LAB3

STACK

- **CreateEmpty:** è progettata per creare e restituire uno stack vuoto. Ecco una spiegazione dettagliata delle operazioni svolte dalla funzione:

    1. **Stack sret;**: Viene dichiarata una variabile di tipo **Stack** chiamata **sret**. Questo è lo stack che verrà restituito alla fine della funzione.
    2. **sret.size = 0;**: L'attributo **size** della struttura **Stack** viene impostato a 0. Questo indica che lo stack appena creato è vuoto, in quanto non contiene nessun elemento.
    3. **sret.maxsize = BLOCKDIM;**: L'attributo **maxsize** della struttura **Stack** viene impostato a **BLOCKDIM**. Questo è il numero massimo di elementi che lo stack può contenere inizialmente.
    4. **sret.data = new Elem[sret.maxsize];**: Viene allocato dinamicamente un array di elementi di tipo **Elem** (che è un sinonimo per **int** in questo caso) con dimensione **sret.maxsize**. Questo array è utilizzato per memorizzare gli elementi dello stack. L'indirizzo di memoria dell'array appena allocato viene assegnato all'attributo **data** della struttura **Stack**.
    5. **return sret;**: Infine, la variabile **sret**, che rappresenta lo stack vuoto appena creato, viene restituita dalla funzione.

    In sintesi, *la funzione **stack::createEmpty()** crea un nuovo stack vuoto con una dimensione massima iniziale specificata da **BLOCKDIM**, allocando dinamicamente un array di dimensione **BLOCKDIM** per memorizzare gli elementi dello stack e inizializzando gli altri attributi della struttura **Stack** di conseguenza*.

- **isEmpty:** Semplicemente controlla se lo stack è vuoto, per farlo uso un return st.size == 0

- **void stack::push:** aggiunge elem in cima.

1. **if(st.size == st.maxsize) {**: Inanzitutto cotrollo se lo stack ha raggiunto la sua capacità massima (maxsize) e quindi ha bisogno di essere ridimensionato. Nel caso servisse un ridimensionamento per prima cosa creo una variabile NewDimensione che appunto mi rappresenta la nuova dimensione dell'array dinamico. Sommo BLOCKDIM (lunghezza dei blocchi da aggiungere) e st.maxsize (dimensione massima attuale dello stack.)
   unsigned int NewDimensione = st.maxsize + BLOCKDIM;: Calcola la nuova dimensione dell'array dinamico, che è ottenuta aggiungendo **BLOCKDIM** alla dimensione massima attuale dello stack.

2. **Elem *NuovoArray = new Elem[NewDimensione];**: Alloca dinamicamente un nuovo array di elementi di tipo **Elem** con la nuova dimensione calcolata.
3. **for (unsigned int i = 0; i < st.size; ++i) { NuovoArray[i] = st.data[i]; }**: Copia gli elementi dall'array originale **st.data** al nuovo array (**NuovoArray**)
4. **delete[] st.data;**: Libera la memoria occupata dall'array originale **st.data** (In poche parole elimino l'array precedente).
5. **st.data = NuovoArray; st.maxsize = NewDimensione;**: Aggiorna il puntatore **data** dello stack per puntare al nuovo array appena creato e aggiorna la dimensione massima dello stack (**maxsize**) con la nuova dimensione calcolata.
6. **st.size += 1;**: Incrementa la dimensione dello stack per indicare che è stato aggiunto un nuovo elemento.
7. **st.data[st.size - 1] = el**: Aggiunge l'elemento **el** alla cima dello stack nell'ultima posizione dell'array.

- **stack::pop:** Restituisce l'elemento in cima allo stack e decrementa la dimensione dello stack.
  Eseguo **–st.size** per ridurre la size dello stack, quindi "elimino" l'ultimo elemento. Infine faccio **return st.data[st.size]** in quanto è come se negli array statici scrivessi return A[N];

- **stack::top** : restituisce l'ultimo elemento dello stack senza toglierlo.
  Per accedere all'ultimo elemento dello stack, si utilizza **st.data[st.size - 1]**, poiché l'indice dell'ultimo elemento in un array inizia da 0 e la dimensione dello stack (**st.size**) rappresenta il numero di elementi nello stack. Quindi **st.size - 1** restituisce l'indice dell'ultimo elemento.

