Progetto Architetture degli Elaboratori: Lista Concatenata Circolare

Studente: Matteo Bollecchino

Matricola: 7115898

E-mail: matteo.bollecchino@stud.unifi.it

Data di consegna: 27/06/2023

Struttura Generale del Progetto e Inizializzazione Memoria

Il progetto sviluppato si struttura essenzialmente in due parti.

La prima sezione si occupa dell'analisi della stringa di input e della creazione di una nuova stringa senza spazi e con solo i comandi formattati correttamente nell'input iniziale. Invece, la seconda parte si occupa della comprensione della nuova stringa creata e di effettuare le chiamate alle funzioni che andranno a istanziare in memoria e a modificare la lista circolare.

Nella sezione del programma indicata con la direttiva .data e dedicata all'inizializzazione della memoria sono state effettuate le memorizzazioni dei seguenti elementi:

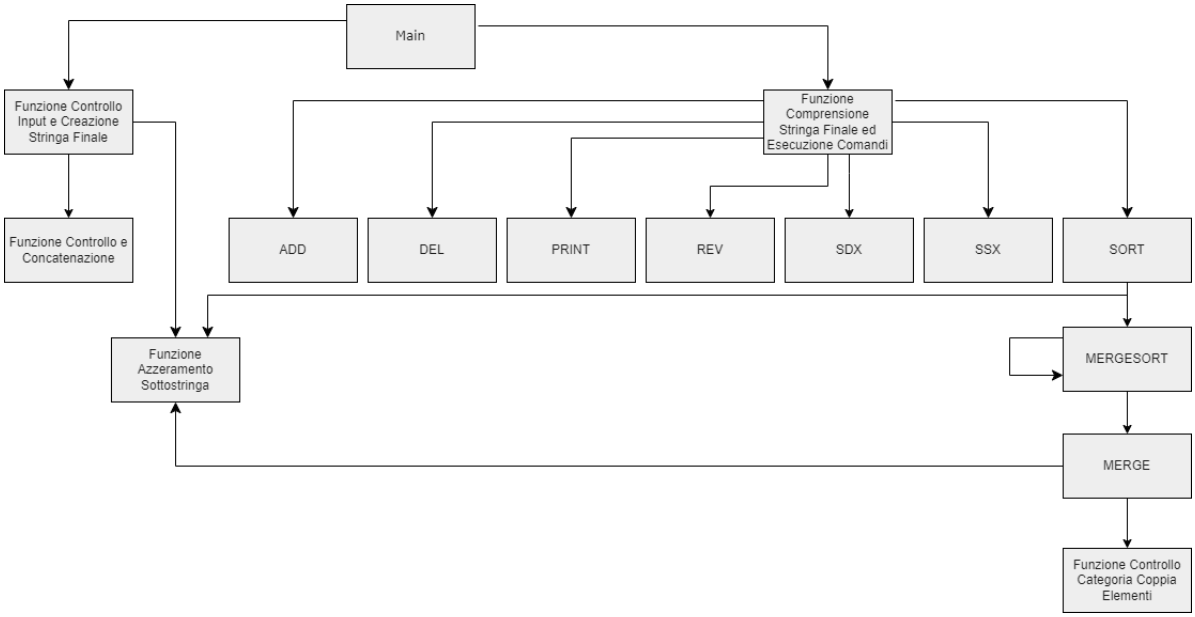
- ListInput dalle quali si dovranno individuare ed eseguire i comandi correttamente formattati
- Codifiche ASCII per i caratteri speciali della tilde e dello spazio vuoto
- Parola, riferitasi con il nome "valore_fine_loop", con utilizzo nella funzione controllo_input_e_creazione_stringa_finale
- Stringa vuota (presenta in memoria solo il fine stringa) che indica l'inizio dell'area di memoria dedicata al puntatore alla testa della lista concatenata circolare da realizzare
- Parola per il puntatore alla testa della lista circolare, inizializzato a 0xffffffff (tale valore indica che la lista è vuota)
- Parola, inizializzata a 0xffffffff, per l'indirizzo di memoria libero in cui memorizzare un nuovo elemento della lista (aggiornato nella funzione ADD)
- Stringa contenente il comando per andare a capo (utilizzato nella funzione PRINT)
- Stringa, contenente solo uno spazio vuoto, dedicata a contenere solo i comandi formattati correttamente (ci si riferirà a tale stringa con l'espressione "stringa finale")

Particolarità del Progetto

Nella parte finale del progetto è presente un'etichetta denominata come "RETURN" che contiene un'unica istruzione (jr ra) utilizzata per la terminazione dell'esecuzione di un metodo. Seppur potrebbe risultare più efficiente effettuare direttamente tale istruzione anziché "saltare" tutte le volte alla label sopra indicata, ho deciso di introdurre tale etichetta per incrementare la leggibilità del codice scritto. Per lo stesso motivo ho deciso di adottare

identificatori autoesplicativi per le labels, anche se a volte possono risultare lunghi. Con il termine “PAHEAD” si indicherà il puntatore all’elemento successivo di un componente della lista realizzata.

