Ferdiando Del Trionfo 566/739 Paolo Selce 50/747
Gruppo 27

Secondo progetto

Analisi del problema:

Realizzare un software, che dato un file di testo composto da TAG e PAROLE ne verifichi la correttezza sintattica seguendo le seguenti regole:
Ogni TAG deve essere delimitato da i seguenti caratteri < > .
Ogni TAG di Apertura deve essere accoppiato da un TAG di chiusura ; il TAG di chiusura e' delimitato dai caratteri <\ >.
Ogni testo deve essere iniziato dal TAG <testo> e finito dal TAG <\testo>
La coppia di TAG <testo><\testo> possono inglobare zero o piu TAG <sec> <\sec>
La coppia di TAG <sec> <\sec> possono inglobare zero o piu TAG <\div>
La coppia di TAG <\div> possono inglobare zero o piu TAG <par> <\par>
La coppia di TAG <par> <\par> puo inglobare zero o piu PAROLE oppure puo inglobare TAG <bold><\bold> o <it><\it>
La coppia di TAG <bold><\bold> e <it><\it> si possono inglobare a vicenda oppure PAROLE.
Il testo affinche sia corretto deve seguire le regole su citate e i TAG devono seguire il teorema della parentesizzazione (ad ogni parentesi aperta ne corrisponde una chiusa e non deve intrecciarsi con altre parentesi).
Inoltre Il software dovra' prevedere l'immagazzinamento e la stampa delle i-esime parole di un paragrafo e fare la congiunzione con le k-esime parole dei paragrafi, senza ripetizioni.

dettagli funzionali:
L' utente dovra' inserire il nome del file del file da analizzare e i numeri delle i-esime e k-esime parole.

Fase progettuale:

Vincoli progettuali
Strutture dati utilizzabili: array con allocazione dinamica o statica, liste,stack implementati con array,
Riutilizzo massimo delle funzioni utilizzate nel primo progetto.

Dettagli progettuali
Il nome e i delineatori dei TAG sono ristretto al programma cioe' non possono essere cambiati dall'utente.

Dall'analisi effettuata si evinco 4 moduli di sviluppo:

- Costruzione e stampa menu
- Scansione e verifica del testo con la stampa della correttezza
- Acquisizione e stampa delle i-esime PAROLE dei paragrafi
- Unione, ordinamento e stampa delle i-esime e delle k-esime parole dei paragrafi (Opzionale)

(Riutilizzo del codice e della documentazione interna esterna del primo progetto)

Analisi e descrizione del modulo Costruzione e stampa menu

La costruzione e la stampa del menu e' la parte di codice che serve all'utente per interfacciarsi con il programma. Infatti una delle richieste esplicite fatte dal cliente e' la possibilita di scegliere una delle seguenti opzioni: Verifica testo, stampa le i-esime parole,Unisci i-esime e k-esime parole. L'interfaccia grafica dovra' essere composta da un'intestazione, una sequenza di scelte, e la domanda di scelta; come di seguito:

MENU
1 Scelta1
2 Scelta2
Fai la tua scelta:

La soluzione ottimale sarebbe implementare tale menu di scelta con la possibilita di riutilizzarlo e di fare nuove aggiunte al menu in futuro. Tale scelta ci vincola a progettare la gestione del menu con file di appoggio in modo da poterli modificare.

sequenza naturale:
- Costruisci il menu
- Stampa menu

Analisi e descrizione del modulo Scansione e verifica del testo con la stampa della correttezza

Questo modulo indica la corretta sintassi del testo. Infatti analizza il testo da file e comunica all'utente se il testo e' corretto.

sequenza naturale:
- Carica il file
- Analizza il testo
- Stampa esito

Questo modulo prevede una classificazione del contenuto del file:

- TESTO
- TAG

Un TAG e' riconoscibile dai seguenti caratteri '<' * '>' . Un TAG e' composto da un TAG di apertura

e un TAG di chiusura, il tag di chiusura e' riconoscibile dai caratteri '<\' * '>'.
Un TAG puo avere le seguenti caratteristiche:Puo contenere Testo,Puo Essere ripetibile.
Puo' contenere testo significa che il tag di apertura e chiusura possono inglobare del teso, mentre puo essere ripetibile significa che i tag dello stesso livello di inglobamento si possono inglobare.

