UNIVERSITÀ DI PARMA

EXTRA
Queues & Stacks

An Englishman, even if he is alone, forms an orderly queue of one.

George Mikes

Sommario

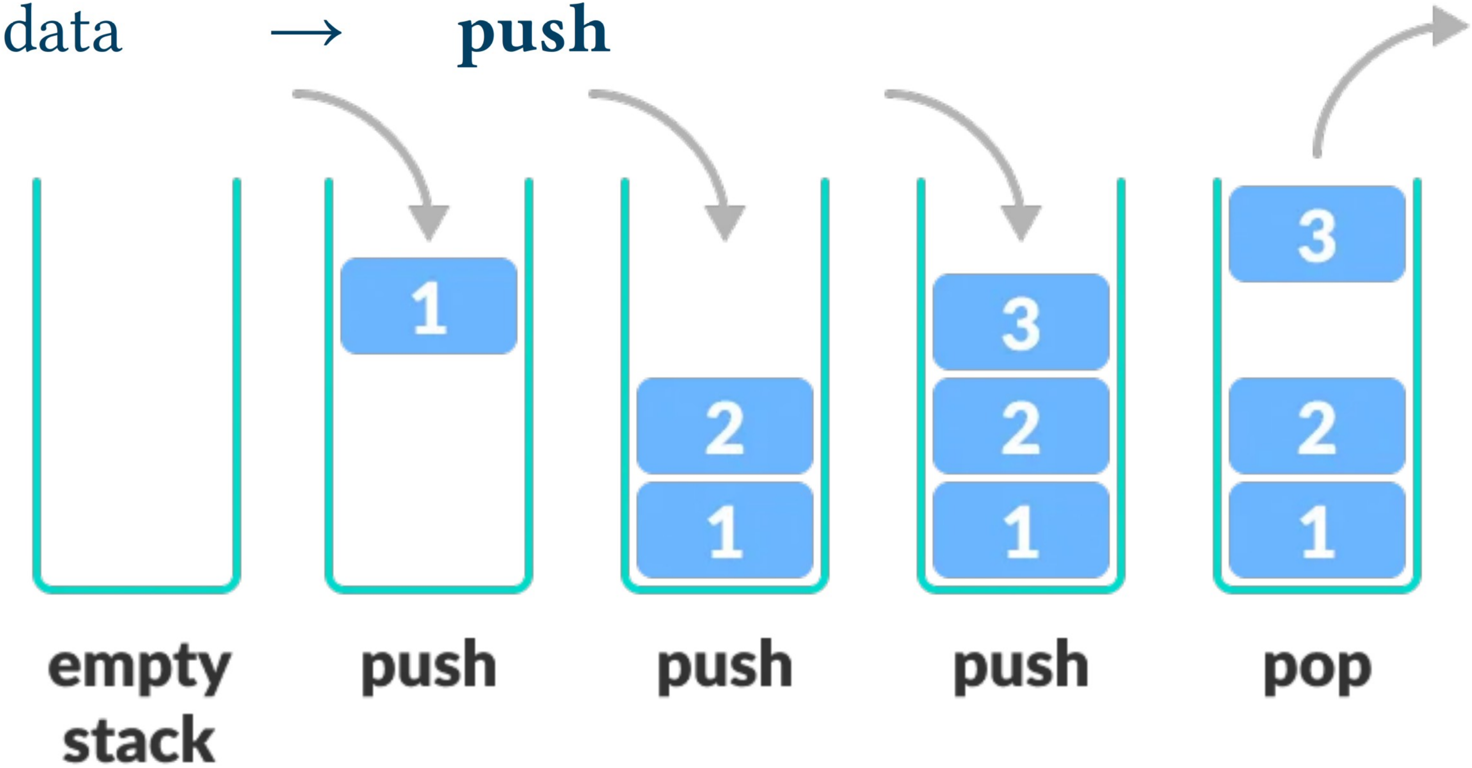
- Stack structure
  - LIFO policy
  - Main operations
  - Example
- Queue structure
  - FIFO policy
  - Main operations
  - Example

What is a Stack?