Diagramma Chiamate Procedure



Main

Il main è la prima funzione del progetto nonché anche l’ultima nella quale viene invocata la chiamata di sistema per la terminazione del programma.

Input e Output: Nessuno

Registri Utilizzati:

- s0-s11: inizializzati con i caratteri A, D, E, L, P, R, I, N, T, S, X, V ed impiegati in funzione_controllo_e_concatenazione per effettuare le verifiche di correttezza del comando considerato (l’unica lettera rimanente è la “O”, il cui controllo avverrà tramite un registro temporaneo nella procedura sopra indicata)
- a0: contiene l’indirizzo di memoria del carattere iniziale della stringa finale
- a1: contiene inizialmente il puntatore alla ListInput, invece dopo la chiamata alla funzione controllo_input_e_creazione_stringa_finale contiene il primo indirizzo di memoria libero per la memorizzazione degli elementi della lista
- a2: contiene inizialmente la codifica ASCII del carattere tilde; invece, dopo la chiamata al metodo controllo_input_e_creazione_stringa_finale contiene l’indirizzo di memoria del puntatore alla testa della lista
- a3: contiene la parola valore_fine_loop
- a4: contiene il puntatore all’area di memoria utile per la memorizzazione di sottostringhe, è calcolato sommando l’indirizzo contenuto in a0 a 70 (tale numero è ottenuto sommando 60, ovvero il numero di byte massimo che potrà mai contenere la stringa finale, e 10, ovvero un numero di byte di scarto). Dopo la chiamata alla

procedura controllo_input_e_creazione_stringa_finale andrà a contenere il primo indirizzo di memoria utile per iniziare a memorizzare la lista circolare
- a5: contiene limite massimo per quantità di comandi analizzati (30), indipendentemente dalla loro corretta formattazione o meno.

Descrizione Procedura:

Nel main, oltre all'utilizzo dei registri specificati nella sezione "Registri Utilizzati", viene invocato il metodo controllo_input_e_creazione_stringa_finale che restituisce in a0 il puntatore alla coda della stringa finale. Riutilizzando l'area di memoria sfruttata per la memorizzazione delle sottostringhe nell'analisi della ListInput, si salva in memoria il puntatore alla testa della lista da realizzare inizializzandolo a 0xffffffff. Si può così ricavare l'indirizzo libero per la memorizzazione del primo elemento della struttura concatenata sommando 4 byte, ovvero la dimensione di una parola, all'indirizzo contenuto nel registro a4. In seguito, nel main verrà invocato il metodo comprensione_stringa_finale_ed_esecuzione_funzioni durante la cui esecuzione la lista verrà aggiornata o stampata a seconda dei comandi correttamente formattati inseriti nella stringa finale. Infine, verrà effettuata la chiamata di sistema dedita alla conclusione del programma.

Funzione di Controllo Input e Creazione Stringa Finale

Input: a0, a1, a2, a3, a4, a5 **Output:** a0

Gli input sono: a0 contenente il puntatore alla testa della stringa finale, a1 contenente il puntatore alla testa della ListInput, a2 contenente la codifica ASCII per il carattere tilde, a3 contenente valore_fine_loop, a4 contenente l'indirizzo di memoria libera per memorizzare le sottostringhe, a5 contenente la quantità massima di comandi che è possibile analizzare. L'unico output è a0 contenente il puntatore alla coda della stringa finale.

Registri Utilizzati:

- t0: contiene il contatore per loop_controllo_stringa_input
- t1: contiene l'indirizzo in memoria dell'elemento della ListInput preso in considerazione
- t2: contiene l'elemento della ListInput preso in considerazione
- t3: contiene l'indice del carattere da inserire nella sottostringa
- t4: contiene l'indirizzo di memoria della sottostringa in cui memorizzare l'elemento della ListInput considerato
- t5: contiene il contatore di comandi analizzati

Descrizione Procedura:

