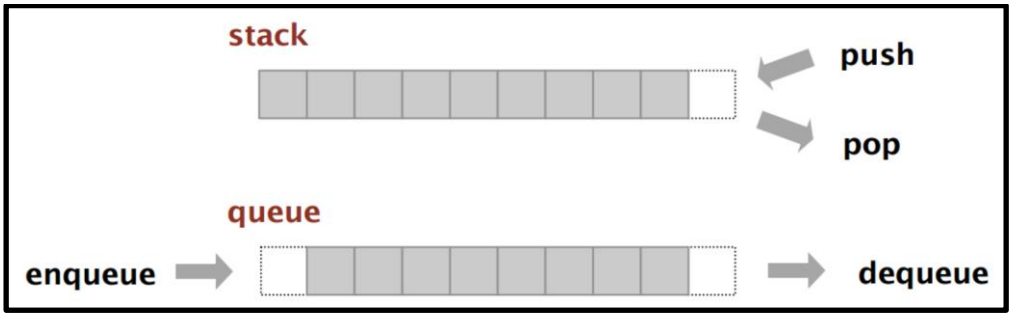
Laboratorio 3

ASD 2020/2021

In questo laboratorio viene richiesto di implementare due tipi di dato (TDD), le **pile** (o stack) e le **code** (o queue), seguendo le seguenti indicazioni (specifiche).



- Implementare il **TDD pila di interi** usando come struttura dati una **lista semplice** (cioè *non* doppiamente collegata, *non* con sentinella, *non* circolare). Il codice da completare contiene un header file riportato sotto, un file ausiliario con le operazioni già predisposte ma non implementate, e un main.
- Implementare il **TDD coda di interi** usando una struttura dati che integri un **array dinamico**, con espansione/contrazione della dimensione dell'array quando necessario. Il codice da completare contiene un header file riportato sotto, un file ausiliario con le operazioni già predisposte ma non implementate, e un main.

***Nota 1**: per implementare in modo efficiente una coda usando un array dinamico sarebbe necessario gestire l'array in maniera circolare, ossia incrementando/decrementando i due estremi della coda mediante operazioni aritmetiche modulo M (dove M e' la dimensione dell'array -- da non confondere con il numero di elementi presenti in coda, che è <= M). Per semplicità ciò non è richiesto; tuttavia è doveroso precisare che, senza la gestione circolare, l'uso dell'array per realizzare una coda è quantomai inefficiente e perfino un po' complicato (perché costringe a spostare tutti gli elementi rimanenti ogni volta che se ne estrae uno).*

***Nota 2**: l'estrazione di elementi dalla coda fa aumentare lo spazio inutilizzato nell'array, dunque può essere opportuno restringere l'array dinamico allo scopo di risparmiare memoria. Tuttavia ciò ha un costo computazionale, e dunque non deve essere fatto ad ogni singola estrazione di un elemento bensì solamente quando lo spazio inutilizzato supera una certa soglia. Se l'array dinamico lavora a blocchi di dimensione BLOCKDIM, un possibile criterio per avviare l'operazione di restringimento è quando lo spazio inutilizzato diventa >= BLOCKDIM.*

Per condurre la fase di test vengono forniti alcuni file di input. Le funzioni di I/O sono già implementate per entrambi i TDD. L'unico file che potete modificare è il file ausiliario, che potrete ridenominare opportunamente, al quale potete aggiungere tutte le funzioni ausiliarie (locali al file stesso, non visibili dal main) che riterrete necessarie. **La struttura dati con cui si implementa il TDD deve essere necessariamente quella indicata nel testo. Il main e l'header non devono essere modificati** (o meglio, potete modificarli a scopo di testing, ma il vostro codice deve funzionare con main e header che vi abbiamo fornito noi e che andranno consegnati nel file .zip).

