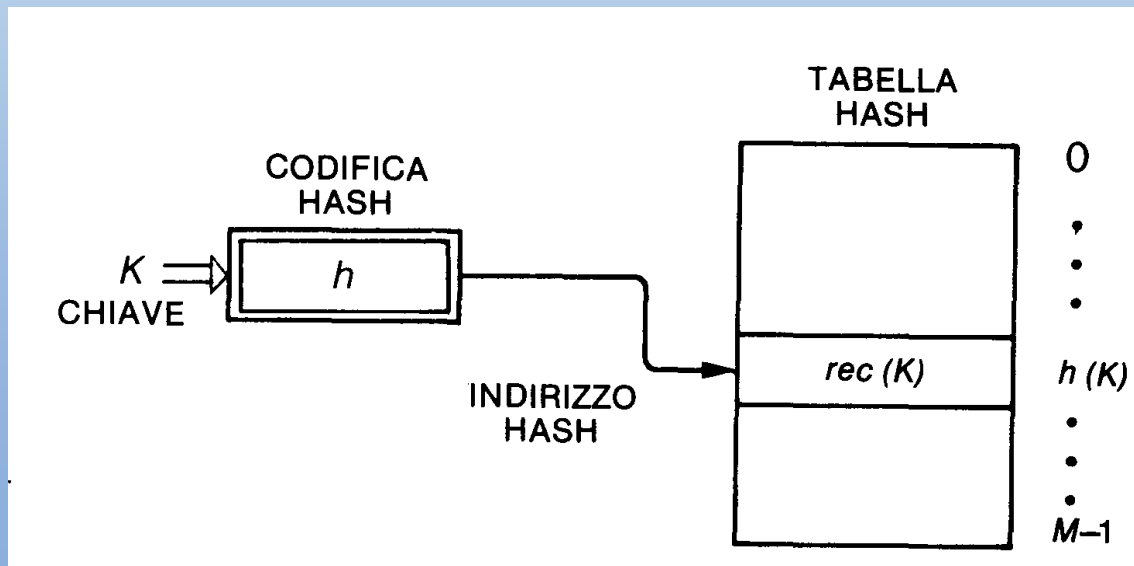


TABELLA DEI SIMBOLI

LA TABELLA DEI SIMBOLI COME TABELLA HASH

Ricordiamo che la struttura dati più efficiente per rappresentare una tabella dei simboli è la hash table.



Occorre avere una funzione hash che agisce sulle chiavi e le trasforma in un indirizzo della tabella. In questo indirizzo verrà conservata la chiave

COLLISIONI

Inoltre la tabella hash deve tenere conto delle collisioni, ossia del fatto che se la funzione hash non è iniettiva, due elementi con chiavi diverse possono collidere nella stessa casella.

E' per questo che, in particolare per le hash table utilizzate dai compilatori, si sceglie di utilizzare le hash table con liste di collisione

HASH TABLE CON LISTE DI COLLISIONE

La struttura dati è costruita in maniera tale da associare ad **ogni cella** della hash table una **lista di tutti gli elementi che collidono su quella cella**.

Per verificare se una chiave è presente nel dizionario occorre calcolare la sua funzione hash e cercare la chiave nella lista corrispondente.

A lunghezza media di una lista di collisione sarà pari al fattore di carico

$$A=n/m$$

Assumendo l'uniformità semplice per la funzione hash, il tempo di ricerca sarà in media

$$T_{avg}(n,m)=O(1+n/m)^2$$

SCELTA DI M E FATTORE DI CARICO

Ricordiamo che se n è il numero variabile degli elementi e m è la dimensione fissata del vettore, il fattore di carico è $A=n/m$

Spesso, secondo quali sono le chiavi degli elementi da inserire, m si sceglie o come potenza di due oppure come numero primo. Ciò dipende dalla funzione hash che definiamo in funzione delle chiavi.

ESEMPI DI FUNZIONI HASH (HASH TABLE DI TAGLIA M)

- Se le chiavi k sono numeri reali casuali in $[0,1)$, una funzione hash può essere per esempio

$$h(k) = \lfloor km \rfloor \text{ (parte intera inferiore del prodotto di } k \text{ per } m)$$

- Se le chiavi k sono numeri reali casuali in $[s,t)$, allora

$$h(k) = \lfloor (k-s)/(t-s) * m \rfloor$$

- Se le chiavi k sono intere,

$$h(k) = k \% m \text{ (hashing modulare).}$$

Oss: per l'hashing modulare, se $m=10^p$, $h(k)$ rappresenta le p cifre meno significative. Una buona scelta sarebbe m un numero primo.

