**Analizzatore lessicale – carriera\_universitaria.fl**

Nella parte iniziale vi sono presenti tutte le librerie da importare all’interno del file C.   
Qui sotto segue una tabella con l’analisi dei token utilizzati dallo scanner:

|  |  |
| --- | --- |
| **anno\_int** | È una stringa, che corrisponde a “A.A.:20”. Dal momento che siamo nel XXI secolo, ho pensato inutile un controllo anche della cifra delle centinaia e delle migliaia, così li ho considerati come una semplice stringa. |
| **Matricola** | Un codice di 7 cifre che corrisponde alla matricola dell’alunno |
| **Codice\_materia** | Un codice di 5 cifre che corrisponde al codice della materia |
| **Nome\_minuscolo** | Un nome tutto in minuscolo |
| **Nome\_maiuscolo** | Un nome che comincia per lettera maiuscola |
| **anno\_crediti\_voto** | Un campo unico per identificare l’anno di corso, i crediti della materia e il voto conseguito. Il controllo sintattico verrà fatto lato bison, l’importante qui è riconoscere un numero di minimo 1 e massimo 2 cifre. |
| **“^^^^^”** | Prima stringa di separazione |
| **“\*\*\*\*\*”** | Seconda stringa di separazione |
| **\(** | Carattere che identifica la parentesi aperta |
| **\)** | Carattere che identifica la parentesi chiusa |
| **\”** | Carattere che identifica gli apici |
| **“->”** | Stringa che identifica la freccia |
| **\-** | Carattere che identifica il trattino |
| **\,** | Carattere che identifica la virgola |
| **\;** | Carattere che identifica il punto e virgola |
| **white** | Identifica gli spazi, le tablature e i new line |

Invece di utilizzare il singolo carattere fra singoli apici lato bison, ho preferito assegnare ad ogni carattere un token.  
La riga **. {return(\*yytext);}** serve a mandare allo scanner tutto quello che non ha riconosciuto così per come lo trova.

**Analizzatore sintattico – carriera\_universitaria.y**

**#define controlla\_data(a, b)  
#define controlla\_anno(a)  
#define controlla\_crediti(a)  
#define controlla\_voto(a)**Sono tutte e quattro delle Macro parametriche, le ho utilizzate al posto di creare delle funzioni nell’ultima sezione del file .y e di andarle a dichiarare. Controllano rispettivamente:

* Controlla\_data: se la data dell’anno accademico è corretta
* Controlla\_anno: se l’anno di corso è conforme alla descrizione del compito
* Controlla\_crediti: se il numero di crediti è uguale a dei dati parametri
* Controlla\_voto: se il voto conseguito è compreso fra 18 e 30

**%union{  
 int intero;  
 char\* stringa;  
 Lista\_materie\* esami;  
}**Utilizzato per definire i vari tipi sia nell’analizzatore lessicale che in quello sintattico. I tipi che mi sono serviti sono int, char\* e Lista\_materie\*

**void yyerror(char const \*s)**  
Funzione di stampa errore, utilizzata per stampare a schermo un messaggio di errore in caso di, appunto, errore.

**Input: Controllo\_parte\_1 SEP1 Controllo\_parte\_2 SEP2 Controllo\_parte\_3**  
Input è l’assioma della grammatica. Esso è diviso in 5 parti, dove due di esse sono i separatori, mentre le altre 3 sono le tre parti del file di input, definite come simboli non terminali.

**Controllo\_parte\_1: ANNO\_INT ANNO\_CREDITI\_VOTO ‘/’ ANNO\_CREDITI\_VOTO**  
Controlla la prima sezione del file di input, ovvero se l’anno accademico ha senso oppure no. Viene fatto un controllo delle ultime due cifre prima e dopo lo slash, per vedere se sono consecutive, e in questa parte del programma viene considerato valido un inserimento del tipo: “A.A.: 2099/00”.

**Controllo\_parte\_2: Materia Controllo\_parte\_2  
 | Materia**  
Controlla la seconda parte del file, e vede se si tratta di una lista di materie, ovvero una materia seguita da una lista di altre materie, e ha come caso base la presenza di una singola materia.

**Materia: PAR\_AP Contenuto PAR\_CH**  
Definisce com’è strutturata la materia, ossia un contenuto compreso fra una parentesi tonda aperta ed una chiusa.

**Contenuto: CODICE\_MATERIA VIRGOLA Insegnamento VIRGOLA ANNO\_CREDITI\_VOTO VIRGOLA ANNO\_CREDITI VOTO**  
Definisce il contenuto presente fra parentesi tonde. La sequenza è del tipo:  
codice\_materia, “Nome insegnamento”, anno\_di\_corso, crediti

**Insegnamento: APICI Nome\_materia APICI**  
Definisce che il nome dell’insegnamento è racchiuso fra apici

**Nome\_materia: NOME\_MAIUSCOLO Lista\_materie\_minuscolo  
Lista\_materie\_minuscolo: NOME\_MINUSCOLO Lista\_materie\_minuscolo  
 | NOME\_MINUSCOLO**  
Definisce il nome dell’insegnamento, ossia una parola che inizia per maiuscola seguita da tante parole tutte in minuscolo.   
ATTENZIONE: Questo significa che materie del tipo “Reti di Calcolatori” non verranno accettate, ma solamente materie del tipo “Reti di calcolatori”.

**Controllo\_parte\_3: Alunno Controllo\_parte\_3  
 | Alunno**  
Definisce la struttura della parte 3, ossia una lista di alunni con, come caso base, almeno un alunno.

