Esempio di costruzione di una classe: un dizionario

Problema: si vuole disporre di un dizionario per tradurre parole dall'italiano in un'altra lingua e viceversa. Il dizionario deve consentire di ottenere la traduzione di una parola italiana in un'altra lingua e anche data una parola nella lingua straniera deve restituire la parola italiana corrispondente. Poichè le parole cambiano significato nel tempo bisogna poter cambiare la traduzione di una parola.

Requisiti del Dizionario:

1. **I dizionari hanno una dimensione massima, che viene specificata al momento della loro costruzione.**
2. **Le parole sono fornite e restituite al dizionario come stringhe di caratteri C.**
3. **Bisogna poter ottenere la dimensione corrente del dizionario.**
4. **Ogni parola italiana ha una sola traduzione nella lingua straniera scelta e viceversa.**
5. **Si deve poter modificare la traduzione di una parola, ma non aggiungerne un'altra, visto il vincolo precedente.**
6. **La ricerca nel dizionario può essere implementata nel modo più semplice, anche se non molto efficiente.**

Esempio di costruzione di una classe: il primo passo

La scelta del nome:

Diz ma rischiamo di non ricostruire il significato

Trad stessa obiezione

Dizionario in fondo non è troppo lungo e certo è ben descrittivo!

Esempio di costruzione di una classe: il nome

```
/* Nome del file: Dizionario.h
   Specificatione della classe
   dizionario
*/
/*nome della classe*/
typedef struct dizionario *DizionarioP;
```

Esempio di costruzione di una classe: dichiarazione dei metodi

Passo 1: I costruttori

Poiché nel primo requisito si afferma che i dizionari hanno una dimensione massima, che viene specificata al momento della loro costruzione, scriviamo il prototipo del costruttore in modo che prenda la dimensione massima come parametro di input.

```
/* Costruttore*/
DizionarioP CostDiz( int numMaxPar );
/* Costruisce un nuovo dizionario che può contenere fino a dimMax parole
   Pre-cond: numMaxPar > 0
   Postc: restituisce un puntatore a un dizionario o
   NULL se non c'è abbastanza memoria
*/
```

Esempio di costruzione di una classe: dichiarazione dei metodi

Passo 2: I distruttori

```
/* Distruttore*/
void CancDiz( DizionarioP d );
/* Cancella il dizionario
Pre-cond: d != NULL
Postc: libera la memoria allocata per d  */
```

Esempio di costruzione di una classe: dichiarazione dei metodi

Passo 3: I selettori di attributi

Dal requisito 3 si ha che bisogna poter ottenere la dimensione corrente del dizionario

```
int Contaparole( DizionarioP d );
/* Prec: d != NULL
   Postc: Restituisce il numero delle parole nel
   dizionario */
```

Esempio di costruzione di una classe: dichiarazione dei metodi

Poichè "il dizionario deve consentire di ottenere la traduzione di una parola italiana in un'altra lingua e anche data una parola nella lingua straniera deve restituire la parola italiana corrispondente". e in base al requisito "2. Le parole sono fornite e restituite al dizionario come stringhe di caratteri C." si introducono le funzioni:

```
char *Trad_da_It( DizionarioP d, char *stringa );
/* dà la traduzione nella lingua straniera della parola italiana
   in stringa
  Prec: d != NULL && stringa != NULL && strlen( stringa ) > 0
  Postc: restituisce la traduzione dall' italiano di stringa o
   NULL se non la trova */
```

Esempio di costruzione di una classe: dichiarazione dei metodi

e

```
char *Trad_da_Stran( DizionarioP d, char *stringa );
/*dà la traduzione in italiano della  parola della lingua
    straniera in stringa
  Prec: d != NULL && stringa != NULL && strlen( stringa ) > 0
  Postc: restituisce la traduzione dalla lingua straniera di
    stringa o NULL se non la trova */
```