ESEMPI DI FUNZIONI HASH (HASH TABLE DI TAGLIA M)

Se le chiavi sono **stringhe**, si cerca di convertirle in un numero naturale interpretando i simboli come numeri in sistemi di numerazione in base diversa.

Per convertire una stringa in un numero naturale si associa ad ogni simbolo della stringa un numero tra 1 e 128, la sua codifica ASCII per esempio.

In base alla sua posizione enumerata dall'ultima alla prima, partendo da zero, ogni numero verrà moltiplicato per 128 elevato alla sua posizione, ossia si considera il numero espresso in un sistema posizionale in base 128.

ESEMPIO

Per convertire la stringa "casa" in base 128 si ha:

$c a s a_{(128)} =$

$c_3 a_2 s_1 a_0 = (99)_3(97)_2(115)_1(97)_0$ codifica ASCII di ogni carattere

$$99 * 128^3 + 97 * 128^2 + 115 * 128^1 + 97 * 128^0 =$$

$$99 * 2097152 + 97 * 16384 + 115 * 128 + 97 * = 209222113$$

ESEMPI DI FUNZIONI HASH (HASH TABLE DI TAGLIA M)

La funzione hash corrispondente può essere quindi definita in C come:

```
int hash(char *s) {  
    /* calcola il valore HASH di s;*/  
    int h=0,a=127;  
    for(;*s!='\0';s++)  
        h=(a*h+*s)% HASHSIZE; /*HASHSIZE var globale, deve essere un primo*/  
    return h;}  

```

Ogni carattere contribuisce in modo unico al risultato finale. Per avere una buona (cioè uniforme) distribuzione degli oggetti nella tabella (e avere collisioni distribuite) è conveniente utilizzare dei numeri primi per la base delle potenze (127 lo è) e per la dimensione delle tabelle.

METODI CLASSICI DI RISOLUZIONE DELLE COLLISIONI

Liste di collisione (hash table with chaining): gli elementi collidenti sono contenuti in liste esterne alla tabella; $t[i]$ punta alla lista di elementi tali che $h(k) = i$

Indirizzamento aperto: tutti gli elementi sono contenuti nella tabella; se una cella è occupata, se ne cerca un'altra libera

HASH TABLE WITH CHAINING

- La probabilità che il numero di chiavi in ciascuna lista sia pari al fattore di carico α è molto vicina a 1.
- **Costo della ricerca (caso medio):** nell'ipotesi di **uniformità semplice** della funzione hash, una ricerca richiede in media un tempo $\theta(1+\alpha)$

ESERCITAZIONE

Si realizzi, utilizzando i programmi flex e bison, una calcolatrice evoluta sui numeri reali integrata con variabili e funzioni predefinite. Oltre a verificare la correttezza sintattica dell'input, il programma dovrà fornire il risultato richiesto nell'input.

Un'espressione è sia un'espressione aritmetica standard che modificata. Gli operatori e le funzioni da implementare sono:

- Gli operatori aritmetici standard: `+`, `-`, `*`, `/` e l'operatore di negazione (`-` unario);
- Gli operatori di assegnazione: `=`, `+=`, `-=`, `*=`, `/=`
- Le funzioni usuali : `log(x)` , `sin(x)` , `cos(x)` , `tan(x)` , `pow(x,y)` , `sqrt(x)` della libreria `<math.h>`, dove `x` è valore numerico.

Le regole di priorità e di associatività di questi operatori sono quelli del linguaggio c.

Con l'aiuto di questi operatori, si deve definire una grammatica che realizzi e valuti tutte le espressioni come quelle comunemente usate, come per esempio :

`alpha=10.52`

`beta += c+log(23*4+0.5)-7.7`

La calcolatrice sarà in grado di restituire un valore solo se le variabili coinvolte nella parte destra dell'assegnazione sono state inizializzate. La calcolatrice è dotata di istruzioni del tipo `print a`, dove `a` è una variabile, che prevedono che la calcolatrice restituisca in output il valore della variabile `a`. Il risultato dato in output è approssimato alla seconda cifra decimale nel caso di variabili floating point.