Nel ciclo della funzione viene scorsa la ListInput finché non si individua un carattere tilde (~), e contemporaneamente la sottostringa analizzata viene memorizzata. In tale caso il

controllo del flusso d'esecuzione viene passato alla label controllo_sottostringa, in cui si effettua una chiamata a funzione_controllo_e_concatenazione. In seguito, viene controllato che la sottostringa memorizzata tramite l'utilizzo del registro a4 sia un comando formattato correttamente o meno. Una volta terminato il controllo viene invocato il metodo azzeramento_sottostringa , affinché venga liberata l'area di memoria occupata dalla sottostringa presa in considerazione per effettuare il controllo. Infine, il registro t3 viene resettato a 0 così che possa fungere da contatore per il ciclo che analizza una nuova sottostringa. Il loop del metodo termina nel momento in cui nel registro t2 viene memorizzato il valore 0 (fine stringa). In tale caso anziché effettuare un salto alla solita etichetta, si passa alla label ultimo_controllo in cui si carica nel registro t2 il valore_fine_loop che lo identifica univocamente e, poiché tale valore viene assunto solo quando si raggiunge la fine della ListInput, sarà allora che verrà fatto il salto ad end_loop_controllo_input_e_creazione_stringa_finale (solo dopo aver effettuato il controllo della correttezza dell'ultimo comando). Tale salto può avvenire anche nel caso in cui venga raggiunto il limite massimo di comandi controllabili.

Funzione Azzeramento Sottostringa

Input: a0 **Output:** nessuno

L'unico input è a0, contenente il puntatore all'area di memoria dedicata alla memorizzazione delle sottostringhe da controllare per l'inserimento, o meno, nella stringa finale.

Registri Utilizzati:

- t0: contiene l'elemento, ovvero zero, da utilizzare per la cancellazione di un elemento della sottostringa
- t1: contiene l'indice dell'elemento della sottostringa da cancellare
- t2: contiene l'indirizzo di memoria dell'elemento della sottostringa preso in considerazione
- t3: contiene un elemento della sottostringa

Descrizione Procedura:

La procedura ha lo scopo di liberare la memoria occupata dalla sottostringa di cui è stata precedentemente analizzata la corretta formattazione come comando, così che con la sovrascrizione successiva non rimangano memorizzati caratteri della stringa precedente. Tale scopo viene raggiunto grazie al ciclo loop_azzeramento_sottostringa, tramite il quale si scorre la sottostringa fino all'elemento di fine stringa (ovvero lo zero). Nel loop ogni volta che si ritrova un elemento, diverso dal fine stringa, questo viene sostituito con il carattere zero. La procedura viene sfruttata anche dai metodi SORT e MERGE per liberare le aree di memoria rispettivamente occupate dall'array contenente tutti gli elementi della lista da ordinare, e dal vettore di supporto utilizzato nel MergeSort.

Funzione di Controllo e Concatenazione

Input: a0, a1, a2, a3 **Output:** a0

Gli input sono: a0 contenente il puntatore ad un elemento della stringa finale, a1 contenente la codifica ASCII del carattere spazio, a2 contenente il puntatore alla testa della sottostringa di cui è necessario analizzare la correttezza, a3 contenente il valore 2 (usato all'etichetta controllo_per_argomento).
L'unico output è a0 che contiene il puntatore alla coda della stringa finale fino a questo momento modificata.

Registri Utilizzati:

- t0: contiene l'indice dell'elemento della sottostringa da analizzare nella sezione di controllo
- t1: contiene l'indirizzo di memoria dell'elemento della sottostringa preso in considerazione
- t2: contiene l'elemento della sottostringa analizzato
- t3: contiene la prima lettera del comando che, se correttamente formattato, sarà concatenato alla stringa finale
- t4: contiene il valore 2 (solo se la codifica ASCII dell'argomento della procedura, ADD o DEL, sia compreso tra 32 e 125) alla label controllo_per_argomento; contiene la codifica ASCII di "(" all'etichetta controllo_per_para, o di ")" alla label controllo_per_parc.
- t5: contiene il secondo carattere di uno dei comandi SDX, SSX, SORT che, se correttamente formattato, sarà concatenato alla stringa finale
- t6: contenente l'argomento del comando ADD o DEL così da poterlo concatenare se necessario

Descrizione Procedura

Il metodo si compone di 2 parti, la prima dove si analizza la correttezza della sottostringa presa in considerazione e la seconda dove, se il comando fosse formattato correttamente, verrebbe eseguita la concatenazione del comando alla stringa finale.

Nella sezione di codice identificata dalla label controllo_spazi_iniziali la sottostringa viene scorsa fintanto che in t2 vengono memorizzati degli spazi. Al primo carattere diverso il controllo del flusso d'esecuzione passa alla label controllo_prima_lettera, con lo scopo di verificare che il carattere memorizzato in t2 sia una delle iniziali dei comandi richiesti (A, D, P, S, R). Si possono quindi distinguere 5 casi:

1. In t2 è presente la codifica ASCII del carattere A e quindi il controllo passa alla label controllo_per_ADD_1;
2. In t2 è presente la codifica ASCII del carattere e D quindi il controllo passa alla label controllo_per_DEL_1;

3. In t2 è presente la codifica ASCII del carattere P e quindi il controllo passa alla label controllo_per_PRINT_1;
4. In t2 è presente la codifica ASCII del carattere S e quindi il controllo passa alla label controllo_per_SDO;
5. In t2 è presente la codifica ASCII del carattere R e quindi il controllo passa alla label controllo_per_REV_1.

