Code e Pile

Linguaggio C++

Code

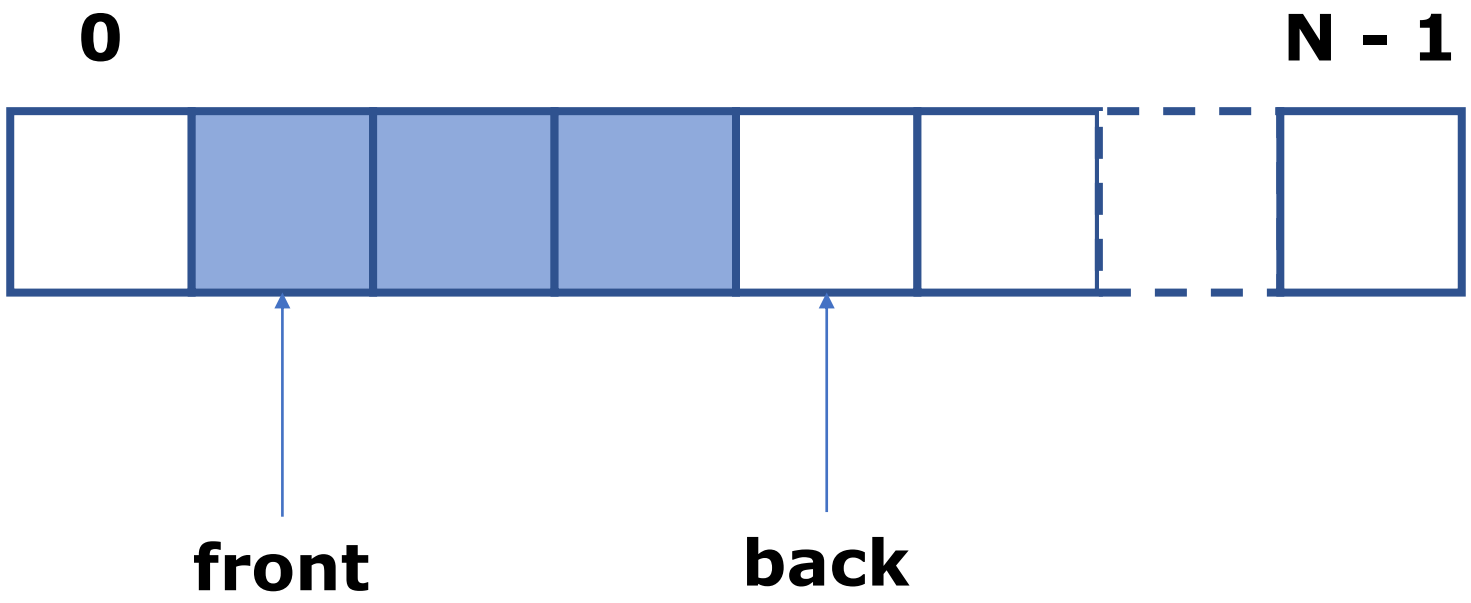
Code

- **DEFINIZIONE**. Un insieme ordinato di dati di ugual tipo in cui è possibile effettuare operazioni di inserimento ed estrazione secondo la seguente regola di accesso: il primo dato inserito è anche il primo ad essere estratto (regola FIFO: *First In First Out*)

Implementazione mediante array

- Array circolare con due indici *back* e *front*
  - Array circolare: l'ultimo elemento è visto come adiacente al primo
- L'indice *back* individua la posizione del prossimo inserimento (posizione vuota)
- L'indice *front* individua la posizione della prossima estrazione (posizione piena)
- Dopo ogni operazione l'indice viene incrementato in modo circolare
- La coda è vuota quando *back == front*
- La coda è piena quando un altro inserimento porterebbe porterebbe *back* ad essere uguale a *front*

Implementazione mediante array



Implementazione mediante array statici

```
const int N = 10;              // capacità della coda
struct Coda {
        int front, back;       // indice della testa e della coda
        int queue[N];          // elementi della coda
};

inline void inizializza(Coda& q) {q.back = q.front = 0;}
inline void enque(Coda& q, int e) {
        q.queue[back] = e; back = (back + 1) % N;
}
inline void deque(Coda& q, int &e) {
        e = q.queue[front]; front = (front + 1) % N;
}
inline int vuoto(Coda q) { return q.back == q.front; }
inline int pieno(Coda q) { return q.front == (q.back + 1)%N; }
```

Implementazione mediante array dinamici

```
struct Coda {
        int front, back;        // indice della testa e della coda
        int size;               // capacità della coda
        int *queue;             // puntatore agli elementi della coda
};

inline void crea(Coda& q, int n) {
        queue = new int[size = n]; q.back = q.front = 0;}
inline void enque(Coda& q, int e) {
        q.queue[back] = e; back = (back + 1) % size; }
inline void deque(Coda& q, int &e) {
        e = q.queue[front]; front = (front + 1) % size; }
inline int vuoto(Coda q) { return q.back == q.front; }
inline int pieno(Coda q) { return q.front == (q.back + 1)%size; }
inline void distruggi(Coda& q) {delete [] queue;}
```

