Università Roma Tre

Esercizi su vettori, stringhe e liste

Appunti di ASD - Roberto De Virgilio

Indice

1 Esercizi

Esercizio 1

Una sequenza S di 27 interi si dice perfetta se per ogni i tale che $i \in [1..9]$ allora i ricorre esattamente tre volte nella sequenza S. Ad esempio, la sequenza

(1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7)

è perfetta. Si scriva una funzione C che prende come parametro un vettore di 27 interi e restituisce true (1) se il vettore rappresenta una sequenza perfetta e false (0) altrimenti.

Esercizio 2

Si assuma presente in memoria una lista composta di abbreviazioni (ad esempio TO, MI, RM) e dalle corrispondenti parole estese implementata tramite record e puntatori utilizzando la seguente struttura:

```
struct elem {
       char abbr[2];
       char *estesa;
       struct elem *next;
 }
```

Scrivere una funzione C che, ricevendo come parametri il puntatore all'inizio della lista, una abbreviazione e la corrispondente parola estesa cerchi l'abbreviazione nella lista e:

a) restituisca 0 se la coppia è presente nella lista

b) restituisca 1 se la coppia non è presente nella lista

c) restituisca 2 se l'abbreviazione è presente ma ad essa corrisponde una parola differente.

Inoltre, nel caso b) un nuovo record deve essere aggiunto in coda alla lista e nel caso c) la parola estesa deve essere sostituita da quella passata come parametro.

Esercizio 3

Si assuma presente in memoria un vettore di puntatori a liste di valori interi positivi. Le liste sono realizzate tramite record del seguente tipo:

```
struct nodo {
      int info;
      struct nodo *next;
   }
```

Scrivere una funzione C che prende come parametri il vettore e la sua dimensione, e restituisce l'indice della locazione del vettore contenente il puntatore alla lista in cui compare l'elemento di valore massimo tra gli elementi di tutte le liste. Nel caso tutte le liste siano vuote oppure il vettore abbia dimensione 0, la funzione deve restituire 0.

Esercizio 4

Si vuole gestire una classe di studenti tramite un vettore di dimensione variabile di record, dove i record hanno la seguente struttura:

```
struct elem {
     char *nome;
     int eta;
}
```

Scrivere una funzione C che prende come parametro il numero di studenti da inserire e che restituisce l'indirizzo del vettore leggendo da tastiera i nomi (non più lunghi di 20 caratteri) e l'età degli studenti. Gestire eventuali errori nella fase di allocazione di memoria, restituendo NULL in caso di errori, l'indirizzo del vettore in caso di successo.

Esercizio 5

Si considerino stringhe di lunghezza qualsiasi che contengono soltanto caratteri alfabetici minuscoli e spazi bianchi. Una stringa si dice palindroma se leggendola da destra verso sinistra ignorando gli spazi bianchi è uguale a se stessa. Ad esempio, le stringhe "aerea" e "etna gigante" sono palindrome.

Si consideri un vettore (di lunghezza ignota) di stringhe terminato dalla stringa vuota "". Si scriva una funzione C che prende come parametro tale vettore e restituisce l'ultima stringa palindroma presente nel vettore. Nel caso il vettore non contenga stringhe palindrome, la funzione deve restituire il valore NULL.

E' inoltre richiesto che la stringa restituita non condivida memoria con il vettore di stringhe.

Esercizio 6

Si assuma presente in memoria una lista (di lunghezza ignota) di record. I record della sequenza hanno la seguente struttura:

```
struct elem {
    char *nome;
    struct elem *next;
 }
```

Scrivere una funzione C che prende come parametro il puntatore iniziale alla lista, e restituisce un vettore di puntatori, ciascuno dei quali punta ad un record della lista.

Esercizio 7

Si scriva una funzione C che prende come parametri una stringa ed un carattere e restituisce un'altra stringa, da cui sono state rimosse tutte le occorrenze del carattere. Ad esempio, se la funzione viene chiamata con parametri "tutta statistica" e 't' deve restituire la stringa "ua saisica".

