Un’utile funzione di stringa

```
char* strAllCpy(char* par);
/* alloca memoria e restituisce una copia di par,
prec: par!=NULL && strlen(par)>0
postc: restituisce un puntatore a una copia di par o NULL se non c'e' memoria*/

char* strAllCpy(char* par)
/* alloca memoria e restituisce una copia di par,
prec: par!=NULL && strlen(par)>0
postc: restituisce un puntatore a una copia di par o NULL se non c'e' memoria*/
{char * nuova;
assert(par!=NULL);
assert(strlen(par)>0);
nuova = malloc((strlen(par)+1)*sizeof(char));
assert(nuova);
nuova = strcpy(nuova ,par);
return nuova;
}
```

Aggiunta di una parola al dizionario: II versione

La versione fornita in precedenza di questa funzione lasciava all'utente il compito allocare la memoria per la stringa. Aveva sottolineato la pericolosità di quella soluzione e la necessità di introdurre una funzione come la strAllCpy per rimediare alla difficoltà. Di seguito trovate la versione adeguata della funzione che carica in memoria il dizionario.

```
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 */
  PAROLA_VALIDA( par1 );
  PAROLA_VALIDA( par2 );
  assert( Trad_da_Stran( d, par1 ) == NULL );
  assert( Trad_da_It( d, par2 ) == NULL );
 d->Italiano[d->num] = strAllCpy(par1);
 d->Straniera[d->num] = strAllCpy(par2);
  d->num++;
}
```

Il distruttore può quindi migliorare a sua volta

```
#define DIZ_VALIDO(d)     assert(d); assert(d -> Italiano);assert(d->Straniera);


 void CancDiz( DizionarioP d )
 { /* Cancella il dizionario
 Prec: d != NULL
 Postc: libera la memoria allocata per d */
   DIZ_VALIDO(d);
   for (i=0;i<d->num-1;i++)
      {free(d->Italiano[i]);
       free(d->Straniera[i]);
      }
   free( d->Italiano );
   free( d->Straniera );
   free( d );
  }
```

```
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 ))
           { w1 = (char*) malloc(NUM_MAX_CAR*sizeof(char));
             w2 = (char*) malloc(NUM_MAX_CAR *sizeof(char));
             for(i=0; i<numPar;i++)
                {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;}
```