Pile

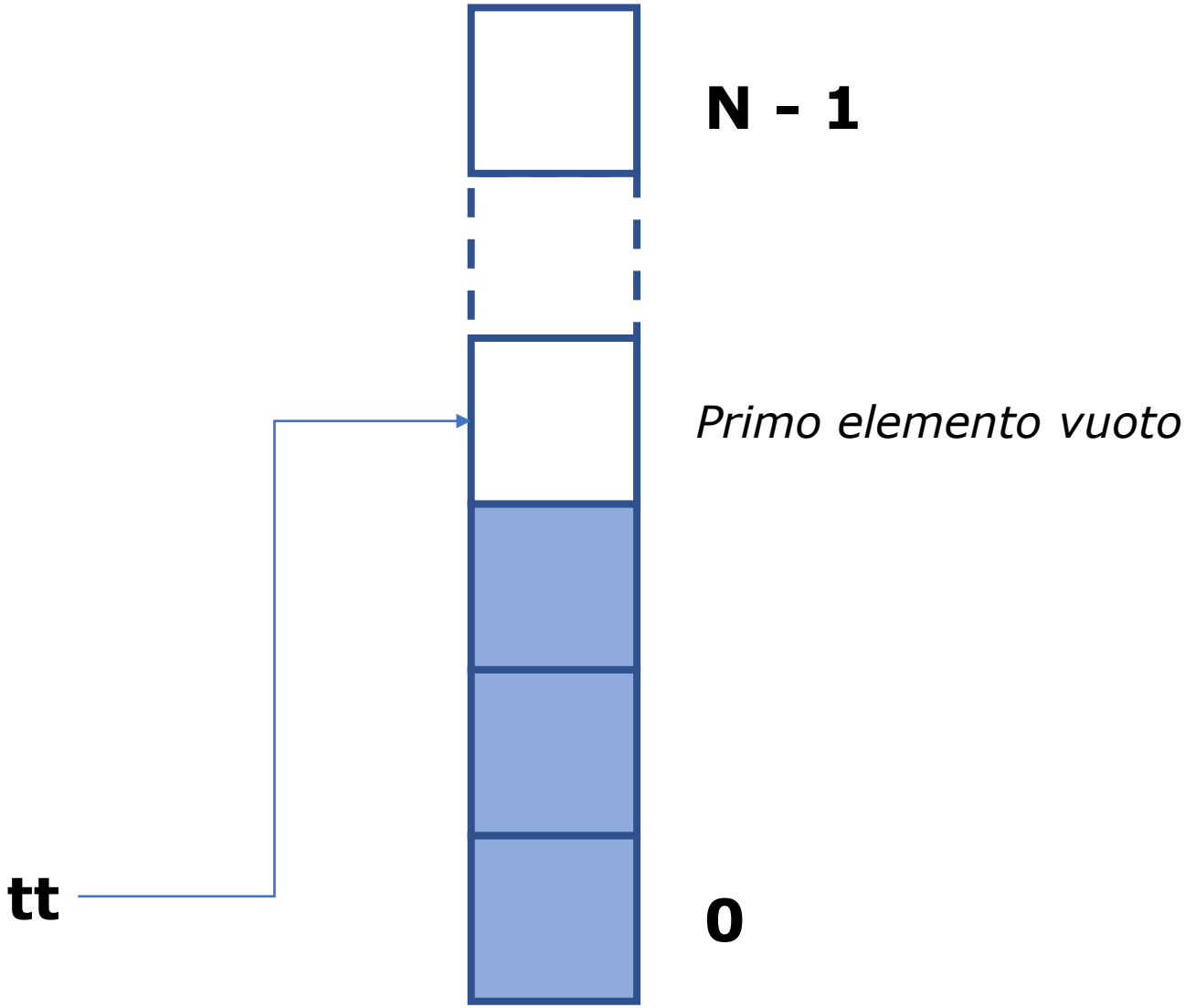
Pile

- **DEFINIZIONE**. Un insieme ordinato di dati di ugual tipo in cui è possibile effettuare operazioni di inserimento ed estrazione secondo la seguente regola di accesso: l'ultimo dato inserito è il primo ad essere estratto (regola LIFO: *Last In First Out*)

Implementazione mediante array

- L’indice *tt* individua la prima posizione vuota
- La pila è vuota quando *tt == 0*
- La pila è piena *tt* == *N*

Implementazione mediante array



Implementazione mediante array statici

```
const int N = 10;    // capacità della pila
struct Pila {
        int tt;      // indice del primo elemento vuoto
        int ee[N];   // elementi della pila
};
inline void inizializza(Pila& s) { s.tt = 0; }
inline void push(Pila& s, int v) { s.ee[s.tt++] = v; }
inline int pop(Pila& s) { return s.ee[--s.tt]; }
inline int top(Pila s) { return s.ee[s.tt - 1]; }
inline int vuoto(Pila s) { return s.tt <= 0; }
inline int pieno(Pila s) { return s.tt >= N; }
```

Implementazione mediante array dinamici

```
struct Pila {
        int tt;         // indice del primo elemento vuoto
        int size;       // capacità della pila
        int *ee;        // elementi della pila
};

inline void crea(Pila& s, int n) { s.ee = new int[size = n]; s.tt = 0; }
inline void push(Pila& s, int v) { s.ee[s.tt++] = v; }
inline int pop(Pila& s) { return s.ee[--s.tt]; }
inline int top(Pila s) { return s.ee[s.tt - 1]; }
inline int vuoto(Pila s) { return s.tt <= 0; }
inline int pieno(Pila s) { return s.tt >= size; }
inline void distruggi(Pila& s) { delete [] ee;  }
```