La lunghezza della stringa di ingresso non è nota. E' richiesto di non sprecare memoria. In particolare, la memoria allocata per la stringa in uscita deve essere esattamente uguale a quella necessaria per la sua memorizzazione. Si richiede inoltre di gestire eventuali errori nella fase di allocazione di memoria, restituendo NULL in caso di insuccesso.

Esercizio 8

Sia dato un vettore di dimensione nota di stringhe contenenti tre o più caratteri (escluso il terminatore di stringa). Si vuole aggiungere ad ogni stringa s nel vettore la prima di quelle seguenti che abbia esattamente le prime due lettere iniziali in comune con le due finali di s, evitando di ripetere le due lettere in comune. Nel caso nessuna delle stringhe seguenti abbia due lettere in comune con s non bisogna modificare s. Ad esempio, se il vettore contiene le seguenti stringhe:

```
casa
postino
sasso
osteria
salvia
notare
renna
```

dovrà essere trasformato nel seguente vettore:

```
casasso
postinotare
sasso
osteria
salvia
notarenna
renna
```

Si scriva una funzione C che ricevendo in ingresso l'indirizzo del vettore e la sua dimensione effettui la trasformazione di cui sopra.

Esercizio 9

Una stringa contiene nome e cognome di una persona separati tra loro da uno o più spazi. Una seconda stringa contiene il soprannome di una persona, e può eventualmente

contenere degli spazi. Si scriva una funzione C che prende in ingresso due stringhe del tipo suddetto e restituisce una nuova stringa in cui tra il nome ed il cognome è inserito il soprannome tra parentesi tonde. Nella nuova stringa, sia tra il nome e la parentesi aperta che tra la parentesi chiusa ed il cognome deve essere presente uno (ed un solo) spazio. Ad esempio, se le due stringhe sono "Bruce Springsteen" e "The Boss", la stringa restituita deve essere: "Bruce (The Boss) Springsteen".

Esercizio 10

Sia data una lista contenente almeno due elementi ed i cui record sono definiti tramite la seguente struttura C:

```
struct nodo{
   int valore;
   struct nodo* next;
   }
```

Si scriva una funzione C che ricevendo in ingresso un puntatore alla lista modifichi la stessa, memorizzando nell'ultimo nodo il prodotto del penultimo ed ultimo nodo, nel penultimo il prodotto del terzultimo e del penultimo e così via. Il primo record non deve essere modificato. Ad esempio, una lista contenente la sequenza di interi 4 6 2 3 9 verrà modificata dalla funzione nella lista 4 24 12 6 27.

Esercizio 11

Un numero intero positivo di lunghezza qualsiasi viene rappresentato tramite una stringa di caratteri numerici (cioè caratteri compresi tra '0' e '9'), con la cifra più significativa nella posizione 0 della stringa. Si scriva una funzione C che prende in ingresso due stringhe che rappresentato due interi positivi nel modo suddetto e restituisce una nuova stringa che rappresenta (nello stesso modo) la somma dei due numeri dati. Si supponga che le stringhe in ingresso siano di lunghezza uguale tra loro. Si assuma inoltre che la somma della cifra più significativa dei due numeri non generi riporto, e che, di conseguenza, la stringa risultato sia esattamente della stessa lunghezza delle stringhe date. Ad esempio, se in ingresso vengono fornite le stringhe "679524113529231 4" e "2314332174634521" il risultato deve essere la stringa "9109573309926835".

Suggerimento: Si ricordi che i codici ASCII dei caratteri numerici sono consecutivi tra loro. Di conseguenza il numero corrispondente ad un carattere numerico ch viene ottenuto con ch - '0'.

