;

   tmpG = (graphT *) malloc(sizeof(graphT));
   if(tmpG==NULL) {
      printf("no memory");
      exit(-1);
   }
   initialize_graph(tmpG, FALSE);

   /* your code ..... */


   *newg = tmpG;
}

print_graph(graphT *g)
{
    edgenodeT *pe;
    int i;
    if(!g) return 0;
    for(i=1; i<=g->nvertices; i++) {
       printf("Node %d: ", i);
       pe = g->edges[i];
       while(pe){
            //          printf(" %d", pe->y);
            printf(" %d(w=%d),", pe->y, pe->weight);
            pe = pe->next;
       }
```

```c
        printf("\n");
    }
}

initialize_graph(graphT *g, bool directed)
{
    int i;
    g->nvertices = 0;
    g->nedges = 0;
    g->directed = directed;

    for (i=1; i<=MAXV; i++)
       g->edges[i] = NULL;

    for (i=1; i<=MAXV; i++)
       g->degree[i] = 0;
}

read_graph(graphT *g, bool directed)
{
    int i;
    int m;
    int x, y, w;

    scanf("%d %d",&(g->nvertices),&m);

    for (i=1; i<=m; i++) {
        scanf("%d %d %d",&x,&y, &w);
        insert_edge(g, x, y, w, directed);
    }
}

insert_edge(graphT *g, int x, int y, int w, bool directed)
{
    edgenodeT *pe;
    pe = malloc(sizeof(edgenodeT));
    pe->weight = w;
    pe->y = y;

    pe->next = g->edges[x];
    g->edges[x] = pe;

    g->degree[x]++;
    if (directed == FALSE)
        insert_edge(g, y, x, w, TRUE);
    else
      g->nedges ++;
}

free_graph(graphT *g)
{
    edgenodeT *pe, *olde;
```

```
    int i;
    for(i=1; i<=g->nvertices; i++) {
        pe = g->edges[i];
        while(pe){
            olde = pe;
            pe = pe->next;
            free(olde);
        }
    }
    free(g);
}
```

/* Don't forget to include comments about the problem, yourself and each major
step in your program! */

_____

_

You must submit your work using Blackboard Learn and respect the following rules:

1) All assignments must be submitted as either a zip or tar archive file unless it is a single pdf file.
2) Assignments must include all source code.
3) Assignments must include an output.txt file which demonstrates the final test output run by the student.
4) If your assignment does not run/compile, the output.txt file should include an explanation of what was accomplished, what the error message was that prevented the student from finishing the assignment and what the student BELIEVES to be the underlying cause of the error.

_____

__