Un TESTO e' rappresento da una stringa di caratteri (<> esclusi).
N.b. La stringa composta da soli caratteri di spaziatura non e' considerata un TESTO.

Questo modulo prevede il riconoscimento dei TAG anche se nel suo interno ci sono dei caratteri differenti dalle lettere.
esempio: < t 5 e s t o> questo equivale al riconoscimento de tag <testo>.

Analisi e descrizione del modulo Acquisizione e stampa delle i-esime parole dei paragrafi
Questo modulo prevede l'acquisizione e stampa dell' i-esime parole dei paragrafi. Ovviamente verranno inserite in una struttura dati e successivamente scorse e stampate a video. Nel caso non ci fosse l' i-esima parola nel paragrafo allora verra' creata una parola fittizia xxx.

sequenza naturale:
- Carica il file
- Analizza il testo
- Deposita le i-esime parole nella struttura dati
- Stampa esito

Questo modulo non garantisce la corretta sintassi essa fornisce una successione di i-esime di parole. Ogni tag deve essere scritto correttamente.(compresi gli spazi)

Analisi e descrizione del modulo
Unione, ordinamento e stampa delle i-esime e delle k-esime parole dei paragrafi (Opzionale)
Il quarto ed ultimo punto prevede la fusione delle parole trovate al i-esimo e al k-simo paragrafo. Le parole dovranno essere fuse in un unica struttura senza ripetizioni e in maniera ordinata. L'algoritmo scelto per l'ordinamento deve essere L'insertion sort che mette al posto giusto la parola ordinata. Dopo tale lavoro apparira' a video il risultato della fusione.

sequenza naturale:
- Carica il file
- Analizza il testo
- Deposita le k-esime parole nella struttura dati
- Fai il Merge con le i-esime parole
- Stampa esito

Questo modulo e' facoltativo.

Cotruzione e stampa del menu (Interfaccia grafica)

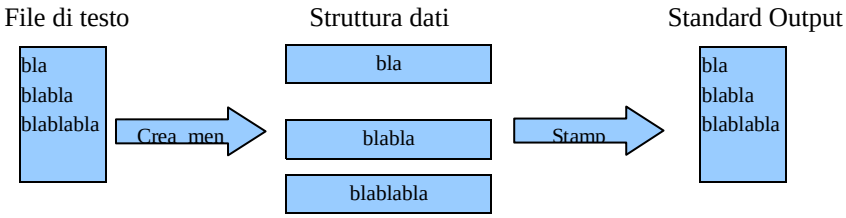
(Riutilizzo del codice e della documentazione interna esterna del primo progetto)

(N.b. questa sezione nel progetto precedente in fase di correzione e' stata definita inutile perche non prevedeva funzioni di controllo di input del menu. Ma queste funzioni non erano state progettate per una gestione completa del menu ma solo per l'acquisizione da file e stampa del menu. Infatti adesso la stiamo riutilizzando)

L'interfaccia grafica visualizzata all'utente all'avvio del programma sara' formata da un menu' composto dalle seguenti selezioni :

1 Carica e analizza file
2 Carica e stampa le i-esime parole dei paragrafi
3 Carica, unisci e stampa le k-sime parole
4 Uscita

L'idea di creare un menu riutilizzabile e aggiornabile e' la seguente: costruire una funzione che prenda da un file di testo (con una sintassi e semantica prestabilita) un Menu. Tale funzione riempira' una struttura dati contenente il menu. Tale menu potrebbe essere formattato visualizzato a proprio piacimento infatti parallelamente a questa funzione ce ne dovra' essere un' altra la quale stampera' tale menu.



Pascal Like

La funzione Crea_menu(file_di_testo) prende come parametro di input una stringa che rappresenta il nome del file di testo; tale file di testo contiene la struttura del menu.
Sintassi e Semantica del file:
Il file e' composto da una parte d' intestazione, da una parte di scelte, e dall'ultima parte di domanda di selezione.
la prima riga rappresenta l'intestazione del menu.
Le righe restanti rappresentano le scelte del menu.
L'ultima riga rappresenta la domanda da visualizzare per le scelte.
valori ritornati:
La funzione ritorna un array di stringhe il quale conterra' al primo posto il nome del menu e nei restanti posti prima dell'ultimo le relative scelte; e nell'ultimo posto la domanda di scelta.

