Practice 1: Data Structures and Abstract Data Types

PART 2: QUESTIONS ABOUT THE PRACTICE

Paula Samper & Laura de Paz

QUESTION 1

P1_E1
```
gcc -c main.c                       //creates main.o
gcc -c node.c                       //creates node.o
gcc -g -o P1_E1 main.o node.o       //creates the executable named
                                      P1_E1 from main.o and node.o
./P1_E1                             //to execute the file
```

P1_E2
```
gcc -c main.c                       //creates main.o
gcc -c node.c                       //creates node.o
gcc -c graph.c                      //creates graph.o
gcc -g -o P1_E2 main.o node.o graph.o //creates the executable named
                                      P1_2 from main.o, node.o and
                                      graph.o
./P1_E2                             //to execute the file
```

P1_E3
```
gcc -c main.c                       //creates main.o
gcc -c node.c                       //creates node.o
gcc -c graph.c                      //creates graph.o
gcc -g -o P1_E3 main.o node.o graph.o //creates the executable named
                                      P1_3 from main.o, node.o and
                                      graph.o
./P1_E3 FileName.txt                //to execute the file with
                                      FileName
```

QUESTION 2

a) The first implementation is correct. The pointer to the node is correctly allocated and initialized. There are no mistakes neither in the arguments of the functions nor the functions.

b) The second implementation is not correct. There are no mistakes in the allocation or in the initialization, but in the following line:
```
if (node_ini (n) == ERROR) ;
```

the node that is being passed as argument is not correct, since that is the one used in the description of the function in node.c, however, in main.c, the pointer to a node that was initialized is n1, so instead of n, it should be n1.

c) The third implementation is not correct, since in these two lines
```
Node *n1;
if (node_ini (&n) == ERROR)
```
there is a pointer to a node declared, and then an a memory address is passed as argument of the function, and there is no relation between them, so the Node *n1, is not being passed to the function.

QUESTION 3

This prototype cannot be implemented instead of the used function (Node *node_copy(const Node *src);), since the two arguments are not pointers, so the information wouldn´t be changed within the two nodes after leaving the function. This means that even if we copy the information of one node (source node) and paste the information within the other (target node), after leaving the function, the target node would not have the information of the source node.

QUESTION 4

Even though none of the node´s data is being change within the function, the pointer Node * is essential since in order to pass a structure as an argument, all fields of the structure are copied in a local variable. As a lot of time and memory are necessary to copy the structure´s data and to create the local variable, it is better to use a pointer to the structure, so only 8 bytes corresponding to the size of the pointer are copied to the local variable of the function. This process is performed in order to save resources and time.

QUESTION 5

Original function:

```
node *node_copy (const Node *src){
   if (!src) return NULL;
   Node *target = NULL;
   if ( !( target = (Node *)malloc(sizeof(Node)) ) ) return NULL;
   target->id = src->id;
   strcpy(target->name, src->name);
   target->nConnect = src->nConnect;
   return target;
}
```

Status function1:

```
status node_copy(const Node *nSource, Node *nDest){
   if (! nSource || !nDdest) return ERROR;
   nDest ->id = nSource ->id;
   strcpy(nDest ->name, nSource ->name);
   nDest ->nConnect = nSource ->nConnect;
   return nDest;
}
```

Yes, the function status node_copy(const Node *nSource, Node *nDest){
would be valid, but the changes above should be made.

The function STATUS node_copy(const Node* nSource, Node** nDest**)** could not be implemented since the second argument of the function is a double pointer, which means that it stores the address of the address of a Node, instead of a Node.

QUESTION 6

The functions in Appendix 4 should not be public functions, since if they are keep as private, the creator has more control of them, since they can only be accessed from other functions in node.c, and node.h; or graph.c and graph.h.