Passo 4: Le proprietà.

```
int AggTrad( DizionarioP d, char *par1, char *par2 );
/* aggiunge una nuova traduzione nel dizionario d
della parola italiana in par1 con la parola straniera par2,
   par1 e par2 non devono essere già presenti in d
   Prec: d != NULL && par1 != NULL && strlen( par1 ) > 0 &&
            par2 != NULL && strlen( par2 ) > 0 &&
            Trad_da_It( d, par1 ) == NULL &&
            Trad_da_Stran( d, par2 ) == NULL
    Postc: ContaParole( d ) = n+1 se   ContaParole( d ) = n prima
              dell'aggiunta && strcmp( Trad_da_It( d, par1 ), par2 ) == 0
              && strcmp( Trad_da_Stran( d, par2 ), par1 ) == 0 */
```

Rispetta il requisito
4 Ogni parola italiana ha una sola traduzione nella lingua straniera scelta e viceversa.

Metodi per l'aggiornamento del dizionario

```
int SostTrad( DizionarioP d, char *par1, char *par2 );
/* Sostituisce la traduzione presente della parola italiana
in par1 con la parola straniera par2.
   par1 deve essere presente nel dizionario
   par2 non deve essere già presente nel dizionario
   Prec: d != NULL &&
            par1 != NULL && strlen( par1 ) > 0 &&
            par2 != NULL && strlen( par2 ) > 0 &&
            Trad_da_It( d, par1 ) != NULL &&
            Trad_da_Stran( d, par2 ) == NULL
   Postc: ContaParole( d ) = n se   ContaParole( d ) =n prima
             dell'aggiunta &&
             strcmp(Trad_da_It ( d, par1 ), par2 ) == 0 &&
             strcmp (Trad_da_Stran ( d, par2 ), par1 ) == 0
*/
```

L'implementazione (incompleta!):Dizionario.c

```
#include <assert.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "Dizionario.h"


#define PAROLA_VALIDA(w)  assert(w!=NULL);assert(strlen(w)>0);

 typedef  char* string;

 struct dizionario {
   int num;
   int numMax;
   string * Italiano;
   string * Straniera;
   };
```

Il costruttore

```
DizionarioP CostDiz( int dimMax )
{DizionarioP d;
int i;
assert( dimMax> 0 );
d = (DizionarioP)malloc( sizeof( struct dizionario ) );
if ( d != NULL )
          { d -> num = 0;
          d -> numMax = dimMax;
          d -> Italiano =  (string*) malloc( dimMax*sizeof( string) );
          d -> Straniera = (string*) malloc( dimMax*sizeof( string) );
          }
else printf("CostDiz: memoria insufficiente\n");
return d;
 }
```

la conversione esplicita non è necessaria in C, ma facilita la lettura, i controlli e la portabilità

Il distruttore

```
void CancDiz( DizionarioP d )
{ /* Cancella il dizionario
Prec: d != NULL
Postc: libera la memoria allocata per d */
  assert(d);
  free( d->Italiano );
  free( d->Straniera );
  free( d );
 }
```

Estrattori di attributi

```
int ContaParole( DizionarioP d )
{/* Restituisce il numero delle parole nel dizionario
 Prec: d != NULL
 Postc: Restituisce il numero delle parole nel dizionario */
assert(d);
return d->num;
}

char *Trad_da_It( DizionarioP d, char *stringa )
/* trova la traduzione nella lingua straniera della  parola italiana stringa
   Prec: d != NULL && PAROLA_VALIDA( stringa ) > 0
   Postc: restituisce la traduzione dall' italiano di stringa o NULL se non c'è */
{ int i;
  assert( d);
  PAROLA_VALIDA( stringa );
  for( i=0; i<d->num; i++ )
            if ( strcmp( (d->Italiano)[i], stringa ) == 0 ) return (d->Straniera)[i];
return NULL;
}
```

