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.