2 Soluzioni

Soluzione esercizio 1

Esistono moltissimi algoritmi per verificare la proprietà richiesta. La nostra soluzione si basa sull'idea di verificare per ciascun numero i tra 1 e 9 se, trovata la locazione delle prima occorrenza di i, le due occorrenze successive esistono e sono nella posizione desiderata. In dettaglio, il ciclo for esterno è governato dalla variabile i che varia da 1 a 9. Il ciclo più interno cerca una locazione j tale che esistano le tre occorrenze di i nel vettore nelle posizioni j, j+i+1 e j+2*i+2. Se una tale locazione j non esiste, oppure è oltre la posizione 24-2*i (che porterebbe il valore j+2*i+2 oltre la locazione 26), allora la funzione restituisce 0. Se invece tale j esiste, la funzione passa ad analizzare il successivo valore di i (istruzione break;). Si noti che utilizzare 24-2*i come estremo per j invece che 26 è indispensabile per evitare che la funzione acceda a locazioni inesistenti di v (con indice maggiore di 26).

```
#include <stdio.h>
int perfetta(int* v) {
  int i,j;
  for (i = 1; i <= 9; i++)
    { /* 24-2*i e' l'ultima locazione per la prima occorrenza di i che lasci
         spazio sufficiente per le altre 2 occorrenze di i */
      for (j = 0; j <= 24-2*i; j++)
        if (v[j] == i && v[j+i+1] == i && v[j+2*i+2] == i)
          break;
      if (j > 24-2*i) /* non ho trovato un j che soddisfi la condizione per i */
return 0; }
return 1; }
main()
{ /* programma principale di prova (non richiesto) */
  int VS[4][27] = {
      /* VS[0]: sequenza perfetta */
 {1,9,1,2,1,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7},
     /* VS[1], VS[2], VS[3]: sequenze non perfette */
    {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1},
    {1,9,1,1,2,8,2,4,6,2,7,9,4,5,8,6,3,4,7,5,3,9,6,8,3,5,7},
    {1,9,1,2,1,8,2,4,6,2,0,9,4,5,8,6,3,4,0,5,3,9,6,8,3,5,0}};
  int i;
  for (i = 0; i < 4; i++)
    {
      printf("La sequenza %d e' perfetta?  ",i);
      if (perfetta(VS[i]))
        printf("Si\n");
      else
        printf("No\n");
```

```
    }
}
```