Se t2 fosse presente la codifica ASCII di un carattere diverso da quelli indicati precedentemente, allora il controllo sarebbe passato all'etichetta esito_negativo in cui si termina la chiamata al metodo passando alla label RETURN. In ciascuna delle label il cui nome inizia con "controllo_per" si carica in t2 il carattere da controllare della sottostringa e si salta al controllo successivo solo se il confronto è asserito, altrimenti si passa all'etichetta esito_negativo. Risulta particolare il controllo_per_argomento, in quanto l'argomento dei comandi ADD o DEL è ammissibile solo se gli corrisponde una codifica ASCII da 32 a 125 compresi. Quindi per tale controllo si effettueranno 2 confronti che, se asseriti, porteranno il registro t4 ad assumere lo stesso valore del registro a5, dal cui confronto si potrà passare al controllo successivo. Una volta controllata la correttezza del comando si passa alla label controllo_spazi_finali che controlla che in t2 siano memorizzati solo spazi, durante lo scorrimento della sezione finale della sottostringa. Solo in caso di correttezza del comando si passa alla fase di concatenazione. In conclusione, nella stringa finale i comandi della ListInput correttamente formattati saranno presenti nella seguente forma: lettera inziale del nome del comando (nel caso di SORT, SDX e SSX saranno memorizzate le prime 2 lettere) e argomento della funzione, se presente. Nella stringa finale i comandi non sono separati né da spazi né da tildi per semplicità nell'analisi successiva.

Funzione Comprensione Stringa Finale ed Esecuzione Comandi

Poiché dopo l'invocazione di tale funzione il programma termina, questa parte di codice può essere considerata logicamente come se fosse il main. Quindi si utilizza il registro s0 per memorizzare la dimensione della lista circolare.

Input: a0, a1, a2 **Output:** nessuno

Gli input sono: a0 contenente inizialmente il puntatore ad un carattere della stringa finale (inizialmente il primo), a1 contenente l'indirizzo di memoria da cui iniziare a memorizzare la lista concatenata, a2 contenente l'indirizzo di memoria del puntatore alla testa della lista.

Registri Utilizzati:

- s0: contiene la dimensione della lista
- a3: contiene il parametro del comando ADD da inserire nella lista
- t0: contiene l'elemento della stringa finale preso in considerazione

- t1: contiene la codifica ASCII del carattere con cui t0 deve essere confrontato per individuare il comando che vada effettivamente eseguito

Descrizione Procedura

La procedura ha l'obiettivo di individuare il comando, formattato nella stringa finale come specificato in "Funzione di Controllo e Concatenazione", ed invocare la corrispondente funzione. Nella sezione di codice indicata dall'etichetta loop_comprensione_stringa_finale viene identificato il comando da eseguire tramite l'utilizzo del registro t1 (se si presentasse il caso_SDO risulterebbe necessario analizzare anche il secondo carattere memorizzato nella stringa finale per distinguere i casi tra SORT, SDX, SSX). In ognuna delle parti identificate con il nome "preparazione_nomeComando" si invoca la corrispettiva procedura e si effettua un incremento del valore memorizzato in a0, che durante il corso del metodo andrà a rappresentare il puntatore all'elemento della stringa finale analizzato. In caso sia stata effettuata una ADD allora verrà anche aggiornata la dimensione della lista, che potrà essere anche aggiornata anche durante l'esecuzione della funzione DEL. Una volta analizzata l'intera stringa finale allora il metodo terminerà.

ADD

Input: a0, a1, a2, a3, a4 **Output:** nessuno

Gli input sono: a0 contenente la dimensione della lista, a1 contenente l'indirizzo di memoria da cui iniziare a memorizzare la lista concatenata, a2 contenente l'indirizzo di memoria del puntatore alla testa della lista, a3 contenente il parametro da memorizzare all'interno della lista.