Estrattori di attributi

```
char *Trad_da_Stran( DizionarioP d, char *stringa )
/* trova la traduzione in italiano della  parola stringa
   Prec: d != NULL && PAROLA_VALIDA( stringa ) > 0
   Postc: restituisce la traduzione in  italiano di stringa o NULL se non c'è */
{ int i;
  assert( d);
  PAROLA_VALIDA( stringa );
  for( i=0; i<d->num; i++ )
            if ( strcmp( (d->Straniera)[i], stringa ) == 0 ) return (d->Italiano)[i];
return NULL;
}
```

proprietà

```
void AggTrad( DizionarioP d, char *par1, char *par2 )
{/* aggiunge una nuova traduzione nel dizionario d
   della parola italiana in par1 con la parola straniera par2.
   par1 e par2 non devono essere già presenti in d
   Prec: d != NULL && par1 != NULL && strlen( par1 ) > 0 &&
   par2 != NULL && strlen( par2 ) > 0 && Trad_da_It( d, par1 ) == NULL &&
   Trad_da_Stran( d, par2 ) == NULL
  Postc: ContaParole( d ) = n+1 se   ContaParole( d ) = n prima dell'aggiunta &&
  strcmp( Trad_da_It( d, par1 ), par2 ) == 0 &&
   strcmp( Trad_da_Stran( d, par2 ), par1 ) == 0 */
   assert(d);
  PAROLA_VALIDA( par1 );
  PAROLA_VALIDA( par2 );
  assert( Trad_da_Stran( d, par1 ) == NULL );
  assert( Trad_da_It( d, par2 ) == NULL );
  d->Italiano[d->num] = par1;
  d->Straniera[d->num] = par2;
  d->num++;
}
```

Un esempio (incompleto) di programma di prova

```
...le inclusioni ...
#define NUM_MAX_CAR 25
#define NUM_MAX_PAR  100

#define NUM_TESTS  sizeof(tests)/sizeof(struct casi_test)

struct casi_test{
  char *parola;
  int in;
  } tests[] =
   { /*mettiamo 1 se la parola sarà presente nel file, 0 altrimenti*/
      {"tu", 1},
      {"noi", 0},
    {"verde",1},
    {"citta",1}
  };
```

```
int CaricaDiz( DizionarioP d, char *nomeFile )
{FILE *f; int i,numPar,dimDiz;  char  *w1, *w2;
 assert( d != NULL ); assert( nomeFile != NULL ); assert( strlen(nomeFile) > 0 );
 dimDiz = ContaParole(d);
  f = fopen( nomeFile, "r" );
  if ( f != NULL )
    if ( fscanf(f,"%d", &numPar ))
     {for(i=0; i<numPar;i++)
          {w1 = (char*) malloc(NUM_MAX_CAR*sizeof(char));
           w2 = (char*) malloc(NUM_MAX_CAR *sizeof(char));
           if ( fscanf(f,"%s%s", w1, w2 ) )
           { printf("le parole lette sono %s , %s\n",w1,w2);     /*le precondizioni!*/
             if ( !(Trad_da_It( d, w1 ) == NULL && Trad_da_Stran( d, w2 ) == NULL) )
               printf("[%s] oppure [%s] è gia' presente.\n", w1, w2 );
             else
               {AggTrad( d, w1, w2 );        /* ora aggiungiamo al dizionario */
                dimDiz++;
                if ( dimDiz != ContaParole(d) )
                printf("errore nell'inserimento di (%s,%s) al dizionario \n", w1, w2 );}} } }
    else  printf("il numero delle parole non è stato letto correttamente\n");
else printf("il file [%s]  non è stato aperto\n", nomeFile );
return dimDiz;}
```

Il main

```
int main(  )
{ int  i;
  char *nf, *par;
 DizionarioP d;
nf = "dizProva";
 d = CostDiz( NUM_MAX_PAR );
 if ( CaricaDiz( d,nf ) > 0 )
    printf("Numero Parole inserite: %d\n", ContaParole( d ) );
 /* Facciamo i tests  */
  for(i=0;i<NUM_TESTS;i++) {
    par = Trad_da_It( d, tests[i].parola );
    printf( "%s -> %s\n", tests[i].parola, (par == NULL) ? "NON presente" : par );
    if ( (par == NULL) == (tests[i].in == 0) ) { }
    else printf("ERRORE nella parola test nella posizione %d\n", i );
    }
   CancDiz(d);
 return 0;}
```