UNIVERSITÀ
DEGLI STUDI
FIRENZE

Progetto di Architettura degli Elaboratori

A.A. 2022/2023

Iacopo Paolucci

Mat. n° 7116426

iacopo.paolucci@edu.unifi.it

Data di consegna : 02/01/2024

Descrizione della soluzione adottata

Indice

TABELLA DELLE VARIABILI

Stato iniziale

s0	Puntatore ad un'area di memoria che conterrà il primo elemento.
s1	Puntatore all'area di memoria che contiene la serie di comandi in input.
s2	Variabile per tenere traccia dei comandi letti. (vedi illustrazione)
a0	Puntatore alla testa della lista (questo potrebbe essere modificato durante l'esecuzione).

Illustrazione funzionamento indice input

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	D	D	(	A	)	~	A	D	D	(	B	)	~	D	E	L	(	A	)

s2 → 13

Leggere l’input e verificarne la correttezza

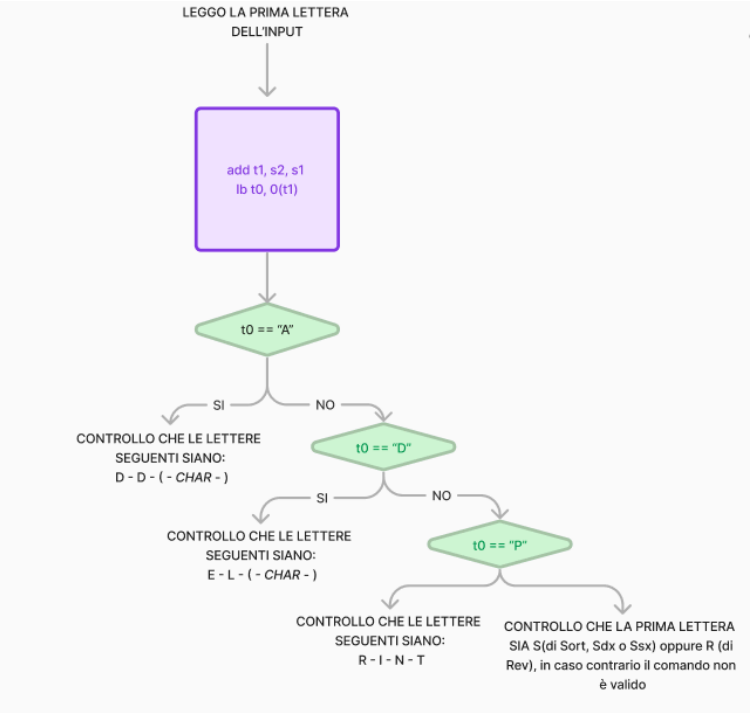
Il primo metodo da realizzare è stato leggere l’input e verificarne la correttezza.
L’implementazione "cerca comando" scorre la lista dell’input, carattere per carattere, verificando se corrisponde ad un comando oppure no.
Se il comando corrisponde ad ADD o DEL verifico che ci sia uno e un solo parametro sennò verifico solamente che ci sia la tilde e che il comando sia corretto.

Come? Descrivo l’implementazione in maniera visiva:
input = “ADD(1) - DEL(1) - PRINT”
Esecuzione
leggo ***carattere*** → *‘A’*
se ***carattere*** *= A → D → D → ADD else not found*
se ***carattere*** *= D → E → L → DEL else not found*
se ***carattere*** *= P → R → I → N → T → PRINT else not found*
se ***carattere*** *= S → O(se carattero != O vado a SDX)→ R → T → SORT else not found*
se ***carattere*** *= D(se carattere != D vado a SSX) → X → SDX else not found*
se ***carattere*** *= S → X → SSX else not found*
se ***carattere*** *= R → E → V → REV else not found*
A - D - D
D - E - L
P - R - I - N - T
S - O - R - T

\ - D - X

\ - S - X

R - E - V



Una volta verificato che il comando sia corretto verifichiamo anche che tra il comando e la tilde ci siano solo spazi, solo a quel punto possiamo dire se un comando è corretto.

ESEMPIO:

"ADD(a) ~ . . . . . " → Comando corretto

"ADD(a) ~ . . . . ." → Comando corretto

"ADD(a) f ~ . . . . . " → Comando errato

"ADD( ) ~ . . . . . ." → Comando errato

ADD

Pseudo codice

```
public void ADD(Nodo head, char parameter){
   var inf = head.getInf();
   if(inf == null){
      head.setInf(parameter).in(memoryStart);
      head.setNext(head).in(memoryStart + 1);
   }else{
      var memoryFree = searchMemory();
      inf = head.getInf();
      var next = head.getNext(),
      if(next == head){
         next.setInf(parameter).in(memoryFree);
         next.setNext(head).in(memoryFree + 1);
      }
      while(next != head){
         next = next.getNext();
      }
      //Qui siamo arrivati in coda
      next.setNext(New Nodo(inf: parameter,next: head).in(memoryFree));
   }
}
```

Implementazione

La add ci permette di creare un nuovo elemento della lista che contiene come informazione DATA=char, e viene aggiunto in coda alla lista esistente.