Inoltre la funzione ritorna il numero di elementi inseriti nell'array.

esempio:

File_di_testo preso in input

Menu
1-Scelta1
2-Scelta2
fai una scelta:

Array ritornato

L'array ritornato verra' stampato a video dalla funzione stampa_menu(array,elementi) che prende in input l'array ritornato dalla funzione e il numero di elementi crea_menu.

PROCEDURE e PARAMETRI
Crea_Menu(INPUT:STRING file_di_testo OUTPUT:STRING ARRAY array , INT elem)
Stampa_Menu(INPUT: STRING ARRAY array , INT elem OUTPUT:)

Dettagli implementativi (orientato a C)
Le funzioni di creazione e stampa menu' sono implementate nel file funzioni_menu.c e definite nel file funzioni_menu.h .
La funzione Crea_Menu non puo costruire menu piu lunghi di 9 scelte per evidenti motivi.
La funzione Stampa_Menu stampa a video senza formattazioni speciali.
prototipi di funzioni:
char ** Crea_Menu(char* ,int);
void Stampa_Menu(char **,int);
I dettagli delle parti implementate si possono trovare nella documentazione interna.

Scansione e verifica del testo con la stampa della correttezza

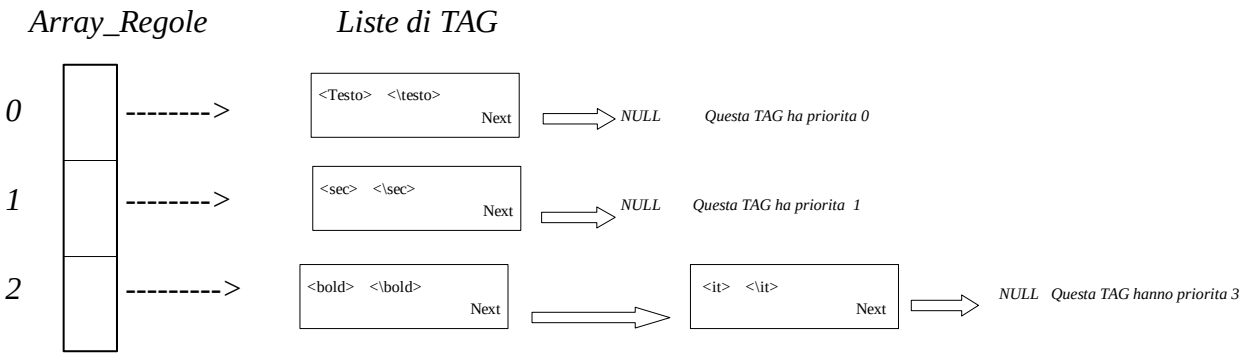
Lo schema principale per il controllo della correttezza sintattica e quella di leggere sequenzialmente il file e riconoscere i TAG dal TESTO e se segue le regole su citate allora la sintassi e' corretta.
L'idea fondamentale di questo modulo e' capire se ad ogni tag aperto ne corrisponde uno chiuso e se per ogni tag aperto se ne puo inserire un altro.
La parte su cui focalizzare l'attenzione per questo modulo e' la parte che riguarda le regole di un TAG. Bisogna prima stabilire la struttura in cui ci sono le regole che i TAG devono rispettare e poi stabilire la struttura e i comportamenti che i TAG devono assumere dopo giudicati positivi alle regole.

Strutture dati fondamentali

Dall'analisi su fatta un TAG e' composto da: Tag di apertura,Tag di chiusura, parola, ripetizioni, priorita. Quindi un Tag potrebbe essere rappresentato da una struttura dati cosi fatta:

```
typedef struct Tag {
int Parola;                 /*1 se viene accettata un Testo 0 se non viene accettata*/
int Ripetizioni;            /*Questa Flag indica che un TAG dello stessa priorita' puo essere aperto*/
int Priorita;               /*Rappresenta la priorita del TAG*/
char * Tag_Apertura;        /*nome tag di apertura*/
char * Tag_Chiusura;        /*nome tag di chiusura*/
} TAG;
```