Registri Utilizzati:

- t0: contiene l'indirizzo di memoria dell'elemento aggiunto
- t1: contiene l'indirizzo di memoria del puntatore alla testa della lista
- t2: contiene, in caso_std_ADD, il PAHEAD di un elemento della lista

Descrizione Procedura

La prima operazione del metodo consiste nel salvataggio dell'elemento contenuto in a3 in coda alla lista. Vengono quindi distinti 2 casi:

1. La lista prima dell'inserimento era vuota, quindi la memorizzazione dell'elemento è seguita dall'aggiornamento del puntatore alla testa precedentemente inizializzato a 0xffffffff nel main e dal salvataggio in memoria del puntatore all'elemento successivo (o PAHEAD), in questo caso sé stesso
2. La lista prima dell'inserimento non era vuota, quindi per effettuare la ADD è necessario scorrere la struttura dati fino all'ultimo elemento, il cui PAHEAD punta alla testa della lista. Solo una volta arrivati alla coda verrà memorizzato il nuovo elemento, modificato il PAHEAD dell'attuale penultimo dato affinché punti alla

nuova coda e, infine, settato il PAHEAD della coda con il puntatore alla testa della lista.

In entrambi i casi, prima della chiusura della chiamata al metodo, viene aggiornata l'area destinata all' indirizzo memoria maggiore, ovvero da cui si riscontra memoria "libera".

DEL

Input: a0, a1 **Output:** nessuno

Gli input sono: a0 contenente la dimensione della lista, a1 contenente il parametro da eliminare dalla lista.

Registri Utilizzati:

- t0: contiene inizialmente il puntatore alla testa della lista e successivamente l'indirizzo di memoria del primo byte del puntatore all'elemento successivo (o PAHEAD) di un elemento della lista
- t1: contiene l'elemento da controllare della lista
- t2: contiene il PAHEAD di un componente della lista
- t3: contiene l'indirizzo di memoria del primo byte del PAHEAD dell'elemento precedente a quello analizzato
- t4: contiene il puntatore alla testa della lista

Descrizione Procedura

La procedura, nel caso in cui la lista sia vuota, non compie alcun tipo di operazione. Se invece fosse presente un unico elemento e corrispondesse al carattere da eliminare, allora l'operazione avverrebbe regolarmente resettando il puntatore alla testa della lista a 0xffffffff. In casi diversi da quelli precedentemente descritti, l'eliminazione allora avviene andando a modificare il campo PAHEAD dell'elemento precedente a quello da cancellare con il puntatore al suo successivo. In particolare, con l'eliminazione del primo elemento della lista il puntatore alla testa della struttura dati e il campo PAHEAD dell'ultimo componente della lista vengono aggiornati con l'indirizzo del secondo elemento. Invece, alla cancellazione dell'ultimo elemento, il PAHEAD di quello che precedentemente all'operazione era il penultimo elemento viene settato con il puntatore alla testa della struttura. Nel caso in cui sia stato cancellato l'ultimo elemento della lista e la lista alla fine risultasse vuota, allora il puntatore alla testa verrebbe resettato al valore 0xffffffff. Dopo ogni cancellazione di un componente della lista, il registro che memorizza la dimensione viene decrementato.

PRINT

Input: a0, a1 **Output:** nessuno

Gli input sono: a0 che conterrà ciò che si vuole stampare a video, a1 contenente la dimensione della lista.

Registri Utilizzati:

- t0: contiene il puntatore alla testa della lista
- t1: contiene l'indirizzo di memoria del primo byte del PAHEAD dell'elemento stampato
- t2: contiene il PAHEAD di un componente della lista

Descrizione Procedura

Il metodo, indipendentemente dalla sua dimensione stampa una riga a capo. Nel caso in cui la lista sia vuota la procedura non svolge alcun tipo di operazione. Altrimenti, a partire dalla testa della struttura dati, in primis si stampa l'informazione del componente della lista e in seguito, caricando in t2 il suo puntatore all'elemento successivo, si ripete il processo fino a quando non si arriva alla coda della lista, individuabile dal fatto che il suo campo PAHEAD corrisponde al puntatore alla testa.

SSX

Input: a0 **Output:** nessuno

Gli input sono: a0 contenente la dimensione della lista.

Registri Utilizzati:

- t0: contiene il puntatore alla testa della lista
- t1: contiene il PAHEAD dell'elemento successivo alla testa della lista

Descrizione Procedura

La procedura, nel caso in cui la lista sia vuota o contenga un unico elemento, non compie alcun tipo di operazione. In caso contrario, lo shift a sinistra della lista avviene modificando il puntatore alla testa della struttura, facendo sì che punti a quello che prima dell'operazione costituiva il secondo componente della lista.

SDX

Input: a0 Output: nessuno

Gli input sono: a0 contenente la dimensione della lista.