Header file per il TDD stack

```
#include <cstddef> // serve per il NULL
#include <iostream>
#include <stdexcept>
#include <vector>
#include <fstream>

using namespace std;

// Implementa STACK con strutture collegate semplicemente mediante puntatori e
tipo base Elem dotato di ordinamento
namespace stack {

typedef int Elem;
struct cell;
typedef cell *Stack;
const Stack EMPTYSTACK = NULL;
const int FINEINPUT = -1000000;
const int EMPTYELEM = -1000000;

bool isEmpty(const Stack&);
void push(const Elem, Stack&);    /* aggiunge elem in cima (operazione safe, si
può sempre fare) */
Elem pop(Stack&);    /* toglie dallo stack l'ultimo elemento e lo restituisce;
se lo stack è vuoto la condizione è di errore. Lo segnaliamo restituendo
EMPTYELEM (potremmo in alternativa sollevare un'eccezione) */
Elem top(Stack&);    /* restituisce l'ultimo elemento dello stack senza
toglierlo; se lo stack è vuoto la condizione è di errore. Lo segnaliamo
restituendo EMPTYELEM (potremmo in alternativa sollevare un'eccezione) */
}


stack::Stack readFromFile(string);
stack::Stack readFromStdin();
stack::Stack readFromStream(istream&);
void print(const stack::Stack&);
```

Header file per il TDD queue

```
#include <iostream>
#include <stdexcept>
#include <vector>
#include <fstream>

using namespace std;

// Implementa QUEUE con array dinamico e tipo base Elem dotato di ordinamento

namespace queue {

const int BLOCKDIM = 1000;
typedef int Elem;      // tipo base
struct queue{
    Elem* queue; // array dinamico
    int size;
    int maxsize;
};
typedef queue Queue;

const int FINEINPUT = -1000000;
const int EMPTYELEM = -1000000;
```

```
Queue createEmpty();          /* restituisce la coda vuota */
bool isEmpty(const Queue&);
void enqueue(Elem, Queue&);   /* inserisce l'elemento "da una parte" della coda
*/
Elem dequeue(Queue&);         /* cancella l'elemento (se esiste) "dall'altra
parte della coda" e lo restituisce; se la coda è vuota la condizione è di
errore. Lo segnaliamo restituendo EMPTYELEM (potremmo in alternativa sollevare
un'eccezione) */
Elem first(Queue&);           /* restituisce l'elemento in prima posizione (se
esiste) senza cancellarlo; se la coda è vuota la condizione è di errore. Lo
segnaliamo restituendo EMPTYELEM (potremmo in alternativa sollevare
un'eccezione) */
}


queue::Queue readFromFile(string);
queue::Queue readFromStdin();
queue::Queue readFromStream(istream&);
void print(const queue::Queue&);
```

E' importante, durante la fase di implementazione, seguire tutti i consigli che vi sono stati spiegati e che sono contenuti nelle slide "*Sarebbe bello che ...*" della lezione del 01-3-2021. In particolare, è importante compilare spesso il codice (meglio compilare una volta in più che una in meno ...) e iniziare la codifica delle funzioni che possono essere testate, seguendo un ordine che vi permetta sempre di verificare la correttezza di quello che state implementando (ad esempio è inutile implementare la funzione size() che restituisce la dimensione di una lista se prima non avete implementato la createEmpty() e la Insert()). Durante la preparazione della traccia i docenti hanno cercato di seguire i consigli di "buona programmazione": come vedete, il codice che vi è stato fornito è stato annotato con commenti utili (in particolare quelli che spiegano la semantica delle operazioni da implementare) ed è stato indentato[1] in modo che sia facilmente comprensibile. In linux è possibile anche utilizzare il comando *indent*[2] che effettua l'indentazione di un programma C/C++ in automatico. Il suo utilizzo è molto semplice: il comando **indent -linux sorgente.cpp indenta il codice contenuto nel file sorgente.cpp (attenzione però che per poter applicare questo comando è necessario che il file sia corretto sintatticamente).**

E' richiesto inoltre di testare in maniera approfondita il codice prodotto cercando di esercitare tutte le funzionalità offerte dal main. Per questo vi potranno essere utili i file di input forniti.

[1] https://it.wikipedia.org/wiki/Stile_d%27indentazione
[2] https://linux.die.net/man/1/indent