

# Relazione progetto Architettura degli elaboratori

Giovanni Palmieri (7006086)

24 giugno 2021

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Descrizione del ciclo principale</b>	<b>2</b>
2.1	Riconoscimento input . . . . .	2
2.2	Chiamata delle operazioni . . . . .	3
<b>3</b>	<b>Implementazione Sort</b>	<b>4</b>
3.1	Funzione getMinAddress . . . . .	4
3.2	Funzione sort . . . . .	4
3.3	Criteri di ordinamento . . . . .	5

## 1 Introduzione

Introduzione

## 2 Descrizione del ciclo principale

Il compito del ciclo principale *mainLoop* è quello di leggere l'array di input *listInput* e in base ad esso chiamare le funzioni relative ai comandi

### 2.1 Riconoscimento input

Un comando viene riconosciuto semplicemente dalla prima lettera, dato che non esistono comandi che abbiano la stessa lettera iniziale.

Carattere	Significato
A	ADD{...}
D	DEL{...}
P	PRINT
S	SORT
R	REV

Tabella 1: Riconoscimento comandi

Bisogna però considerare il caso in cui, una lettera relativa ad un comando sia in realtà il parametro di un comando, esempio:

$\textcircled{A}\text{DD}\{\textcircled{A}\}$

Per distinguere questi due casi ci avvaliamo del fatto che:

1. Il primo carattere che troviamo, relativo ad un comando, non può essere il parametro di un comando.
2. Per ogni comando sappiamo quanto è lungo, dato che non sono ammessi spazi all'interno dei comandi.

Perciò ogni volta che leggiamo un comando, spostiamo il puntatore alla lista di input alla fine del comando.

Comando	Valore da sommare
ADD	6 + 1
DEL	6 + 1
PRINT	5 + 1
SORT	4 + 1
REV	3 + 1

Tabella 2: Tabella valori da sommare per evitare letture incorrette

Il "+1" in Tabella2 ci permette di risparmiare la validazione di un char, dato che i comandi sono divisi da almeno un "~" e eventuali spazi.

## 2.2 Chiamata delle operazioni

Ogni volta che viene riconosciuto un comando, viene eseguita una jump al blocco di codice che gestisce la chiamata al relativo comando.

(*callAdd*, *callDel*, *callSort*, *callRev*, *callPrint*)

Prima di ogni chiamata vengono salvati i registri nello stack, e in caso di comandi con un parametro (*ADD*, *DEL*), il parametro viene messo in **a0**. Poi viene incrementato il puntatore alla stringa di input secondo la Tabella 2

### 3 Implementazione Sort

**Descrizione** L'algoritmo utilizzato è un selection sort. L'algoritmo scorre tutta la lista, e ad ogni elemento lo sostituisce con l'elemento più piccolo, tra quelli successivi.

**Implementazione** L'algoritmo è composto principalmente da due funzioni.

1. La funzione che trova il minimo tra i nodi successivi: *getMinAddress*
2. La funzione che per ogni nodo lo scambia con il risultato di *getMinAddress*: *sort*

Entrambe le funzioni sono implementate ricorsivamente.

#### 3.1 Funzione *getMinAddress*

La funzione *getMinAddress* ha due parametri, **a0** è l'indirizzo del nodo corrente, **a1** è l'indirizzo del nodo con il valore minimo. La funzione confronta i valori dei due nodi, se il nodo corrente è minore del minimo allora il nodo minimo diventa il nodo corrente. Dopo di che, controlla se il nodo corrente ha un nodo successivo, in caso non ci sia la funzione ritorna. Mentre se il nodo successivo è presente la funzione si richiama sostituendo al nodo corrente il nodo successivo.

```
private Node getMinAddress(Node current , Node min){
    if(current.getValue() < min.getValue()){
        min = current;
    }
    if(current.getNext() != null){
        min = getMinAddress(current.getNext() , min);
    }
}
```

Listing 1: Codice java algoritmo *getMinAddress*

#### 3.2 Funzione *sort*

La funzione *sort* ha un solo parametro, **a0** l'indirizzo nodo da sostituire con il minimo. La funzione chiama *getMinAddress* sul nodo corrente, e lo scambia con il risultato della funzione. Dopo di che, controlla che ci sia il nodo

successivo e in caso sia presente richiama la funzione *sort* sul prossimo nodo.

```
private void sort(Node current){  
    Node min = getMinAddress(current , current);  
    swap(current , min);  
    if(current.getNext() != null){  
        sort(current.getNext());  
    }  
}
```

Listing 2: Codice java algoritmo sort

### 3.3 Criteri di ordinamento

**Descrizione** Il programma non ordina i nodi in base al valore ASCII, ma ha un ordinamento specifico:

- Lettere Maiuscole ASCII(da 65 a 90 compresi)
- Lettere Minuscole ASCII(da 97 a 122 compresi)
- Numeri ASCII(da 48 a 57 compresi)
- Tutti gli altri simboli

All'interno dei gruppi gli elementi sono ordinati in base al loro valore ASCII.

**Implementazione** Per fare questo ordinamento associamo ad ogni valore ASCII un valore di ordinamento, di modo che gli elementi possano essere ordinati in base al loro valore di ordinamento. Abbiamo perciò bisogno di una funzione che associa un numero ad ogni carattere di modo che:

Lettere Maiuscole > Letter Minuscole > Numeri > Resto.

Per fare questa funzione quello che facciamo è sommare un certo valore in base al gruppo di appartenenza di un carattere.