Le **variabili** sono definite come una **successione di lettere minuscole**. Inoltre possono essere **intere** o **floating point** in base al valore ad esse assegnato.

I **nomi delle funzioni sono considerate parole riservate**. Non potranno dunque in alcun modo essere usate come variabili.

Scanner

```
%{
#include <string.h>
#include <stdlib.h>
#include "calc.tab.h"
}%
%option noyywrap
delim      [ \t]
ws         {delim}+
digit      [0-9]
number     {digit}+(\.{digit}+)
var        [a-z]+
%%
"+"        return('+');
"-"        return('-');
"*"        return('*');
"/"        return('/');
"="        return('=');
"+="       return(ADD_ASSIGN);
"-="       return(SUB_ASSIGN);
"*="       return(MUL_ASSIGN);
"/="       return(DIV_ASSIGN);
```

```
"("        return('(');
")"        return(')');
";"        return(';');
", "       return(',');
"print"    return(PRINT);
"log"      return(LOG);
"sin"      return(SIN);
"cos"      return(COS);
"tan"      return(TAN);
"pow"      return(POW);
"sqrt"     return(SQRT);
{ws}       ;
{digit}+   {yylval.int_t=atoi(yytext);
            return(NUM_INT);}
{number}   {yylval.double_t=atof(yytext);
            return(NUM_REAL);}
{var}      {yylval.char_t=strdup(yytext);
            return(VAR);}
\n         ;
%%
```



In caso di intero, reale o variabile, viene assegnato a `yylval` il valore corrispondente al lessema riconosciuto e viene assegnato il nome del token

PARSER

```
%{
    #include <math.h>
    #include <stdio.h>
    #include "symb_tab.h"
    void yyerror (char const *);
    double tmp;
}%
%union {
    char      *char_t;
    double     double_t;
    int        int_t;
}
%error-verbose
%token <double_t>NUM_REAL
%token <int_t>NUM_INT
%token <char_t>VAR
%token ADD_ASSIGN SUB_ASSIGN MUL_ASSIGN DIV_ASSIGN
%token PRINT LOG SIN COS TAN POW SQRT
%left '+' '-'
%left '*' '/'
%left NEG
%type <double_t>right
%type <char_t>left
%start instr_list
%%
```

I token e i non terminali possono essere di tipo int, double o char

Vengono assegnati i tipi corrispondenti al significato dei token. Si usano nomi diversi ai token per tipi di dato diversi

Assegna i tipi ai non terminali right e left che denotano l'operando destro e l'operando sinistro

ogni volta che trova una nuova variabile inserisce il token nella tabella dei simboli

Il risultato (il valore della variabile) si trova nella locazione lookup(\$2) ->value

```
instr_list: instr_list instr
           | instr
           ;
```

```
instr      : assg ';'
           | PRINT VAR ';'
           ;
```

```
{ins_token($2);
 printf("\nResult: %.2f\n", lookup($2)->value);}
```

```
assg       : left '=' right
           | left ADD_ASSIGN right
           | left SUB_ASSIGN right
           | left MUL_ASSIGN right
           | left DIV_ASSIGN right
           ;
```

```
{ins_attr($1, $3);}
{tmp=lookup($1)->value+$3; ins_attr($1, tmp);}
{tmp=lookup($1)->value-$3; ins_attr($1, tmp);}
{tmp=lookup($1)->value*$3; ins_attr($1, tmp);}
{tmp=lookup($1)->value/$3; ins_attr($1, tmp);}
```

```
left       : VAR
           ;
```

```
 {$$=$1; ins_token($1);}
```

Si inserisce il token della nuova variabile

Si inserisce il valore dell'attributo della variabile a destra (right) nella locazione della variabile a sinistra (left)

Si inserisce il risultato in una variabile temporanea e si inserisce il valore ottenuto nella locazione in cui si trova la variabile di sinistra


```

right : NUM_REAL          { $$=$1 }
      | NUM_INT           { $$=$1 }
      | VAR               { $$=lookup($1)->value }
      | right '+' right   { $$=$1+$3 }
      | right '-' right   { $$=$1-$3 }
      | right '*' right   { $$=$1*$3 }
      | right '/' right   { $$=$1/$3 }
      | '-' right %prec NEG { $$=-$2 }
      | '(' right ')'     { $$=$2 }
      | LOG '(' right ')' { $$=log($3) }
      | SIN '(' right ')' { $$=sin($3) }
      | COS '(' right ')' { $$=cos($3) }
      | TAN '(' right ')' { $$=tan($3) }
      | SQRT '(' right ')' { $$=sqrt($3) }
      | POW '(' right ',' right ')' { $$=pow($3, $5); }
      ;