Un insieme di regole per i TAG e' formato da un Array di Liste di TAG; Dove L'indice dell'array rappresenta il livello di inglobamento (che e' uguale al campo priorita' del TAG).
Ovviamente i TAG con la stessa priorita stanno nella stessa lista e cio significa che sono tag fratelli.
esempio:



<testo> <sev> <bold><\bold><it><\it> <\sev> <testo> Questa stringa reispetta le regole dell'esempio

Nel disegno dovrebbe esserci un puntatore per ogni oggetto della lista che punta alla struttura TAG perche il nodo di una lista possiede un puntatore ad un indirizzo di memoria (in questo caso al tag) (riferimento a struttura dati e routine ausiliare).
Capito come sono composte le regole, bisogna stabilire come un tag letto viene giudicato idoneo, dopodiche capirne cosa farcene.
Un tag e idoneo alle regole se:

supponiamo che (UTA)=Ultimo Tag Aperto e (TL)=Tag Letto
UTA rappresenta l'ultimo tag che e' stato aggiunto, metre TL rappresenta il tag candidato per diventare UTA (cioe da aprire).

Per i Tag (TL) di Apertura:
In base alla priorita un tag puo essere aperto se e solo se rispetta la seguente regola

$$priorita(UTA) + 1 = priorita(TL)$$

Mentre solo nel caso che UTA ha la proprieta di ripetibilita si controllano anche i tag che rispettano la seguente regole:

$$priorita(UTA) = priorita(TL)$$

n.b. Ovviamente nel caso ci fossero piu TAG nella lista puntata dall'array verrebbero scorsi tutti e trovato il tag con nome tag di apertura uguale.

sequneza:
- aggiunto 1 alla priorita di UTA
- accedere nell'array delle regole alla posizione priorita+1
- scorrere la lista e verificare se TL di apertura corrisponde ad uno dei TAG della lista

in maniera duale viene fatto se TL ha la proprieta ripetibilita attiva.

Per i Tag (TL) di Chiusura
Un Tag di chiusura rispetta le regole se e solo se:

$$Tag\_Chiusura(UTA) = Tag\_Chiusura(TL)$$

Una volta stabilito le regole dei tag bisogna capire che cosa succede quando un tag e' stato definito idoneo.
premessa - La correttezza della sintassi si basa sul fatto che i tag di apertura e chiusura devono rispettare il teorema della parentesizzazione.
TEOREMA parentesizzazione
Ad ogni parentesi di apertura della stessa specie deve corrispondere una parentesi della stessa specie di chiusura. Le parentesi non possono intrecciarsi cioe devono inglobarsi.
esempio:
buono {[()][()]} errore {(})

Per garantire tale teorema si puo pensare di inserire i tag di apertura idonei in uno stack, mentre estrarli dallo stack se il tag di chiusura e' idoneo. La correttezza sara garantita se e solo se alla fine del file lo stack e' vuoto.
Dettagli implementativi (orientati a C)
La prima fase prevede il riconoscimento dei TAG dal TESTO.
Leggo singolarmente ogni carattere e le inserisco in una stringa.
Controllo se tale stringa e' valida (Una stringa e' valida se contiene un TAG oppure un TESTO)
La Seconda Fase prevede il riconoscimento di un tag di apertura, di chiusura oppure testo.

I 3 Casi:

TAG di Apertura
Controllo se il tag letto rispetta le regole col tag in testa alla pila. Esito positivo faccio un push del tag letto sullo stack, esito negativo errore sintassi con terminazione lettura file.
TAG di Chiusura
Controllo se il tag letto corrisponde al tag di chiusura del tag sullo stack. Esito positivo faccio un pop del tag sullo stack, esito negativo errore sintassi con terminazione lettura file.
TESTO
Controllo che il tag sulla testa dello stack possa inglobare TESTO tale controllo viene fatto se la proprita parola del Tag e' attiva

La terza ed ultima fase prevede il controllo della correttezza sintattica. Se lo stack e' vuoto e non ci sono stati errori in precedenza allora l'esito e' positivo.