- Specific data structure
- Real world example, stack of chairs
- When I need a chair?
  - I have to get the top one
- When I want to put a chair?
  - On the top
- **LIFO → Last In First Out**

Main stack operations

- Get data → **pop**
- Put data → **push**



C & Stack structures

- Several High Level programming languages natively supports stacks
  - i.e. C++
- But not C…
- We need to create our own primitives set

C & Stack structures

- What we can use to manage stacks in C?
- Potentially, large set of data
  - Array!
- In the following
  - Example of a stack for floating point numbers

Stack of floating point numbers

- Based on array
- A sufficiently large array
- Example:

```
double stack[1000];
```

Stack of floating point numbers

- Is the array alone enough?
  - Of course, it allows to store data
  - Does it allow to understand where to put data?
  - Does it allow to understand from where we have to take data?
  - Does it allow to understand whether the array is empy or full?
- No...

Stack of floating point numbers

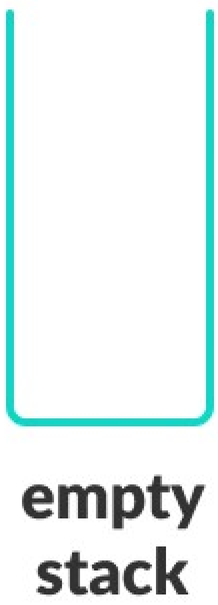
- We also need an additional info
  - The top of the stack → index where we store/get data
- We can put beside the array an index
  - Write position

```
double stack[1000];
int stack_index = -1; // -1 means ‘empty’
```

Stack of floating point numbers: creation

- We can then merely create a stack defining:
  - A sufficiently large array
  - An index to set the stack top

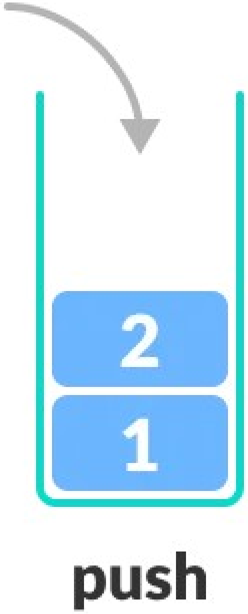
```
double stack[1000];
int stack_index = -1;
```



Stack of floating point numbers: PUSH

- When we need to add data (i.e. PUSH)
  - Put them on the index position
    - If there is space…
  - Increment index for the stack top

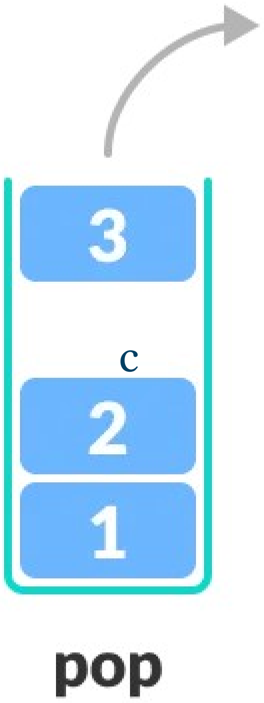
```
if(stack_index<1000) // there is space?
   stack[stack_index++] = <value to be inserted>;
else
   <ERROR>
```



Stack of floating point numbers: POP

- When we need to get data (i.e. POP)
  - Get them exploiting the index position
    - If there are data
  - Caveat: stack top index actually encodes the “write” position
  - Decrement index for the stack top

```
if(stack_index >= 0) // do we have data?
  <result> = stack[--stack_index];
else
  <ERROR>
```



Stack of floating point numbers: improvements

UNIVERSITÀ DI PARMA

- Problem 1: array and index are independent data
  - We need to strictly manage them toghether
  - Not confortable to define multiple stacks
- Problem 2: redundant code
  - When we need multiple push/pop operations
  - We need to “replicate” code
  - Do you remember?
    - Duplicated code is **evil**