Una volta richiamata la ADD e verificato che il comando sia corretto abbiamo verifichiamo in quale dei due casi ci troviamo:

1) La lista è vuota: in questo caso inseriamo nella posizione puntata da s1.
2) La lista ha almeno un elemento: in questo caso scorriamo tutti gli elementi fino alla coda, cioè quando l'elemento successivo corrisponde alla testa, ed inseriamo nella prima

posizione libera in memoria, trovata grazie alla label *searchMemory.*

SearchMemory: E' la label dove, a partire dal puntatore alla posizione corrispondente il primo elemento, cioè **s1**, si cerca 5 byte liberi ed una volta trovati si ritorna alla label normal_case1 con il puntatore alla posizione libera.

Esempio

Esadecimale	INF	NEXT
0x01000000	A	0x01000005
0x01000005	B	0x0100000f
0x0100000a		
0x0100000f	C	0x01000000

Andando ad inserire ora un elemento, quest'ultimo verrebbe messo nella posizione *0x0100000a*, questo perché ad ogni inserimento andiamo a ricercare 5 byte liberi a partire dal puntatore alla memoria libera in **s1**

DEL

Pseudo codice

```
public void DEL(Nodo head, char parameter){
   var inf = head.getInf();
   var next = head.getNext();
   if(inf == parameter){
      if(next == head){
         head = null;
      }else{
         var tempHead = head;
         head = head.getNext();
         while(next.getNext() != tempHead){
            next = next.getNext();
         }
         next.setNext(head);
      }
   }
   var tempPrev = head;
   while(next != head){
      inf = next.getInf();
      if(inf == parameter){
         tempNext = next.getNext();
         tempPrev.setNext(tempNext);
         next.setInf(null);
         next.setNext(null);
         if(tempNext == head){
            return;
         }
      }else{
         tempPrev = next;
         next = next.getNext();
      }
   }
   return;
}
```

Implementazione

La funzione DEL ci permette di eliminare uno o più elementi della lista scelto dall'utente.

Durante l'operazione di DEL possiamo lavorare su due casi differenti:

1) L'elemento da eliminare è in testa: Se l'elemento da eliminare è in testa possiamo avere 2 casi distinti:
    a) In caso l'elemento da eliminare sia l'unico elemento della lista, in questo caso possiamo eliminare tutto.
    b) Nel caso in cui ci siano altri elementi nella lista, in questo caso ci basterà far puntare all'elemento in coda l'elemento successivo alla "*testa precedente*", mentre il puntatore alla testa divenne l'elemento puntato dalla "*testa precedente*".
2) L'elemento da eliminare non è in testa, in questo caso basterà salvare il puntatore all'elemento precedente dell'elemento da eliminare e farlo puntare al successivo dell'elemento da eliminare.
   Inoltre l'eliminazione che andiamo a fare è una cancellazione fisica, in modo da poter riutilizzare lo spazio di memoria in caso di una ADD e ottimizzare quindi la memoria utilizzata.

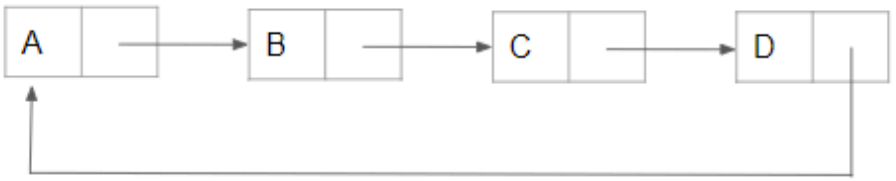
PRINT

Pseudo codice

```
public void PRINT(Nodo head){
   var inf = head.getInf();
   if(inf == null) return;
   print(inf);
   var next = head.getNext();
   while(next != head){
      print(next.getInf());
      next = next.getNext();
   }
   return;
}
```

Implementazione

La funzione PRINT ci permette di scorrere tutta la lista e stamparne di volta in volta il contenuto, fino a quando non arriviamo in coda.



STEP	INF	PUN	OUTPUT
1	A	*B	A
2	B	*C	B
3	C	*D	C
4	D	*A	D

In step 4, PUN == head
quindi stampo ed esco

SSX

Pseudo codice

```
public void SSX(Nodo head){
   var inf = head.getInf();
   var next = head.getNext();
   if(inf == null || next == head){
      return;
   }else{
      head = next;
   }
   return;
}
```

Implementazione

La funzione SSX sposta gli elementi della lista verso sinistra, o in senso antiorario. Quindi, ogni elemento della lista vede il suo indice decrementato di uno, e il primo elemento diventa l'ultimo (per la circolarità della lista)
Quello che facciamo è controllare che la lista abbia almeno due elementi e in caso positivo dire che la testa punta adesso al secondo.