Prototipi di Funzioni:
(Funzioni che allocano e gestiscno i tag)

```
/*Questa funzione alloca una struttura TAG e la inizializza il valore di ritorno e' il puntatore alla struttura
INPUT: puntatore a stringa Tag di apertura, puratore a stringa Tag di chiusura,parola,ripetizioni sono
Flag che possono assumere valori 0 e 1,ordine priorita rappresenta la priorita */
TAG * Alloca_Tag(char * Tag_Apertura,char * Tag_Chiusura,int Parola,int Ripetizioni,int Priorita);

/*Questa funzione ritorna 0 se la variabile Tag non e' un tag 1 se e' un tag di apertura 2 se e' un tag di chiusura*/
int IsTag(char * Tag);

/*Questa funzione dealloca il Tag */
void Dealloca_Tag(void * Tag);
```

Queste funzioni sono contenute nei file tag.c e tag.h

(Funzioni che Analizzano il testo)

```
/*Funzione che costruisce le regole di inclusione TAG (Queste regole sono statiche possono essere cambiate sole se
create nuove funzioni regole) ritorna una array di liste_regole*/
LIST ** Costruisci_Regole();

/*Questa funzione controlla se il TagDaInserire rispetti le regole del TagInTesta
INPUT Lista dell'Array delle Regole, Il TagInTesta reppresenta il tag nella testa dello stack,TagDaInserire rappresenta il
tag che vorrei inserire in testa
n.b. se il TagIntesta e' NULLO allora per default si controlla la priorita piu bassa 0
OUTPUT ritorna l'indirzzo della struttura TAG (allocato) se lo si puo inserire altrimenti NULL*/
TAG *Poss_Inserire(LIST ** Array_Regole,TAG * TagInTesta,char * TagDaInserire);

/*Questa funzione controlla se il parametro TagDaEstrarre rispetta le regole del TagDiTesta (deve essere un tag di
chiusura)
INPUT: TagDiTesta rappresenta il tag sullo stack mentre il TagDaEstrarre reppresenta il tag incontrato nel testo
```

```
OUTPUT : Ritorna 1 se lo posso estrarre, altrimenti 0*/
int Poss_Estrarre(LIST ** Array_Regole,TAG *TagDiTesta,char *TagDaEstrarre);

/*Questa funzione controlla se il TAG puo contenere testo se il tag e NULL per definizione non puo essereci testo.
ritorna 1 se si puo inserire 0 atrimenti
INPUT Tag da conrollare*/
int Contr_Text(TAG * Tag);

/*Funzione che analizza un file di testo
INPUT: path del file
OUTPUT:ritorna 0 se la correttezza e' verificata 1 se e' errore di TAg 2 se e' errore di stack 3 se e' erroe di apertura file*/
int  Analizza_Testo(char* File_Di_Testo);
```

Tutta la fase di analisi con le relative funzioni sono contenute nel file funzioni_analisi_file.c e funzioni_analisi_file.h

Acquisizione e stampa delle i-esime parole dei paragrafi

Lo schema principale per l'acquisizione e la stampa delle i-esime parole dei paragrafi e quello di leggere sequenzialmente il file e riconoscere i TAG dalle PAROLE. Una volta trovato il TAG di apertura <par> si procede col riconoscimento e il conteggio delle parole, arrivati alla parola di numero desiderato viene inserita in una lista ordinata alla fine essa viene stampata a video.

Dettagli implementativi

- Scansione sequenzialmente dei caratteri
- Riconoscimento dei caratteri (vengono selezionati solo caratteri alfanuerici e caratteri di TAG)
- Confezionamento di una Stringa valida (una stringa e' valida se e' una PAROLA o un TAG)
- Per ogni stringa valida controllo se e' un TAG <par> o <\par>
- Se <par> e aperto inizio a conteggiare
- Arrivato all' i-esimo carattere inserisco nella lista
- Trovato <\par> paragrafo di chiusura si ricomincia nuovamente
- Stampa a video del risultato

n.b. si e' pensato di eseguire tutti questi punti in una unica funzione. Infatti tale funzione ritorna una lista, la quale viene stampata a video.

Strutture dati fondamentali:

Le struttura dati fondamentale utilizzate sono le liste. Esse vengono spiegate in dettaglio nel paragrafo strutture dati fondamentali e routine ausiliarie.