Soluzione esercizio 2

La funzione scandisce la lista sino a trovare l'abbreviazione cercata o l'ultimo record. Si noti che il test del ciclo l-¿next, equivalente a l-¿next != NULL, è inconsistente nel caso di liste vuote. Inoltre, avendo utilizzato un array di due elementi per memorizzare le abbreviazioni non è possibile utilizzare le funzioni di libreria strcmp() e strcpy() che si basano sulla presenza del terminatore di stringa.

```
#include <stdio.h>
struct elem {
  char abbr[2];
  char *estesa;
  struct elem *next;
};
int controlla(struct elem *l, char a[2], char *p)
{
  int app;
  while ( l->next && (l->abbr[0] != a[0] || l->abbr[1] != a[1]) )
    l = l->next;
  /* si esce dal ciclo perche' siamo sull' ultimo record (CASO A)
     o sul record che contiene l'abbreviazione          (CASO B)
     o entrambe le cose contemporaneamente              (CASO C) */
  if ( (l->abbr[0] == a[0]) && (l->abbr[1] == a[1]) )
    if ( strcmp(l->estesa, p) !=0 )
      {/* Caso B o C. La parola estesa e' differente: la sostituisco */
       l->estesa = (struct elem *) realloc(l->estesa, strlen(p)+1);
       strcpy(l->estesa, p);
       app = 2;   /* abbreviazione esistente parola estesa differente */
      }
    else app = 0; /* Caso B o C. La parola estesa e' OK */
  else
    {   /* Caso A: siamo sull'ultimo record che NON contiene l'abbreviazione
           aggiungo un record in coda alla lista */
     l->next = (struct elem *)malloc(sizeof(struct elem));
     l = l->next;
    l->estesa = (char *) malloc(strlen(p)+1);
     strcpy(l->estesa, p);
     l->abbr[0] = a[0];
     l->abbr[1] = a[1];
l->next = 0,
```

```
     app = 1; /* abbreviazione non esistente */
    }
return app; }
void stampa(struct elem * l)
{  /* stampa la lista, non richiesta */
  while (l != NULL){
    printf("%c%c %s \n",l->abbr[0],l->abbr[1],l->estesa);
    l = l->next;
} }
struct elem *crealista(int k)
{
  int i;
  struct elem *l, *app;
  char c[21];
  if(k == 0)
    return NULL;
  else
/* funzione di appoggio, non richiesta */
/* crea una lista di k record          */
    {
     app = l = (struct elem *) malloc(sizeof(struct elem)); /* record generatore */
     for(i = 0;i < k;++i)
       {
        l->next = (struct elem *)malloc(sizeof(struct elem));
        l = l->next;
        printf("Abbreviazione : ");
        scanf("%c%c",&(l->abbr[0]), &(l->abbr[1]));
        printf("Estesa        : ");
        scanf("%s%*c",c);        /* %*c consuma il \n */
        l->estesa = (char *) malloc(strlen(c)+1);
        strcpy(l->estesa,c);
       }
     l->next = NULL;
     l = app->next;
     free (app);
     return l;
} }
main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p1, **v;
  int i;
  char app[2] = {'z','z'},str[21];
  p1 = crealista(3);  /*crea una lista di 3record */
```

```
  stampa(p1);
  printf("inserire xx per uscire dal ciclo di controllo...\n");
  while ((app[0] != 'x') || (app[1] != 'x') )
    {
 printf ("Abbreviazione : ");
 scanf("%c%c",&app[0],&app[1]);
  printf("Estesa        : ");
 scanf("%s%*c",str);
printf("%c%c %s : %d \n",app[0],app[1],str,controlla(p1,app,str));
}
stampa(p1);
}
```

Soluzione esercizio 3

La funzione richiesta si compone di un doppio ciclo di scansione. Il ciclo esterno scandisce il vettore di liste, e quello interno scandisce ogni singola lista. Quando viene incontrato un elemento maggiore del massimo corrente viene memorizzato il suo valore e l'indice della lista in cui compare. Si noti che non è necessario memorizzare il massimo di ogni singola lista, ma è sufficiente cercare il massimo globalmente su tutto il vettore. Sfruttando il fatto che i valori sono tutti interi positivi è possibile inizializzare l'ottimo corrente al valore 0. Questa inizializzazione permette anche di trattare implicitamente i casi che i vettore abbia lunghezza 0 e le liste siano tutti vuoti. In questi due casi infatti, il massimo corrente non viene mai aggiornato, e viene quindi restituito correttamente il valore 0.

```
#include <stdio.h>

struct nodo {
  int info;
  struct nodo *next;
};

int ElementoMassimo(struct nodo **v, int n) {
  int i, indice_max = 0, val_max = 0;
  struct nodo *p;
  for (i = 0; i < n; i++) {
      p = v[i];
      while (p != NULL) {
          if (p->info > val_max){
              val_max = p->info;
              indice_max = i;
}
p = p->next;
```

```
} }
  return indice_max;
}

/* funzioni ausiliarie di prova (non richieste dal testo) */

struct nodo **LeggiVettore(int n)
{
  struct nodo **v;
  struct nodo *p;
  int i,j;
  v = (struct nodo **) malloc(n * sizeof(struct nodo*));
  for (i = 0; i < n; i++)
    {
      printf("Elementi (rovesciati) della riga %d [0 per finire]: ",i);
      v[i] = NULL;
      scanf("%d",&j);
      while (j > 0)
        {
          p = (struct nodo *) malloc(sizeof(struct nodo));
          p->info = j;
          p->next = v[i];
          v[i] = p;
          scanf("%d",&j);
} }
return v; }
void StampaVettore(struct nodo **v, int n)
{
  int i;
  struct nodo *p;
  for (i = 0; i < n; i++)
    {
      p = v[i];
      while (p != NULL)
        {
          printf("%d ", p->info);
          p = p->next;
        }
      printf("\n");
} }
main() { /* programma principale di prova (non richiesto) */
  int n;
  struct nodo **vett;
```

```
  printf("Quante liste? ");
  scanf("%d",&n);
  vett = LeggiVettore(n);
  StampaVettore(vett,n);
  printf("Il valore massimo e' nella lista %d\n", ElementoMassimo(vett,n));
}
```