SDX

Pseudo codice

```
public void SDX(Nodo head){
   var inf = head.getInf();
   var next = head.getNext();
   if(inf == null || next == head){
      return;
   }else{
      while(next.getNext() != head){
        next = next.getNext();
      }
      head = next;
   }
   return;
}
```

Implementazione

La funzione SDX sposta gli elementi della lista verso destra, o in senso orario. Quindi, ogni elemento della lista vede il suo indice incrementato di uno, e l'ultimo elemento diventa il primo (per la circolarità della lista).
Quello che facciamo è controllare che la lista abbia almeno due elementi e in caso positivo trovare la coda e dire che questa è la nuova testa della lista.

SORT

Pseudo codice

```
public void SORT(Nodo head, int n){
   if (n == 1) return;
   var count = 0;
   var inf = head.getInf();
   var next = head.getNext();
   if(next == head) return;
   var i = 1;
   var prevNodo = head;

   while(i < n){
      next = next.getNext();
      var nextInf = next.getInf();
      if(inf > nextInf){
         prevNodo.setInf(nextInf);
         next.setInf(inf);
         count++;
         prevNodo = next;
         next = next.getNext();
      }else{
         i++;
         prevNodo = next;
         next = next.getNext();
         inf = nex
      }
   }

   if (count == 0) return;

   bubbleSort(arr, n-1);
}
```

Implementazione

La funzione SORT è stata una delle funzioni più delicate del progetto, essendo questa funzione ricorsiva utilizziamo lo stack per salvarsi l'address di ritorno, questa operazione in linguaggi come Java viene gestita automaticamente.

La scelta di implementare il bubble sort ricorsivo non è una scelta casuale, questo perché lo ritengo uno degli algoritmi più efficienti se applicato ad una lista circolare, inoltre così come per le altre funzioni del progetto, ho deciso di cercare di ottimizzare la complessità computazionale. Infatti l'algoritmo che ho implementato è il bubble sort con "sentinella", la sentinella non è altro che una variabile che ci consente di terminare il sort qualora non avvenissero scambi (significherebbe quindi che la lista sarebbe già ordinata).

REV

Pseudo codice

```
public void REV(Nodo head, memoryStart){
   Queue<Character> stack = new Queue<Character>();
   var next = head.getNext();
   var inf = head.getInf();
   while(next != head){
      stack.push(inf);
      inf = next.getInf();
      next = next.getNext();
   }
   next = head;
   while(!stack.isEmpity()){
      next.setInf(stack.pop());
      next = next.getNext();
   }
}
```

Implementazione

La funzione REV inverte la lista esistente. Tale operazione viene fatta modificando l'informazione contenuta negli elementi grazie all'ausilio dello stack.

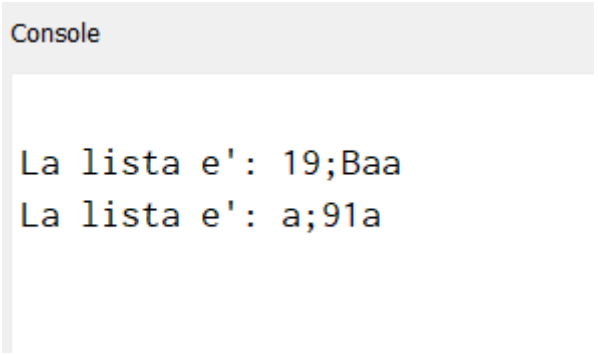
ESECUZIONE DEGLI INPUT DI ESEMPIO

Esempio 1

Input : ADD(1) ~ ADD(a) ~ ADD(a) ~ ADD(B) ~ ADD(;) ~ ADD(9) ~SSX ~SORT~PRNT~DEL(b) ~DEL(B) ~PRI~SDX~REV~PRINTI

Comando corrente	ADD(1)	ADD(a)	ADD(a)	ADD(B)	ADD(;)	ADD(9)	SSX	SORT	PRINT	DEL(b)	DEL(B)	PRI	SDX	REV	PRINT
Elementenți in lista	1	1a	1aa	1aaB	1aaB;	1aaB;9	aaB;91	19;Baa	19;Baa	19;Baa	19;aa	19;aa	a19;a	a;91a	a;91a

OSSERVAZIONI: Ho notato che nell'esempio fornito dal docente una volta eseguito il SORT la lista appare come ;19Baa mentre nel mio programma la lista è 19;Baa questo perché il carattere **;** in ASCI è maggiore del carattere **1** e del carattere **9**, credo quindi ci sia un errore nel testo del progetto.



Esempio 2

Input : “ADD(1) ~ SSX ~ ADD(a) ~ add(B) ~ ADD(B) ~ ADD ~ ADD(9) ~PRINT~SORT(a)~PRINT~DEL(bb) ~DEL(B) ~PRINT~REV~SDX~PRINT

Comando corrente	ADD(1)	SSX	ADD(a)	add(B)	ADD(B)	ADD	ADD(9)	PRINT	SORT(a)	PRINT	DEL(bb)	DEL(B)	PRINT	REV	SDX	PRINT
Elementenli in lista	1	1	1a	1a	1aB	1aB	1aB9	1aB9	1aB9	1aB9	1aB9	1a9	1a9	9a1	19a	19a

Console

```
La lista e': 1aB9
La lista e': 1aB9
La lista e': 1a9
La lista e': 19a
```