Prototipi di Funzioni:

```
/*Ritorna 1 se Carattere e' uno dei caratteri della stringa Range 0 altrimenti*/
int Special_Char(char Carattere,char *Range)

/*Funzione che ritorna la I-esima parola dei paragrafi <par>
OUTPUT:ritorna il puntatore alla lista ordinata delle parole */
LIST * Crea_Lista(char* File_Di_Testo,int I)

/*Funzione che stampa un Elem di una Lista*/
void Print(void *Elem)
```

Tali funzioni sono contenute nel file funzioni_caric_parole.c e funzioni_caric_parole.h

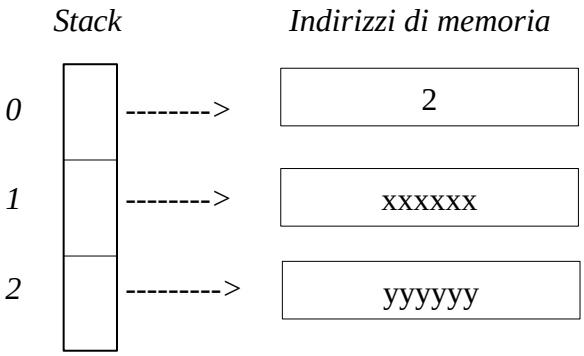
Unione, ordinamento e stampa delle i-esime e delle k-esime parole dei paragrafi (Opzionale)

Tale modulo e' un' estensione del modulo precedente.
Infatti viene richiamata nuovamente la funzione Crea_Lista con parametro k che rappresenta il numero di parole e richiamata la funzione merge tra le due liste ritornate da Crea_Lista.
Successivamente verra stampata a video.
Tale modulo viene implementato direttamente nel main.
Le funzioni sulle liste sono contenute nel paragrafo Strutture dati fondamentali e routine ausiliarie.

Strutture dati fondamentali e routine ausiliarie

Le struttura dati fondamentali in questo progetto sono lo stack e le liste.
Lo stack e' una struttura dati composta da due funzioni fondamentali: Push e Pop.
La struttura dati stack o pila e di tipo FILO (First IN Last OUT) cioe' il primo elemento che viene inserito e' l'ultimo che verra estratto.
L'inserimento viene fatto dalla funzione Push che inserisce un nuovo elemento nella testa mente l'estrazione viene fatto dalla funzione Pop che estrae l'elemento dalla testa.
dettagli orientati a C
In questo progetto questa struttura dati viene implementata con un array di puntatoti ad indirizzi. Dove il primo indirizzo punta ad un intero che rappresenta l'indice di testa e i restanti agli indirizzi di memoria degli elementi.
L'array e' allocato staticamente ed si possono inserire massimo 100 elementi, se viene superato la funzione di push dara' un errore.

Tali funzioni si trovano nel file stack.c le implementazioni e stack.h le definizioni.



In questo caso la testa sta nell'elemento 2 dell'array.

prototipi di funioni:

/*Quaesta funzione crea e inizializza uno stack ritorna un array di puntatori a void*/
void **CreaStack()

/*Questa funzione ritrona 1 se lo stack e' vuoto altrimenti zero*/
int IsEmpty(void ** Stack)

/*Questa funzione ritrona 1 se lo stack e' pieno altrimenti zero*/
int IsFull(void ** Stack)

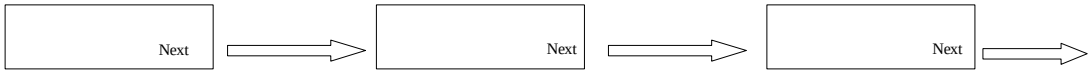
/*Funzione che inserisce un elemento sulla testa dello stack ritorna 1 se l'esito e' positivo 0 altrimenti.
INPUT puntatore allo Stack, puntatore all'emento da inserire*/
int Push(void ** Stack,void * Elem)

/*Funzione che estrae l'elemento di testa sullo stack ritorna l'indirizzo se e' positivo NULL altrimenti. INPUT puntatore allo Stack*/

```
void* Pop(void ** Stack);
/*Funzione che estrae senza eliminare l'elemento di testa sullo stack ritorna l'indirizzo se e' positivo NULL altrimenti.
INPUT puntatore allo Stack*/
void * ViewHead(void ** Stack);
```