```

```
%%
```

```

void yyerror (char const *s) {
    fprintf(stderr, "%s\n", s);
}

int main() {
    yyparse();
    print_sym_table();
    return 0;
}r

```

Secondi il tipo di istruzione di assegnazione si svolgono le operazioni opportune. Se a destra dell'assegnazione c'è una variabile, occorre fare un lookup nella tabella dei simboli

La regola per la negazione include %prec NEG. L'unico operatore in questa regola è -, che ha bassa precedenza, ma vogliamo il meno unario abbia una precedenza più alta della moltiplicazione. Il %prec dice a bison di usare la precedenza di NEG per questa regola, stabilita in fase di definizione dei token.

Stampa la tabella dei simboli

HEADER FILE PER LA HASH TABLE (symb_tab.h)

```
#define HASHSIZE 101
```

definizione della costante lunghezza
della hashtable(numero primo)

```
struct nlist {  
    struct nlist *next;  
    char *name;  
    double value;  
};
```

Definizione della struttura della
Symbol Table

```
struct nlist *hashtab[HASHSIZE];
```

definizione della tabella hash

```
unsigned int hash(char *s);
```

definizione della funzione di hash

```
struct nlist *lookup(char *s);
```

definizione funzione di lookup

```
struct nlist *ins_attr(char *s, double i);
```

definizione funzione di inserimento attributo

```
struct nlist *ins_token(char *name) ;
```

definizione funzione inserimento token

Implementazione tabella dei simboli (symb_tab.c)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "symb_tab.h"
```

```
unsigned int hash(char *s) {
    int h=0;
    for (;*s!='\0';s++)
        h=(127*h+*s)%HASHSIZE;
    return h;
}
```

calcola il valore HASH di s

```
struct nlist *lookup(char *s) {
    struct nlist *np;
    for (np = hashtab[hash(s)]; np != NULL; np = np->next)
        if (strcmp(s,np->name) == 0)
            return np;
    return NULL;
}
```

cerca l'elemento di chiave s nella tabella HASH

restituisce zero se le
stringhe sono identiche

```
struct nlist *ins_attr(char *s,double i) {  
    struct nlist *np;  
    np=lookup(s);  
    if(np!=NULL){  
        np->value=i;  
        return np;  
    }  
    return NULL;  
}
```

cerca s nella tabella HASH e
inserisce l'attributo i

```
struct nlist *ins_token(char *name) {  
    struct nlist *np;  
    unsigned hashval;  
    if ((np = lookup(name)) == NULL) {  
        if (np = (struct nlist *)malloc(sizeof(*np)))==NULL)  
            return NULL;  
        np->name = strdup(name);  
        np->value=0;  
        hashval = hash(name);  
        np->next = hashtab[hashval];  
        hashtab[hashval] = np; }  
    return np;  
}
```

inserisce un nuovo
elemento nella
tabella dei simboli

se non è già presente

inserisce la struttura in testa
alla lista

```
void print_sym_table() {  
    int key;  
    struct nlist *np;  
    for (key=0; key<HASHSIZE; key++) {  
        printf("{%d}: ", key);  
        for (np = hashtab[key]; np != NULL; np=np->next)  
            printf("[%f] [%s]", np->value, np->name);  
        printf("\n");  
    }  
}
```

funzione che stampa la
tabella dei simboli

PROGETTO - LINEE GUIDA

Il progetto descrive anzitutto come deve essere fatto il file d'ingresso per essere accettato dal **parser**.

Nella prima parte del progetto occorrerà quindi costruire con flex un analizzatore lessicale che permette di riconoscere e classificare con dei token i lessemi del testo.

Quindi occorrerà comprendere quali sono le regole sintattiche che permettono di riconoscere la forma corretta di un testo. Occorre quindi costruire la grammatica generatrice.

Conoscendo la grammatica è possibile costruire il **parser** mediante bison, dentro il quale, oltre alle regole sintattiche per il riconoscimento, ci saranno operazioni di tipo semantico e/o delle istruzioni per svolgere operazioni di ricerca e di inserimento di token e valori di attributi nella symbol table.

PROGETTO

Occorrerà quindi costruire la (o le) tabella/e dei simboli. Definire le strutture base e implementare le operazioni di ricerca e inserimento valori.

In questo file è possibile anche inserire altre funzioni che devono operare sui dati, in base a quanto richiesto dal testo.

I file `lex.yy.c`, `nomefile.tab.c`, `symb_tab.c` andranno compilati insieme