Registri Utilizzati:

- t0: contiene il puntatore alla testa della lista
- t1: contiene l'indirizzo di memoria del primo byte del PAHEAD dell'elemento preso in considerazione
- t2: contiene il PAHEAD del componente della lista considerato

Descrizione Procedura

La procedura, nel caso in cui la lista sia vuota o contenga un unico elemento, non compie alcun tipo di operazione. In caso contrario la lista viene scorsa tramite loop_SDX fino all'ultimo elemento della struttura. L'operazione finale per effettuare lo shift a destra è quindi quella di settare il puntatore alla testa della lista con l'indirizzo di memoria dell'elemento a cui si era arrivati tramite il ciclo.

REV

Input: a0 **Output:** nessuno

Gli input sono: a0 contenente la dimensione della lista.

Registri Utilizzati:

- t0: contiene il puntatore alla testa della lista
- t1: contiene l'indirizzo di memoria del primo byte del PAHEAD dell'elemento preso in considerazione
- t2: contiene il PAHEAD del componente della lista considerato

Descrizione Procedura

La procedura, nel caso in cui la lista sia vuota o contenga un unico elemento, non compie alcun tipo di operazione. Altrimenti, tramite il ciclo indicato dalla label loop_REV_inserimento che scorre la lista dalla testa alla sua coda, le informazioni degli elementi della struttura vengono caricate nello stack. Invece è attraverso loop_REV_sovrascrizione che le informazioni vengono riprese dalla stack nell'ordine inverso rispetto a quello in cui ci erano state inserite. Contemporaneamente la lista viene scorsa e si sostituisce l'informazione con la corrispettiva opposta prelevata dallo stack. Si ottiene così come risultato la lista iniziale ma con le informazioni degli elementi invertite.

SORT

L'obiettivo finale della procedura è quello di memorizzare gli elementi della lista in area memoria libera (poiché le celle di memoria sono consecutive lo si considera un array), in seguito eseguire l'ordinamento dei dati, e infine reinserire gli elementi ordinati nella lista.

Input: a0 **Output:** nessuno

Gli input sono: a0 contenente la dimensione della lista.

Registri Utilizzati:

- t0: contiene il puntatore alla testa della lista
- t1: contiene inizialmente il puntatore alla cella di memoria dell'array in cui inserire l'elemento della lista memorizzato nel registro t2, e in seguito contiene l'indirizzo di memoria di un elemento del vettore da reinserire nella lista
- t2: contiene inizialmente l'elemento della lista che deve essere inserito nell'array, e in seguito memorizza l'indirizzo di memoria del primo byte del PAHEAD dell'elemento preso in considerazione.
- t3: contiene inizialmente l'indirizzo di memoria del primo byte del PAHEAD dell'elemento preso in considerazione, e in seguito memorizza l'elemento del vettore da reinserire nella lista
- t4: contiene il PAHEAD del componente della lista considerato
- a0: contiene l'indirizzo di memoria del primo elemento nell'array
- a1: contiene l'indirizzo di memoria dell'ultimo elemento nell'array
- a2: contiene il puntatore alla testa dell'array di supporto che verrà utilizzato per la realizzazione dell'algoritmo di Merge Sort.

Descrizione Procedura

La procedura, nel caso in cui la lista sia vuota o contenga un unico elemento, non compie alcun tipo di operazione. Altrimenti si hanno le condizioni per essere un caso standard in cui applicare l'ordinamento degli elementi della lista. In tale condizione si identifica un'area di memoria libera dedicata all'array da ordinare, in 20 byte di scarto dall'indirizzo di memoria maggiore per la memorizzazione della lista. In loop_SORT_inserimento la struttura dati iniziale viene scorsa e per ogni suo componente l'informazione viene inserita nel vettore realizzato. In seguito, il controllo del flusso d'esecuzione viene passato alla funzione MERGESORT, in cui vengono gestite le chiamate ricorsive tipiche del Merge Sort. Successivamente, poiché ormai i dati contenuti nel vettore sono stati ordinati, alla label reinserimento_lista la lista viene scorsa e si immettono le informazioni dall'array ordinato alla struttura dati da realizzare. Infine, il vettore viene svuotato delle sue informazioni e la memoria precedentemente occupata viene liberata.

MERGESORT

Input: a0, a1, a2 **Output:** nessuno

Gli input sono: a0 contenente l'indirizzo di memoria del primo elemento memorizzato nell'array, a1 contenente l'indirizzo di memoria dell'ultimo elemento memorizzato nell'array, a2 contenente il puntatore alla testa dell'array di supporto sfruttato per l'implementazione del Merge Sort ricorsivo.

Registri Utilizzati:

- t0: utilizzato per il calcolo dell'indirizzo dell'elemento centrale dell'array considerato nella chiamata ricorsiva
- a1: contiene l'indirizzo di memoria dell'elemento centrale del sottoarray considerato
- a2: contiene l'indirizzo di memoria dell'elemento finale del sottoarray considerato
- a3: contiene il puntatore alla testa dell'array di supporto

Descrizione Procedura

La procedura sfrutta la struttura dello stack per organizzare le chiamate ricorsive tipiche dell'algoritmo d'ordinamento Merge Sort.