Stack of floating point numbers: struct

- We can "join" the two stack components using a `struct`:

```
struct stack
  {
    double dati[1000];
    int top;
  };
```

Stack of floating point numbers: struct

- Creating a struct based stacks can be then

```
struct stack mystack;
mystack.top = -1;
```

- It is also easier to create additional stacks

```
struct stack other_stack;
other_stack.top = -1;
```

Stack of floating point numbers: primitives

- Pop & Push operations can be easily implemented using functions

```
void push(struct stack *s, double n)
{
  if(s->top == (1000-1)) <ERRORE>
  s->dati[++s->top] = n;
}
```

Stack of floating point numbers: primitives

- Pop & Push operations can be easily implemented using functions

```
double pop(struct stack *s)
{
   if(s->top == 0) <ERRORE> //vuoto
   return s->dati[s->top--];
}
```

Stack of floating point numbers: primitives

- Pop & Push operations use is something like:

```
push(&miostack, 3.14);
...
push(&miostack, variabile_double);
...
double a=pop(&miostack);
```

What is a queue?

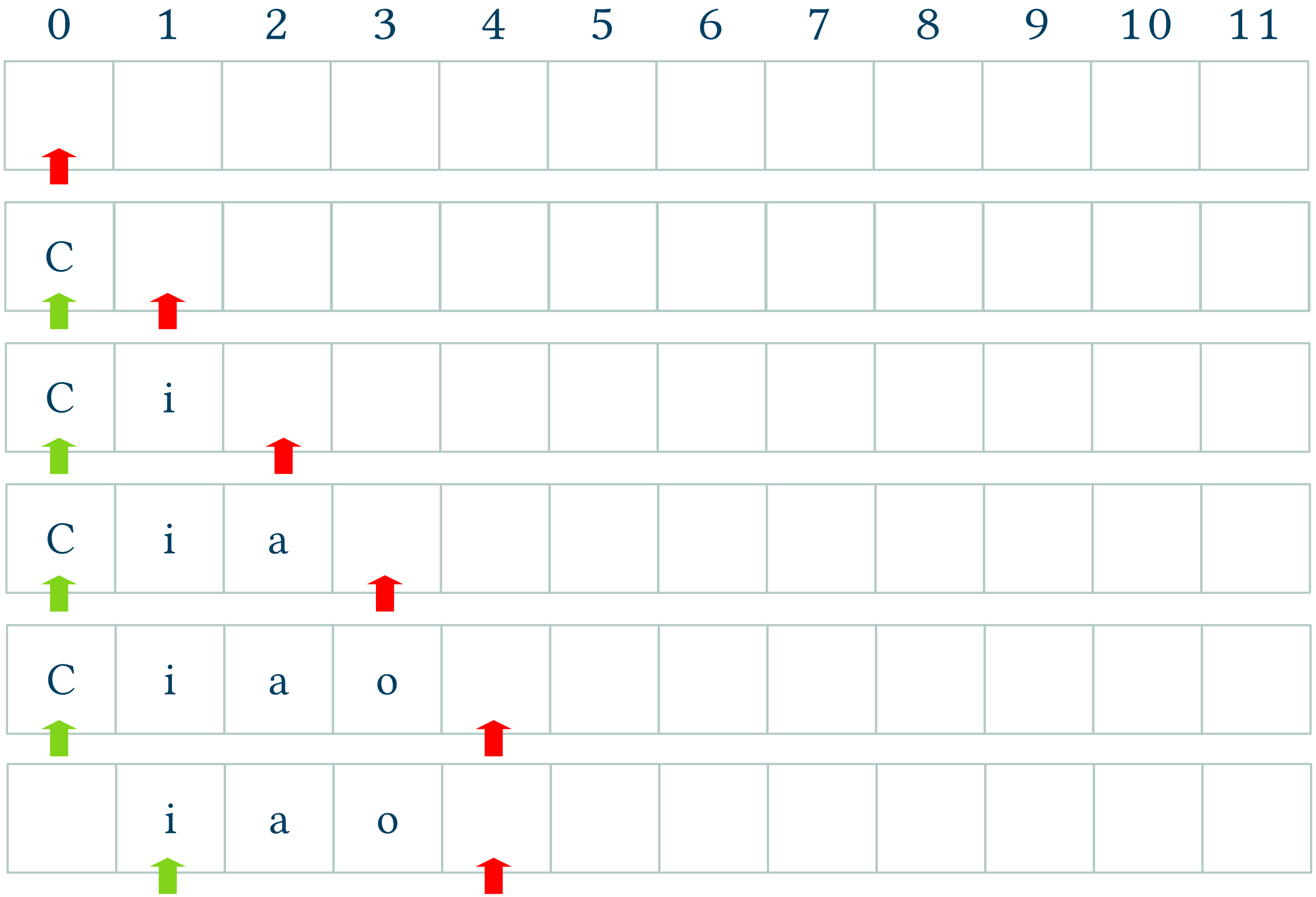
- Specific data structure
- Real world example, people moving on a e
- Who is the first to exit?
  - The first that entered the escalator
- Where I have to enter the escalator?
  - From the starting
- **FIFO → First In First Out**