```
gcc lex.yy.c nomefile.tab.c symb_tab.c
```

PROGETTO - ESEMPIO

VALUTAZIONE RIVISTE

Si costruisca, utilizzando la coppia di programmi FLEX e BISON, un traduttore guidato dalla sintassi in grado di riconoscere un linguaggio che descrive alcune **riviste scientifiche** ed attribuisce loro una **valutazione**.

Il programma deve esaminare un file in input scritto con un linguaggio descritto di seguito. Tale linguaggio consente di definire una serie, univocamente determinata, di **categorie di riviste** sulla base della tipologia di argomenti trattati nella rivista stessa; per ognuna è possibile definire **un insieme di attributi** che si applicano a tale categoria ed assegnare agli attributi **un peso**, utile per la valutazione complessiva di una rivista.

Dopo la definizione delle categorie di riviste possono comparire le descrizioni delle riviste che assegnano una valorizzazione agli attributi.

Il file in input è suddiviso in due sezioni, separate da \$\$.

La prima sezione contiene le definizioni delle categorie. Ciascuna definizione di categoria è costituita dalla lista degli attributi, seguita da una freccia ("->") e dal nome della categoria di rivista.

La lista degli attributi è racchiusa tra parentesi tonde e consiste di una sequenza di definizioni di attributi, separate da virgole (','). Le definizioni degli attributi sono formate dal nome dell'attributo, seguito da un due punti (':') e dal peso assegnato a tale attributo. Il peso è un numero intero senza segno.

I nomi delle categorie e degli attributi sono parole costituite da caratteri minuscoli. Per esempio,

(esperimenti : 10 , ricerca : 8, algoritmi: 2) -> biologia

Le varie definizioni di categorie che possono comparire non sono separate da alcun carattere, ma si trovano su righe diverse.

La **seconda sezione** contiene le **descrizioni delle riviste**.

In tale sezione possono comparire varie descrizioni di riviste; ogni descrizione di rivista è formata come segue:

nome categoria : giudizi = nome della rivista ;

Il **nome della categoria** deve essere uno di quelli definiti in precedenza.

I **giudizi** sono costituiti da una sequenza di valorizzazioni separate da virgole (',') relativamente ad alcuni degli attributi della categoria.

Il **nome della rivista** è rappresentato da una frase. Una frase è costituita da un insieme di parole separate da caratteri di spaziatura (non importa quanti). Le parole sono sequenze di caratteri alfabetici, maiuscoli o minuscoli, consecutivi.

Una **valorizzazione** è formata da un valore, descritto con un intero (3, 2, 1, 0), il cui significato è descritto nella tabella seguente:

Significato Valore

Eccellente 3

Buono 2

Sufficiente 1

Scadente 0

Per esempio, una descrizione di una rivista potrebbe essere:

Biologia: 3 esperimenti, 2 ricerca = Bioinformatics;

Come risulta evidente dall'esempio, i **nomi degli attributi** devono essere quelli **definiti per la categoria della rivista**. Non è necessario che nella descrizione di una rivista compaiano per forza tutti gli attributi definiti per la sua categoria. Se un attributo non compare nella descrizione si assume, per default, che gli sia stato attribuito il giudizio di sufficiente, cioè 1.

Il programma deve essere in grado di riconoscere ed ignorare commenti che iniziano con la coppia di caratteri "//" e terminano dal ritorno a capo

Scopo del programma.

Il programma deve **analizzare un testo in input** scritto nel linguaggio descritto precedentemente e, terminata l'analisi, **deve fornire in output, suddivise per categoria, un elenco delle riviste con il loro punteggio.**

Il **punteggio** di una rivista si calcola nel modo seguente. Per ogni attributo della categoria si moltiplica il **peso** dell'attributo **per il valore** assegnato nella descrizione della rivista.

Il **punteggio della rivista** è dato dalla **somma dei valori ricavati in tal modo**. Più precisamente: siano p_i il peso dell' i -esimo attributo, v_i il valore assegnato a tale attributo nella categoria di rivista in esame e sia N il numero di attributi:

$$\text{punteggio della rivista} = \sum_{i=1}^N p_i v_i$$

Quando il programma incontra un errore, sia sintattico che semantico, deve segnalarlo e terminare l'esecuzione: non è richiesta la gestione ed il recupero degli errori.

Esempio di input