**Alunno: MATRICOLA FRECCIA Nome\_e\_cognome PUNTO\_VIRGOLA ANNO\_CREDITI\_VOTO PUNTO\_VIRGOLA Lista\_esami PUNTO\_VIRGOLA**  
Definice la struttura del non-terminale Alunno come segue:  
Matricola -> Nome\_Cognome ; anno\_di\_corso ; lista\_di\_esami

**Nome\_e\_cognome: NOME\_MAIUSCOLO Nome\_e\_cognome  
 | NOME\_MAIUSCOLO**  
Definisce il campo Nome\_e\_Cognome come una sequenza di parole che iniziano per lettera maiuscola, con caso base almeno una parola che inizia per lettera maiuscola.

**Lista\_esami: CODICE\_MATERIA VIRGOLA ANNO\_CREDITI\_VOTO TRATTINO  
 | CODICE\_MATERIA VIRGOLA ANNO\_CREDITI\_VOTO TRATTINO Lista\_esami**  
Definisce la lista di esami di ogni studente come una sequenza del tipo:  
codice\_materia, voto –   
Questa sequenza va ripetuta più volte, a seconda di quanti esami ha sostenuto l’alunno.

**Tabella dei simboli – symbol\_table.h**

**struct Alunno{  
 char\* matricola;  
 int anno\_corso;  
 struct Lista\_materie \*esami;  
};**Struttura che definisce un Alunno, con la sua matricola, il suo anno di corso e la sua lista di esami svolti.

**struct Lista\_alunni{  
 struct Alunno alunno;  
 struct Lista\_alunni \*next;  
};**  
Struttura che definisce una lista di alunni, quindi con un campo alunno e un puntatore alla lista alunni successiva.

**struct Materia{  
 char\* codice;  
 int anno\_corso;  
 int crediti;  
 int voto;  
};**  
Struttura che definisce una materia, con il suo codice di corso, il suo anno di corso, i suoi crediti e con un campo voto. Quest’ultimo, se la materia si trova nella lista concatenata dell’elenco delle materie, sarà settato a zero. Se invece si troverà nella lista di esami di un alunno, allora avrà il valore del voto dell’alunno.

**struct Lista\_materie{  
 struct Materia materia;  
 struct Lista\_materie \*next;  
};**  
Struttura che definisce una lista di materie, quindi, come alunno, avrà un campo Materia e un puntatore alla lista di materie successiva.

**typedef struct Alunno Alunno;  
typedef struct Lista\_alunni Lista\_alunni;  
typedef struct Materia Materia;  
typedef struct Lista\_materie Lista\_materie;**  
Typedefs utilizzati per rinominare i tipi rimuovendo la keyword “struct”, il tutto per rendere la scrittura e la lettura un po’ più semplice.

**Lista\_materie\* materie;  
Lista\_alunni\* alunni;**  
Questa è la dichiarazione delle due liste concatenate, quella delle materie e quella degli alunni. La prima conterrà tutte le materie dichiarate nella seconda parte del file di input(ovviamente con campo voto settato a zero), mentre la seconda conterrà una lista di tutti gli alunni dichiarati nella terza parte del file.

**Alunno creaAlunno(char\*, int, Lista\_materie\*);  
Materia creaMateria(char\*, int, int, int);**  
Funzioni utilizzate per la creazione delle variabili Alunno e Materia.

**void aggiungiAlunno(Lista\_alunni \*\*, Alunno\*);  
void aggiungiMateria(Lista\_materie \*\*, Materia\*);**Funzioni che permettono di aggiungere una variabile di tipo alunno o materia alla propria lista concatenata. Qui ho preferito l’utilizzo di puntatori e doppi puntatoti per non dover restituire niente con la funzione.

**float mediaPonderata(Alunno\*);  
bool controllaAnno(const Alunno, const Materia);**  
Funzioni che servono per fare determinati controlli all’interno della stampa dell’alunno. Il primo calcola la media ponderata fra tutte le materie sostenute, la seconda controlla che l’anno dell’alunno sia possibile rispetto alle materie conseguite.

**void outputAlunno(Alunno\*);  
void outputMaterie(Lista\_materie \*);  
void outputMateria(Materia\*);  
void outputAlunni(Lista\_alunni \*);**  
Funzioni che permettono la stampa delle varie strutture definite precedentemente. Ai fini del progetto, servivano solo le funzioni outputAlunni e outputAlunno, a livello di debug ho deciso di avvalermi anche delle altre due funzioni.

**Lista\_materie\* concatena(Lista\_materie\*, Lista\_materie\*);**Funzione che permette la concatenazione di due liste di materie. All’interno di questa vado a definire un doppio puntatore, al quale verranno aggiungi, uno per uno, gli elementi della prima e della seconda lista. Una volta terminato questo processo, viene restituito il puntatore puntato dal doppio puntatore definito inizialmente (chiedo persono per il giro di parole in questa spiegazione).

**Materia prendiMateria(Lista\_materie\*\*, char\*, int);  
Materia cercaMateria(Lista\_materie\*\*, char\*);**  
Queste due funzioni servono per poter inserire una determinata materia nella lista di esami personale dello specifico alunno. La funzione prendiMateria richiama la funzione cercaMaterie, la quale cercherà la materia nell’elenco totale delle materie, avendo a disposizione il codice materia. Questa restituirà la materia trovata, dove verrà inserito il voto conseguito dallo studente. Infine verrà restituita la variabile di tipo Materia, con il voto dell’alunno specifico.