System-oriented Programming
Spring 2018

S11

Professor : Philippe Cudré-Mauroux
Assistant : Michael Luggen

Submitted by Sylvain Julmy

Exercise 1

We use a circular array representation for the queue. We use two integer variable *top* and *back* to represent the start and the end of the dequeue.
We slightly modify the struct *Stack* in order to use a *size_t* type instead of an integer for the type of the *member_size* field.

```
// Main structure containing all elements
typedef struct
{
    size_t memberSize;
    int maxElements;
    void* data;
    int top;
    int back;
} Stack;

Stack* stackCreate(size_t memberSize, int maxElements);
void stackDestroy(Stack* s);
void stackPush(Stack* s, void* data);
void stackPop(Stack* s, void* target);
void stackTop(Stack* s, void* target);
```

Then, the modification for *push*, *pop* and *top* are straightforward :

```
void stackPush(Stack* s, void* data)
{
    // check if data is valid; if false, writes to stderr
    assert(data);
    //check is the stack is full
    if (s->top == s->back - 1)
    {
        fprintf(stderr, "Stack is full\n");
        exit(2);
    }
    s->top = (s->top + 1) % s->maxElements;
    //calculate starting location for the new element
    void* target = (char*) s->data + (s->top * s->memberSize);
    memcpy(target, data, s->memberSize);
}
```

```c
void stackTop(Stack* s, void* target)
{
    assert(target);
    if (s->top == s->back)
    {
        printf("Stack is empty\n");
        target = NULL;
        return;
    }
    void* source = (char*) s->data + (s->back * s->memberSize);
    memcpy(target, source, s->memberSize);
}
```

```c
void stackPop(Stack* s, void* target)
{
    assert(target);
    // check if stack is empty
    if (s->top == s->back)
    {
        fprintf(stderr, "Couldn't pop an empty stack\n");
        exit(3);
    }
    s->back = (s->back + 1) % s->maxElements;
    void* source = (char*) s->data + (s->back * s->memberSize);
    memcpy(target, source, s->memberSize);
}
```

Exercise 2

The whole implementation is available just below :

```c
typedef struct Node Node;

// main data structure
struct Node
{
    //payload
    int data;
    // in a graph, a node has an arbitrary number of neighbors
    int capacity;
    int nb_neighbors;
    Node** neighbors;
};

// allocates a new node
Node* newNode(int data)
{
    // Allocate memory for new node
    Node* node = malloc(sizeof(node));
    assert(node != NULL);

    // Assign data to this node
    node->data = data;
    node->neighbors = NULL;
    node->capacity = 10;
    node->nb_neighbors = 0;
    return (node);
}
```

As well with a function to connect two nodes :

```c
// connect two node src -> dst
// directed graph, so dst -> src is not implied
void connect(Node* src, Node* dst)
{
    if (src->neighbors == NULL)
    {
        src->neighbors = malloc(src->capacity * sizeof(Node*));
        assert(src->neighbors != NULL);
    }
    if (src->nb_neighbors == src->capacity)
    {
        src->capacity *= 2;
        src->neighbors = realloc(src->neighbors, src->capacity * sizeof(Node*));
        assert(src->neighbors != NULL);
    }
    src->neighbors[src->nb_neighbors] = dst;
    src->nb_neighbors++;
}
```