```
// Definizione delle categorie di riviste:  
( esperimenti : 10 , ricerca : 8, algoritmi: 2 ) -> biologia  
( tecnologia : 5, algoritmi:10, ricerca: 7) -> informatica  
$$  
// Descrizione delle riviste:  
informatica: 3 algoritmi, 1 tecnologia = Theoretical Computer Science;  
biologia: 3 esperimenti, 2 ricerca = Bioinformatics;  
informatica: 3 algoritmi, 2 tecnologia, 3 ricerca = Information and Computation;
```

Esempio di output

```
Punteggi raggiunti:  
Theoretical Computer Science, 42  
Bioinformatics, 46  
Information and Computation, 61
```

SCANNER

```
%{
    #include "riv.tab.h"
    #include <string.h>
}%
%option noyywrap
ws      [ \t\n]+
%%
{ws}+      ;
"$ $"      {return (SEP) ;}
"//" ".*\n" ;
0|[1-9][0-9]*      {yylval.intero=atoi(yytext) ; return (P_V) ;}
[A-Z][A-Za-z ]*    {yylval.stringa=strdup(yytext) ; return (NOM) ;}
[a-z]+            {yylval.stringa=strdup(yytext) ; return (ATT_CAT) ;}
"->"            {return (ARR) ;}
":"             {return (DPN) ;}
";"            {return (PVIR) ;}
","           {return (VIR) ;}
"("           {return (OPENP) ;}
")"          {return (CLOSEP) ;}
"="          {return (EQ) ;}
%%
```

PARSER

```
%{
    #include <stdio.h>
    #include "symb_tab.h"
    int yylex();
    void yyerror (char const *);
}%
%union {
    char *stringa;
    int intero;
}
%error-verbose
%token <stringa> NOM ATT_CAT
%token <intero> P_V
%token SEP ARR DPN PVIR VIR
OPENP CLOSEP EQ
%start Input
%%
```

PARSER

Input: Sezione1 SEP Sezione2

;
Sezione1 : El_cat

;
El_cat : Categoria
| El_cat Categoria
;

Categoria: OPENP El_attr CLOSEP ARR ATT_CAT {ins_cat(\$5);}

;
El_attr : Attr VIR El_attr
| Attr
;

Attr : ATT_CAT DPN P_V {ins_attr(\$1,\$3);}

;
Sezione2 : El_riv
;

Nome categoria

Peso dell'attributo

PARSER

```
El_riv      : Rivista
             | El_riv Rivista
             ;
Rivista     : ATT_CAT DPN El_attr2 EQ NOM PVIR {ins_riv($5,$1);}
             ;
El_attr2:   Attr2 VIR El_attr2
             | Attr2
             ;
Attr2      : P_V ATT_CAT {ins_attr($2,$1);}
             ;

%%

void yyerror (char const *s) {
    fprintf(stderr, "%s\n", s);
}

int main() {
    if(yparse()==0) {
        print_res();
    }
    return 0;
}
```

Nome rivista

HEADER FILE PER LA SYMBOL TABLE

```
#define HASHSIZE 101

//struttura degli attributi
typedef struct a {
    char *nome;
    unsigned int pv;
    struct a *next;
}attributo;

/*struttura per categorie
di riviste*/
typedef struct c {
    char *nome;
    attributo *att_1;
    struct c *next;
}categoria;
```

```
//struttura per riviste
typedef struct r {
    char *nome;
    attributo *att_1;
    categoria *c;
    unsigned int voto;
    struct r *next;
}rivista;

//hashtable per categorie
//categoria *hashtable[HASHSIZE]; funzioni.