Soluzione esercizio 4

La funzione alloca dinamicamente un vettore di n record e gestisce in un ciclo for l'input dei dati relativi agli n studenti. Si noti che in ogni record del vettore c'è lo spazio per un puntatore a carattere e non per una stringa: è quindi necessario allocare dinamicamente, per ogni record, la memoria per il nome dello studente.

```
#include <stdio.h>

struct elem {
  char *nome;
  int eta;
};

struct elem * leggi(int n) {
  struct elem *p;
  char app[21];
  int i;
  p = (struct elem *) malloc(n*sizeof(struct elem));
  if (p == NULL) return NULL;
  for (i = 0; i < n; ++i)
    {
     printf("Nome: ");
     scanf("%s", app);
     printf("Eta': ");
     scanf("%d", &(p[i].eta));
     p[i].nome = (char *)malloc(strlen(app)+1);
     if (p[i].nome == NULL) return NULL;
     strcpy(p[i].nome, app);
} return p;
}

main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p;
  int i;
```

```
  p = leggi(3);
  printf("*************\n");
  for (i = 0;i < K;++i)
   printf("Nome: %-20s eta': %3d\n", p[i].nome, p[i].eta);
}
```

Soluzione esercizio 5

La funzione UltimaPalindroma() che risolve l'esercizio si avvale di una funzione ausiliaria Palindroma() che verifica se una stringa è palindroma o meno. La funzione principale si avvale di una scansione in avanti del vettore, durante la quale ogni volta che viene incontrata una stringa palindroma, viene registrato nella variabile indice l'indice della locazione in cui compare. In questo modo ci si assicura che alla fine della scansione l'indice memorizzato sia quello dell'ultima palindroma. Il test che governa il ciclo while si basa sulla presenza del terminatore di stringa nella prima posizione della stringa. Analogamente avremmo potuto usare dei test del tipo

```
strlen(vettore_stringhe[i]) == 0
```

oppure

```
!strcmp(vettore_stringhe[i],"")
```

Sbagliato sarebbe stato invece il test

```
vettore_stringhe[i] == NULL
```

in quanto in questo caso si sarebbe verificato il fatto che il puntatore ha valore nullo e non che questo punta ad una locazione contenente la stringa nulla. Riguardo alla funzione Palindroma(), questa deve gestire l'eventuale presenza di spazi bianchi nella stringa. Si noti che i due cicli while iniziali sono necessari perch?e si assume che gli spazi bianchi possano trovarsi anche ad inizio e fine di stringa. Il secondo ciclo ha la condizione aggiuntiva j ¿ 0 per il caso che la stringa contenga solo spazi bianchi (si considera tale stringa come palindroma).

```
#include <stdio.h>

int Palindroma(char* s)
{/* funzione ausiliaria: restituisce 1 se la stringa s
    e' palindroma, restituisce 0 altrimenti*/
  int i = 0, j = strlen(s) - 1;
  while (s[i] == ' ')
    i++;
  while (s[j] == ' ' && j > 0)
```

```
    j--;
while (i < j) {
      if (s[i] != s[j]) return 0;
      while (s[++i] == ' ')
        ; /* istruzione vuota */
      while (s[--j] == ' ')
        ; /* istruzione vuota */
    }
return 1; }

char* UltimaPalindroma(char* vettore_stringhe[])
{/* funzione principale dell'esercizio */
  int i = 0;
  char* ultima_palindroma;
  int indice = -1;
  while (vettore_stringhe[i][0] != '\0')
    {
      if (Palindroma(vettore_stringhe[i]))
        indice = i;
i++; }
  if (indice == -1)
    return NULL;
else {
ultima_palindroma = (char*) malloc(strlen(vettore_stringhe[indice]) + 1);
  strcpy(ultima_palindroma,vettore_stringhe[indice]);
  return ultima_palindroma;
}
}

main()
{ /* programma principale di prova (non richiesto) */
  char* v[] = {"   a  d da", "etna gigante", "paolo", "marco", ""};
  char* palindroma = UltimaPalindroma(v);
  if (palindroma == NULL)
    printf("Il vettore non contiene parole palindrome\n");
  else
    printf("L'ultima parola palindroma e': ??%s''\n", palindroma);
}
```

