1-

gcc -c node.c

gcc -c p1\_e1.c

gcc -o p1\_e1 p1\_e1.o node.o

Las dos primeras crean un fichero objeto con el respectivo nombre y la extension .o y la tercera linea crea un ejeucutable de nombre p1\_e1 con los objetos pe\_e1.o y node.o.

2-

a) no es correcto porque no sigue el paradigma de programación en cuanto a reservar memoria para el tipo Node (puesto que hay un constructor debería usarlo).

b) Es correcto.

c) Es innecesario el uso de un doble puntero.

3-

No es posible porque el retorno es de tipo STATUS y los argumentos no son punteros, así que aunque se modifiquen los valores no se van a modificar los originales.

4-

No es necesario pero se utiliza porque evita pasarlo por valor, lo que implicaría costos en la memoria y en el tiempo, es más barato hacer una copia de un puntero que la de un nodo.

5-

El puntero a puntero es innecesario.

STATUS node\_copy(const Node\* nSource, Node\* nDest){

if(!Node)return ERROR;

Node \*n = NULL;

n=node\_ini();

if(!n)return ERROR;

n→name=nSource→nombre;

n→id=nSource->id;

n→nConnect=nSource→nConnect;

nDest=n;

return OK;

}

6-

Porque son funciones especificas para el graph.c que solo se usan internamente (dependen de la implementación) y que no son útiles externamente (el usuario no tiene por qué saber cómo se guardan los index de los nodos en la matriz de adyacencia), así que solo se usarán cuando sean llamadas por otras funciones del graph.c.