QUEUE

- **CreateEmpty:** creo e restituisco una coda vuota.
  Dichiaro una variabile di tipo Queue chiamata q che rappresenterà la coda vuota da restituire.
  q.li e **q.end** entrambi li inizializzo a **EMPTYLIST** che è un puntatore speciale che indica una lista vuota.
- **IsEmpty:** Questa funzione controlla se la coda è vuota. Basta fare un return q.li == EMPTYLIST (Sarebbe l'equivalente di scrivere l == nullptr nelle liste normali)

- **Enqueue:** inserisce l'elemento "da una parte" della coda (inserimento in testa).
    1. **list aux = new cell;**: Viene allocata dinamicamente una nuova cella **aux** per memorizzare l'elemento da inserire nella coda.
    2. **aux->el = e;**: Viene assegnato l'elemento **e** alla variabile **el** della cella **aux**.
    3. **aux->next = EMPTYLIST;**: Viene impostato il puntatore **next** della cella **aux** a **EMPTYLIST** per indicare che questa è l'ultima cella della coda.
    4. **if (q.li == EMPTYLIST) { ... }**: Controlla se la coda è vuota.
    5. Se la coda è vuota, la nuova cella **aux** diventa sia il primo che l'ultimo elemento della coda (**q.li** e **q.end** puntano entrambi a **aux**).
    6. Inoltre, essendo l'unica cella nella coda, il suo puntatore **prev** viene impostato a **nullptr**.
    7. **aux->next = q.li;**: Il puntatore **next** di **aux** viene impostato per puntare all'elemento che era precedentemente in testa alla coda.
    8. **aux->prev = EMPTYLIST;**: Il puntatore **prev** di **aux** viene impostato a **EMPTYLIST**, poiché **aux** diventa il nuovo primo elemento della coda.
    9. **aux->next->prev = aux;**: Il puntatore **prev** dell'elemento successivo al nuovo primo elemento (cioè l'elemento che era in testa prima dell'inserimento di **aux**) viene aggiornato per puntare a **aux**, in modo che punti alla nuova testa della coda.
    10. Infine, **q.li** viene aggiornato per puntare a **aux**, rendendolo il nuovo primo elemento della coda.

- **Dequeue:** cancella l'elemento (se esiste) "dall'altra parte" della coda (elimino ultimo elemento, quindi appunto elimino in coda).

1. **if (q.li == nullptr) { ... }**: Controlla se la coda è vuota. Se la coda è vuota (cioè **q.li** è **nullptr**), viene sollevata un'eccezione con un messaggio di errore.
2. **Elem ret = q.end->el;**: Memorizza il valore dell'elemento in coda (**q.end->el**) nella variabile **ret**. Questo valore verrà restituito alla fine della funzione.
3. **if (q.li == q.end) { ... }**: Verifica se la coda contiene un solo elemento. Se **q.li** è uguale a **q.end**, significa che la coda ha un solo elemento.
   Se la coda ha un solo elemento, significa che stiamo rimuovendo l'ultimo elemento della coda. In tal caso, viene deallocata la memoria dell'elemento (**q.li**) e i puntatori **q.li** e **q.end** vengono entrambi impostati su **EMPTYLIST**, indicando così che la coda è vuota.

   Se la coda ha più di un elemento, procediamo come segue:

4. **cell* aux = q.end->prev;**: Memorizziamo il puntatore all'elemento precedente all'ultimo elemento (**q.end**) nella variabile **aux**.
5. **aux->next = EMPTYLIST;**: Impostiamo il puntatore **next** dell'elemento precedente (**aux**) a **EMPTYLIST**, indicando che l'elemento precedente diventa l'ultimo elemento della coda.
6. **delete q.end;**: Deallochiamo la memoria dell'elemento in coda.
7. **q.end = aux;**: Aggiorniamo il puntatore **q.end** per puntare all'elemento precedente, rendendolo così il nuovo ultimo elemento della coda.
8. **Return ret:** Restituiamo il valore memorizzato nella variabile **ret**, che rappresenta l'elemento rimosso dalla coda.