B2 - Elementary Programming in C

B-CPE-200

SOS linked lists

Linked lists and graphs

SOS linked lists

binary name:	list/graph
repository name:	SOS_linked_lists
language:	C
compilation:	via Makefile, including re, clean and fclean rules
build tool:	Makefile

- The totality of your source files, except all useless files (binary, temp files, obj files,...), must be included in your delivery.
- All the bonus files (including a potential specific Makefile) should be in a directory named *bonus*.
- Error messages have to be written on the error output, and the program should then exit with the 84 error code (0 if there is no error).

SIMPLY LINKED LISTS

DATA STRUCTURE

This is the data structure that we will use to implement the linked lists. Do not modify it !
The two typedefs are used to make the code more readable.

```
typedef struct list_s {
    int data;
    struct list_s *next;
} list_t, node_t;
```

Authorized functions: malloc, free, write

EXERCICES

LIST_IS_EMPTY

```
int list_is_empty(list_t *list);
```

This function returns 1 if the linked list is empty or 0 if it is not.

LIST_ADD_BEGIN

```
list_t *list_add_begin(list_t *list, int data);
```

This function adds a node at the beginning of the linked list “list” with the data passed in parameters.

LIST_ADD_END

```
list_t *list_add_end(list_t *list, int data);
```

This function adds a node at the end of the linked list “list” with the data passed in parameters.

LIST_FREE

```
list_t *list_free(list_t *list);
```

This function frees the memory of the linked list “list”.

LIST_ADD

```
list_t *list_add(list_t *list, int data, int pos);
```

This function adds a node in the list “list” at the position pos with the data passed in parameters.

LIST_GET_ELEM

```
int list_get_elem(list_t *list, int pos);
```

This function returns the data of the linked list “list” at the position pos.
For ease of use, the function should return 0 if the position is not valid.

LIST_FREE_POS

```
list_t *list_free_pos(list_t *list, int pos);
```

This function frees a node in the linked list “list” at the position pos.

The other nodes of the list must be linked together.

LIST_GET_LEN

```
int list_get_len(list_t *list);
```

This function returns the length of the linked list “list”.

LIST_MOD

```
void list_mod(list_t *list, int data, int pos);
```

This function updates a node in the linked list “list” at the position pos with the data passed in parameters.

LIST_PUT

```
void list_put(list_t *list);
```

This function prints the linked list on the standard output.
If the linked list is : 1->3->9->78->63->4 this function needs to display the following:

Terminal

```
~/B-CPE-200> ./list | cat -e
1$
3$
9$
78$
63$
4$
```

GRAPHS

DATA STRUCTURE

This is the data structures that we will use to implement the graphs. Do not modify them !

Notice how there is two different structures. Do you understand why ?

```
typedef struct node_s {
    int data;
    struct node_s **links;
    size_t link_count;
} node_t;

typedef struct graph_s {
    node_t **nodes;
    size_t node_count;
} graph_t;
```

Authorized functions: malloc, free, write

EXERCICES

NODE_CREATE

```
node_t *node_create(int data);
```

This function creates a node with the data passed in parameters.

NODE_DESTROY

```
void node_destroy(node_t *node);
```

This function frees the memory of the node passed in parameters.

GRAPH_CREATE

```
graph_t *graph_create(void);
```

This function creates an empty graph.

GRAPH_DESTROY

```
void graph_destroy(graph_t *graph);
```

This function frees the memory of the graph passed in parameters.

GRAPH_ADD_NODE

```
int graph_add_node(graph_t *graph, node_t *node);
```

This function adds a node to the graph passed in parameters.

GRAPH_ADD_LINK

```
int graph_add_link(node_t *from, node_t *to);
```

This function adds a link between the two nodes passed in parameters.

We are implementing an oriented graph, which means that “from” will be linked to “to”, but not the other way around.

GRAPH_REMOVE_LINK

```
int graph_remove_link(node_t *from, node_t *to);
```

This function removes a link between the two nodes passed in parameters.

GRAPH_PRINT

```
void graph_print(graph_t *graph);
```

This function prints all the nodes of a graph.

We do not care about links for this functions.

GRAPH_DFS

```
void graph_dfs(graph_t *graph, node_t *start);
```

This function prints all the nodes of a graph using a depth-first search algorithm.

Here we do care about links. What if a node already got visited ?
Use the given “queue” structure to help you