MERGE

Lo scopo della procedura è quella fondere 2 array già ordinati in un vettore a sua volta ordinato, la cui dimensione è data dalla somma delle dimensioni dei vettori di partenza. In seguito, si identificherà il vettore di cui vanno ordinati gli elementi con il nome "A".

Input: a0, a1, a2, a3 **Output:** nessuno

Gli input sono: a0 contenente l'indirizzo di memoria del primo elemento del sottoarray sinistro da ordinare (verrà identificato dal nome "left1"), a1 contenente l'indirizzo di memoria dell'ultimo elemento del sottoarray sinistro da ordinare (verrà identificato dal nome "right1"), a2 contenente l'indirizzo di memoria dell'ultimo elemento del sottoarray destro da ordinare (verrà identificato dal nome "right2"), a3 contenente il puntatore alla testa dell'array di supporto sfruttato per l'implementazione del Merge Sort ricorsivo.

Registri Utilizzati:

- t0: contiene l'indirizzo di memoria del primo elemento del sottoarray destro da ordinare (verrà identificato dal nome "left2")
- t1: contiene la dimensione dell'array dato dalla fusione dei 2 sottoarray e identificato dall'operazione right2-left1+1
- t2: contiene A[left1] prima della chiamata alla funzione CONTROLLO_CATEGORIA_COPPIA_ELEMENTI, e dopo l'invocazione contiene la categoria di A[left1]
- t3: contiene A[left2] prima della chiamata alla funzione CONTROLLO_CATEGORIA_COPPIA_ELEMENTI, e dopo l'invocazione contiene la categoria di A[left2]
- t4: contiene il limite sinistro array supporto (viene usato nella sezione di codice identificate dalla label ricopia_array)
- t5: viene usato come registro di supporto nelle parti del programma identificate dalle etichette if_left1>right1 e if_left1<=right1
- t6: contiene il carattere da ricopiare in modo ordinato nell'array di partenza

Descrizione Procedura

Nella procedura, fintanto che letf1 <= right1 e left2 <= right2, si confronta gli elementi A[left1] e A[left2]. Se il primo è minore del secondo allora verrà inserito nel vettore di supporto, altrimenti verrà inserito l'altro elemento e si scorre l'array contenente tale elemento inserito. Nel momento in cui uno dei 2 sottoarray ordinati, che si ha l'obiettivo di fondere in un unico vettore ordinato, è stato scorso completamente allora si procederà con l'analisi dell'altro fino ad aver scorso anche quest'ultimo. In seguito, passando alla label ricopia_array, gli elementi dell'array di supporto verranno reinseriti ordinatamente nel vettore di partenza. Infine, l'area di memoria occupata dal vettore di supporto verrà liberata.

Funzione Controllo Categoria Coppia Elementi

Input: a0, a1 **Output**: a0, a1

Si identificano 4 categorie, l'appartenenza alle quali dipende dalla codifica ASCII degli elementi passati come parametri di input alla procedura: carattere speciale (categoria 0), numero (48-57) (categoria 1), lettera minuscola (97-122) (categoria 2), lettera maiuscola (65-90) (categoria 3). Per la categoria 0 non sono definiti limiti poiché gli corrispondono tutti gli elementi con codifiche ASCII comprese tra 32 e 125, e che non appartengano a nessuna delle restanti categorie.
Gli input sono: a0 e a1 contenenti 2 elementi di un array che devono essere confrontati.
Gli output sono: a0 e a1 contenenti le 2 categorie corrispondenti agli elementi se questi appartengono a categorie diverse, altrimenti corrisponderanno agli elementi stessi.

Registri Utilizzati:

- t0: contiene uno dei valori limiti delle categorie sopra descritte
- t1: contiene la categoria di a0
- t2: contiene la categoria di a1

Descrizione Procedura

La procedura per ognuno dei 2 elementi verifica l'appartenenza a 3 range ben determinati di codifiche ASCII alle label controllo_numero, controllo_lettera_minuscola, controllo_lettera_maiuscola. Se invece i parametri di input non corrispondono a nessuna delle categorie precedenti, allora essi rappresenteranno dei caratteri speciali e sarà verificata la condizione dell'etichetta controllo_carattere_speciale. Infine, una volta effettuati i controlli necessari, il controllo del flusso d'esecuzione passa alla label controllo_categorie in cui si presentano 2 casi:

1. I parametri di input appartengono alla stessa categoria e quindi nei registri a0 e a1 saranno già presenti i dati che si vuole ritornare e che verranno confrontati poi nella funzione MERGE.