unsigned int hash(char *s);
categoria *lookupc(char *s);
attributo *ins_attr(char *nome, int pv);
categoria *ins_cat(char *nome);
rivista *ins_riv(char *nome, char *categoria);
unsigned int calc_voto(attributo *r, attributo *c);
unsigned int voto_pesato(attributo *r, attributo *c);
void print_res();
```

HASHTABLE

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "symb_tab.h"

attributo *att_list=NULL;//lista di attributi da collegare a riviste o categorie
rivista *riv_list=NULL; //lista concatenata per memorizzare le riviste

unsigned int hash(char *s) {
    int h=0;
    for(;*s!='\0';s++)
        h=(127*h+*s)%HASHSIZE;
    return h;
}

categoria *lookupc(char *s) {
    categoria *c;
    for (c = hashtable[hash(s)]; c != NULL;c = c->next)
        if (strcmp(s,c->nome) == 0)
            return c;
    return NULL;
}
```

→ calcola il valore HASH di s

→ ricerca di s nella tabella HASH delle categorie

```
attributo *ins_attr(char *nome, int pv) {  
    attributo *a;  
    a = (attributo *)malloc(sizeof(*a));  
    a->nome = strdup(nome);  
    a->pv = pv;  
    a->next = att_list; //operazione di inserimento in testa alla lista  
    att_list = a;  
    return a;  
}
```

inserimento di un attributo nella lista di attributi riferita all'attuale categoria o rivista

```
categoria *ins_cat(char *nome) {  
    categoria *c;  
    unsigned hashval;  
    if ((c = lookupc(nome)) == NULL) { /*se non è già presente*/  
        if ((c = (categoria *)malloc(sizeof(*c))) == NULL)  
            return NULL;  
        c->nome = strdup(nome)  
        c->att_l = att_list;  
        att_list = NULL;  
        hashval = hash(nome);  
        c->next = hashtable[hashval];  
        hashtable[hashval] = c;  
    }  
    return c;}  
}
```

Inserimento di una categoria nella lista delle categorie. La lista degli attributi con relativi valori è ottenuta al passo precedente e contenuta nella variabile globale att_list

la struttura viene riinizializzata a null per essere riutilizzata per altre categorie o riviste

Viene inserita in testa alla lista di pertinenza

```
rivista *ins_riv(char *nome, char *categoria) {  
    rivista *r;  
    unsigned hashval;  
    if ((r = (rivista *)malloc(sizeof(*r)))==NULL)  
        return NULL;  
    r->nome = strdup(nome);  
    r->att_l = att_list;  
    att_list = NULL;  
    r->c = lookupc(categoria);  
    r->voto = calc_voto(r->att_l, r->c->att_l);  
    r->next = riv_list;  
    riv_list = r;  
    return r;  
}
```

inserimento rivista nella lista delle riviste

La lista di attributi contenuta nella variabile globale att_list viene inserita nell'attuale rivista esaminata

viene assegnata una categoria alla rivista tramite lookup sull'hashtable

viene calcolato il voto utilizzando le informazioni nella lista di attributi della rivista e della categoria di appartenenza

operazioni di inserimento in lista

```

unsigned int calc_voto(attributo *r, attributo *c) {
    if(c->next == NULL)
        return voto_pesato(r,c);
    return voto_pesato(r,c) + calc_voto(r,c->next);
}

```

```

unsigned int voto_pesato(attributo *r, attributo*c) {
    if(r==NULL)
        return c->pv;//peso moltiplicato per 1
    if(!strcmp(r->nome,c->nome))
        return c->pv * r->pv;
    return voto_pesato(r->next,c);
}

```

funzione ricorsiva che restituisce il voto complessivo della rivista (r). La funzione restituisce il voto pesato del primo elemento della lista (calcolato con un'opportuna funzione d'appoggio) più il calcolo del voto della rivista relativamente a tutti i successivi elementi della lista, appoggiandosi al valore della funzione voto_pesato

funzione ricorsiva che restituisce il voto pesato per ogni attributo presente tra quelli della categoria corrispondente. Restituisce il peso*1 se non trova attributi corrispondenti tra quelli della rivista, il peso*voto altrimenti

```
void print_res() {  
    rivista *first = riv_list;  
    printf("Punteggi raggiunti:\n");  
    while(riv_list!=NULL) {  
        printf("%s, %d\n",riv_list->nome,riv_list->voto);  
        riv_list=riv_list->next;  
    }  
    riv_list = first;  
}
```

semplice funzione di stampa
con visita per ogni elemento
nella lista di riviste