C & queues

UNIVERSITÀ DI PARMA

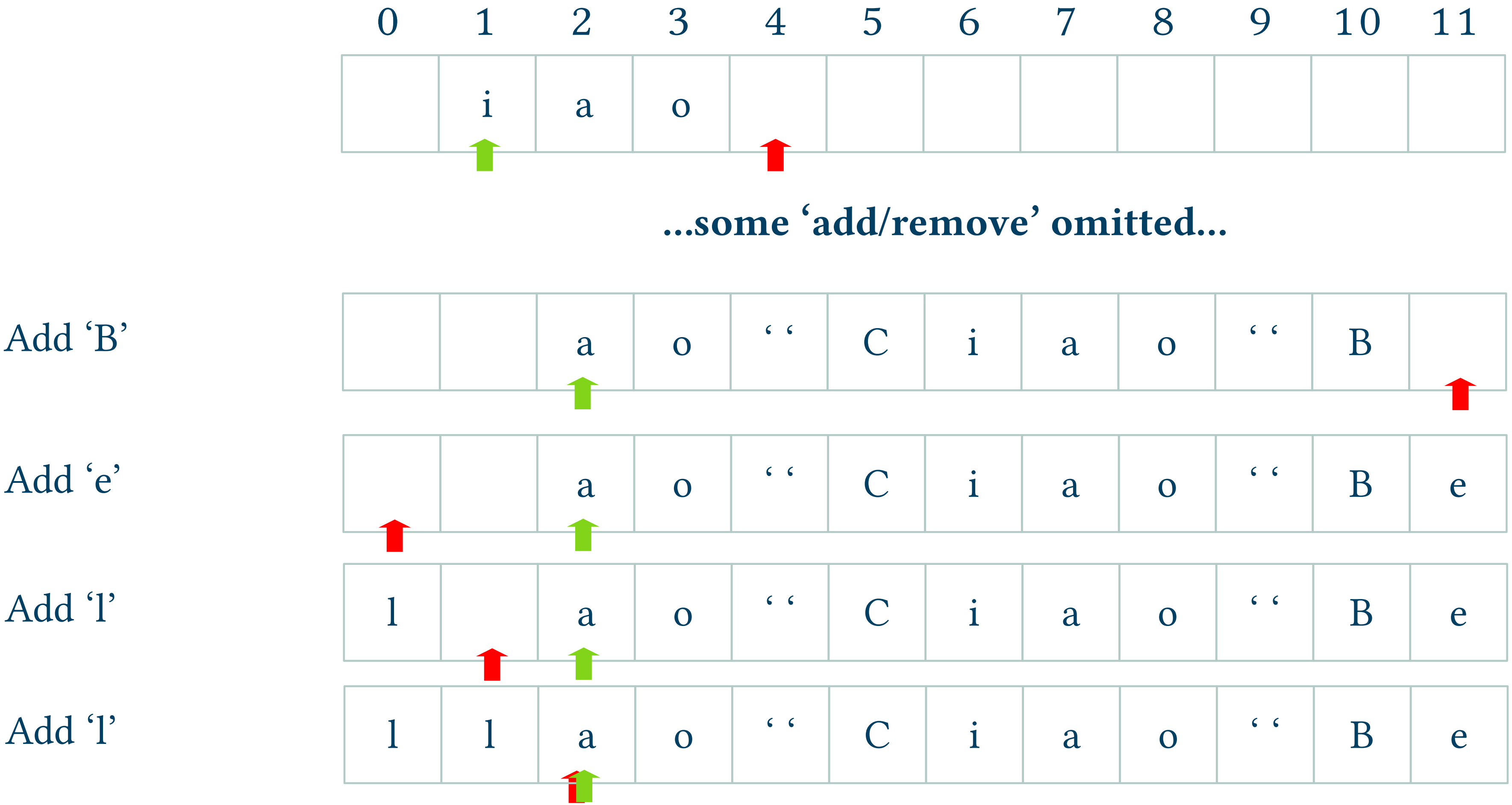
- Not natively supported
- Again, an array can be a solution
  - Lists are usually better
- Differently from stacks we have separate reading and writing points:
  - We add at one end
  - We remove from other end