2. Gli elementi controllati appartengono a categorie diverse e quindi i valori contenuti nei registri t1 e t2 vengono rispettivamente passati ai registri a0 e a1 che costituiranno l'output della procedura.

Test 1: primo esempio PDF (con aggiunta di PRINT)

listInput1 = "ADD(1) ~ PRINT~ SSX ~ PRINT~ ADD(a) ~ add(B) ~ ADD(B) ~ ADD ~ ADD(9) ~PRINT~SORT(a)~PRINT~DEL(bb) ~DEL(B) ~PRINT~REV~PRINT~SDX~PRINT".

```
Console
A1PSSPAaABA9PPDBPRPSDP
1
1
1aB9
1aB9
1a9
9a1
19a
Program exited with code: 0
```

0x10000698	0x41000a10	0x10	0x0a	0x00	0x41
0x10000694	0x0006f910	0x10	0xf9	0x06	0x00
0x10000690	0x0006f400	0x00	0xf4	0x06	0x00

Tale test viene sfruttato per mostrare come i comandi mal formattati, o errati, vengano esclusi dall'inserimento nella stringa finale, dove invece sono presenti i comandi effettivamente eseguiti. La prima riga dell'output costituisce la stringa finale, mentre le righe successive corrispondono alle stampe, così dilazionate per mostrare l'evoluzione della struttura dati dopo ogni comando eseguito. La parte di memoria mostrata contiene il puntatore alla testa della lista (da 0x10000691 a 0x10000694) e per l'indirizzo di memoria libero in cui memorizzare un nuovo elemento della lista (da 0x10000695 a 0x10000698).

Test 2: secondo esempio PDF (con aggiunta di PRINT)

listInput2 = "ADD(1) ~ ADD(a) ~ ADD(a) ~ ADD(B) ~ ADD(;) ~ ADD(9) ~ PRINT ~SSX~PRINT~SORT~PRINT~DEL(b) ~DEL(B) ~PRINT~PRI~SDX~PRINT~REV~PRINT".

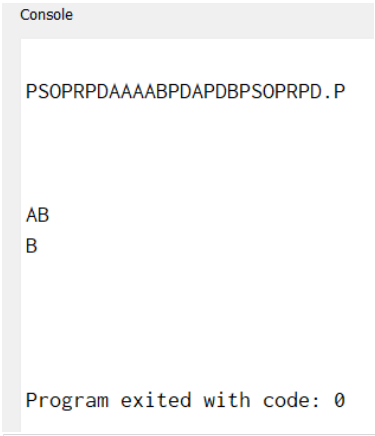
```
Console
A1AaAaABA;A9PSSPSOPDbDBPSDPRP
1aaB;9
aaB;91
;19aaB
;19aa
a;19a
a91;a
Program exited with code: 0
```

0x10000698	0x41000a10	0x10	0x0a	0x00	0x41
0x10000694	0x00070310	0x10	0x03	0x07	0x00
0x10000690	0x0006fe00	0x00	0xfe	0x06	0x00

Tale test viene sfruttato per mostrare come tutti i comandi vengano correttamente eseguiti. La prima riga dell'output costituisce la stringa finale, mentre le righe successive corrispondono alle stampe, così dilazionate per mostrare l'evoluzione della struttura dati dopo ogni comando eseguito. La parte di memoria mostrata contiene il puntatore alla testa della lista (da 0x10000691 a 0x10000694) e per l'indirizzo di memoria libero in cui memorizzare un nuovo elemento della lista (da 0x10000695 a 0x10000698).

Test 3

ListInput3 = "PRINT~SORT~PRINT~REV~PRINT~DEL(A) ~ADD(A) ~ADD(B)~PRINT ~DEL(A) ~PRINT~DEL(B)~PRINT~SORT~PRINT~REV~PRINT~DEL(.) ~PRINT"



0x100006ac	0x10000706	0x06	0x07	0x00	0x10
0x100006a8	0xffffffff	0xff	0xff	0xff	0xff

Tale test viene sfruttato per mostrare il comportamento del programma per eventuali operazioni svolte a catena vuota, sia con la catena ancora da inizializzare, sia con catena riempita e successivamente svuotata. La prima riga dell'output costituisce la stringa finale, mentre le righe successive corrispondono alle stampe, così dilazionate per mostrare l'evoluzione della struttura dati dopo ogni comando eseguito. La parte di memoria mostrata contiene il puntatore alla testa della lista (da 0x100006a8 a 0x100006ab), settato a 0xffffffff poiché la lista una volta svolte tutte le operazioni è vuota, e per l'indirizzo di memoria libero in cui memorizzare un nuovo elemento della lista (da 0x100006ac a 0x100006af).