L'altra struttura dati utilizzata sono le liste. Le liste sono strutture dati dinamiche cioe' che possono modificare la loro dimensione a runtime). In una lista si possono aggiungere o cancellare elementi, inoltre si possono aggiungere elementi in maniera ordinata.
Un nodo di una lista e' composta da riferimenti al prossimo nodo, e un campo che punta all'elemento.
dettagli orientati a C
In questo progetto questa struttura dati viene implementata con un struttura chiamata LIST che contiene il puntatore Next di tipo LIST, inoltre un puntatore void ad un Elem.

```
typedef struct List {
void *Elem;
struct List *Next;
} LIST;
```



Tale struttura e relative funzioni si trovano nel file list.c le implementazioni e list.h le definizioni.

prototipi di funioni:

```
/*Questa funzione iserisce un nuovo oggetto nella testa della lista e ritorna la nuova testa
se la lista e' vuota allora ritorna il singolo oggetto
INPUT: Lista in questione, Elem rappresenta il puntatore all'area di memoria dell'elemento
OUTPUT : nuova testa della lista*/
LIST *InsElem(LIST *,void *);

/*Questa funzione alloca e inizializza una struttura LIST
INPUT:   Inidirizzo dell'elemento
OUTPUT:  elemento allocato */
LIST *AllocaElem(void *);


/*Questa funzione iserisce un novo oggetto nella lista in ordine e ritorna la nuova testa
se la lista e' NULLA allora ritorna il singolo oggetto
INPUT : Testa della lista, Elem elemento da inserire , puntatore alla funzione di ordinamento*/
LIST *InsElemByOrder(LIST *,void *,int (*Confronto) (void *,void *));

/*Funzione di Confronto, Confronta gli elementi dell'indirizzo dgli elementi della lista ritorna <0 se a<b, 0 se a==b , >0
se a>b In questo caso la funzione confronta 2 stringhe*/
int conf(void * ,void * );
```

```
/*Questa funzione dealloca ricorsivamente una lista
INPUT Testa della Lista e funzione di deallocazione Elemento*/
void Dealloca_Ric(LIST *Lista,void (*Dealloca_Elem) (void *));

/*Funzionme ricorsiva che stampa a video dei valore della lista
INPUT: Lista da stampare, funzione che stampi l'elemento
Print prende in input l'elemento della lista*/
void Stampa_Ric(LIST *Lista,void (*Print) (void *));

/*Questa funzione fa il marge tra 2 liste senza aggiungere doppioni
alloca una nuova lista
INPUT: Lista 1 Lista2 funzione di confronto tra elementi di liste
OUTPUT: Lista compattata
n.b. le liste non devono essere deallocate*/
LIST *Merge(LIST *Lista1,LIST *Lista2,int (*Confronto) (void *,void *))
```

Funzione Principale

La funzione principale per le possibili scelte viene implementata con una struttura decisionale SWITCH CASE ripetuta in un WHILE.

Crea_Menu(INPUT: FILE file_di_menu OUTPUT: STRING Array, INTEGER elementi)
Stampa_Menu(INPUT: STRING Array, INTEGER elementi) //stampa a video il menu

Attendi un carattere di scelta da testiera
Scelta_menu=read_char(1234)

SCELTA DAL MENU:

1 Analizza_File(INPTU:Path_File OUTPUT: Integer risultato)
2 Lista1=Crea_Lista(INPUT:Path_File , integer i OUTPUT: LIST)
  Stampa_Ric(INPUT: Lista)
3 Lista2=Crea_Lista(INPUT:Path_File , integer k OUTPUT: LIST)
  ListaMarge=Marger(INPUT: Lista1, Lista2 OUTPUT: LIST)

File Essenziali:

funzioni_analizza_file.c
funzioni_analizza_file.h
funzioni_menu.c
funzioni_menu.h
funzioni_caric_parole.c
funzioni_caric_parole.h
list.c

list.h
stack.c
stack.h
tag.c
tag.h

Menu Contiene la struttura del menu

Raccomandazioni D uso:

Il file che contiene la struttura del Menu deve stare nella cartella dove viene eseguito il programma