PART 2: QUESTIONS ABOUT THE PRACTICE

1. We have created a makefile, as it is easier for us to compile the programs and see if they work as they should. These are some of the commands we have used:

gcc -Wall -ansi -pedantic -c

gcc: this command stands for telling the terminal we want to compile a C program.
-Wall: turn on most compiler warnings.
-ansi: turn off certain gcc features that are incompatible with ANSI C.
-pedantic: causes the compiler to comply with the C standard.
-c: if we want to compile but not link the project.

gcc -Wall -ansi -pedantic -o

-o let us define the name of the executable file.

2.

2a.

```
int main() {
	Node *n1;

	n1 = (Node*) malloc(sizeof(Node));
	if (!n1) return EXIT_FAILURE;

	/* Set fields */
	node_setId (n1, -1);
	node_setName (n1, "");
	node_setConnect (n1, 0);

	node_free (n1);
	return EXIT_SUCCESS;
}
```

This function could work correctly, but it is not well implemented because it is not necessary to allocate memory for n1, as we have a function called node_init() created for that purpose. In addition, there are no error checking after setting the different fields of the node.

2b.

```
// In the file node.h
 Status node_init (Node *n);

// In the file node.c
Status node_init (Node *n) {
	n = (Node *) malloc(sizeof(Node));
	if (!n) return ERROR;

	/* init fields */
```

```
        node_setId (n, -1);
        node_setName (n, "");
        node_setConnect (n, 0);
        node_setLabel (n, BLANCO);

        return OK;
}

// In main.c
int main() {
        Node *n1;

        if (node_ini (n) == ERROR)
                return EXIT_FAILURE;

        node_free (n1);
        return EXIT_SUCCESS;
}
```

This function will not work correctly because in main.c the function node_ini has as argument n, but the Node type declared was n1, so that's an error. Also, the function is called node_init, not node_ini as it is in main.c.
In our case, we have implemented node_init in a different way, initializing the different fields by accessing directly to the structure, but we don't think this will suppose a problem, it is just a different style for programming.

2c.

```
// In the file node.h
Status node_init (Node **n);

// In the file node.c
Status node_init (Node **n) {
        *n = (Node *) malloc(sizeof(Node));
        if (*n == NULL) return NULL;
        /* inicializa campos */
        node_setId (*n, -1);
        node_setName (*n, "");
        node_setConnect (*n, 0);
        return OK;
        }

// In main.c
int main() {
        Node *n1;
        if (node_ini (&n) == ERROR)
        return EXIT_FAILURE;

        node_free (n1);
        return EXIT_SUCCESS;
        }
```

In this function we have the same problems as in the previous one. In addition to that, as it is using a double pointer in node_init, it will be necessary to pass a double pointer also in the main program.

3. No, because you are not returning any value in nDest. You will need to add a pointer to that value you are passing, for it to keep the new values you are assigning.

4. Yes, because Node is a structure, and you have to be pointing to it.
5. You should create a node in the main file and after that, pass that new node as an argument to the new function. The new function will copy all the elements of the node and return the pointer to the main function.

```
STATUS node_copy (const Node * nSource, Node * nDest){
      if (!nSource || !nDest) return ERROR;

      strcpy(nSource->name, nDest->name);
      nSource->id = nDest->id;
      nSource->nConnect = nDest->nConnect;
      nSource->label = nDest->label;

      Return OK;
}
```

The other function would be the same, but nDest could be used as a two dimensional node.

6. The private functions for ADT Graph should not be public because those functions are only needed to manage data inside public functions, and the user does not need to use them.
7. Yes, it could be alse Node*, because the thing that is returning is of type Node.