Queues & arrays



	0	1	2	3	4	5	6	7	8	9	10	11
Empty Queue												
Add ‘C’	C											
Add ‘i’	C	i										
Add ‘a’	C	i	a									
Add ‘o’	C	i	a	o								
Remove		i	a	o								

Queues & arrays



	0	1	2	3	4	5	6	7	8	9	10	11
		i	a	o								

…some ‘add/remove’ omitted…

	0	1	2	3	4	5	6	7	8	9	10	11
Add ‘B’			a	o	‘ ‘	C	i	a	o	‘ ‘	B	
Add ‘e’			a	o	‘ ‘	C	i	a	o	‘ ‘	B	e
Add ‘l’	l		a	o	‘ ‘	C	i	a	o	‘ ‘	B	e
Add ‘l’	l	l	a	o	‘ ‘	C	i	a	o	‘ ‘	B	e

Queue of floating point numbers

- Again we can use a struct

```
struct queue
  {
    double data[1000];
    int front, rear;
  };
```

Queue of floating point numbers

- When creating a new queue we can use something like:

```
struct queue myqueue;
myqueue.front = -1;
myqueue.rear  = -1;
```

Queue of floating point numbers

- Why -1s?
- We need to differentiate from 2 states
- Empty queue
  - front = rear = -1
- Full queue
  - front = rear ≠ -1

Queue of floating point numbers: insert

```
void queue_insert(struct queue *q, double n){
  if(q->rear == q->front && q->front != -1)
      <ERRORE> // full queue
  if(q->front == -1){
    q->front = 0;
    q->rear = 0;
  }
  q->data[q->rear++] = n;
  q->rear = q->rear % 1000; // manage overflow
}
```

Queue of floating point numbers: remove

```
double queue_remove(struct queue *q){
  if(q->rear == -1 && q->front != -1)
      <ERRORE> // empty queue
  double val = q->data[q->front++];
  q->front = q->front % 1000; // manage overfloW
  if(q->front == q->rear){ // empty queue
    q->front = -1;
    q->rear = -1;
    }
  return val;
}
```

UNIVERSITÀ DI PARMA

EXTRA
Queues & Stacks

> *An Englishman, even if he is alone, forms an orderly queue of one.*
>
> George Mikes