Soluzione esercizio 6

Dovendo la funzione restituire un array di puntatori a record decidiamo che il tipo della funzione stessa è di puntatore a puntatore di record (struct elem **). La funzione scandisce

due volte la lista: la prima per contare i record e, dopo aver allocato memoria, la seconda per assegnare agli elementi del vettore appena creato gli indirizzi dei record della lista.

```
#include <stdio.h>

struct elem {
  char *nome;
  struct elem *next;
};

struct elem **list2vet(struct elem *p)
{
  int i, n = 0;
  struct elem *app, **v;
  app = p;
  while (app != NULL)
    {
     ++n;        /* conta i record della lista */
     app = app->next;
}
if (n != 0)
    {
     v=(struct elem **) malloc(n*sizeof(struct elem*));
     for (i = 0; i < n; ++i)
     {
      v[i] = p;
      p = p->next;
} return v;
  }
else return NULL;
/* lista vuota */
}

struct elem *crealista(int k)    /* funzione di appoggio, non richiesta */
{ /* crea una lista di k record          */
int i;
struct elem *l, *app;
  char c[21];
  if (k == 0)
    return NULL;
  else
    {
     app=l=(struct elem *) malloc(sizeof(struct elem)); /* record generatore */
```

```
    for(i = 0;i < k;++i)
      {
       l->next = (struct elem *)malloc(sizeof(struct elem));
       l = l->next;
       printf("Inserire un elemento : ");
       scanf("%s",c);
       l->nome = (char *) malloc(strlen(c)+1);
       strcpy(l->nome,c);
      }
    l->next = NULL;
    l = app->next;
    free(app);
    return l;
} }

main () /* programma principale di test, non richiesto dal compito */
{
  struct elem *p1, **v;
  int i;
  p1 = crealista(3);  /*crea una lista di 3 stringhe */
  v = list2vet(p1);
  for(i = 0; i < 3; ++i)
    printf("Nell' elemento %d di v c'e': %s\n",i,v[i]->nome);
}
```

Soluzione esercizio 7

La funzione alloca, in una variabile di appoggio, memoria sufficiente a contenere la stessa stringa passata per parametro. Un ciclo di strlen(str) + 1 iterazioni scandisce la stringa di ingresso copiandola, carattere per carattere su quella di appoggio, saltando i caratteri uguali al parametro di ingresso c. In particolare la variabile i è utilizzata per scandire la stringa sorgente mentre la variabile j è utilizzata per scandire la stringa destinazione. L'ultimo ciclo del for copierà il terminatore di stringa. Essendo str una copia del parametro di ingresso è possibile utilizzarlo per copiarci sopra la nuova stringa. Guadagnando in leggibilità ed utilizzando una variabile in più la stessa cosa poteva essere fatta utilizzando una ulteriore variabile di appoggio.

```
#include <stdio.h>

char * cancella(char *str, char c)
{
  int i, j = 0;
  char *app;
```

```
  app = (char *) malloc(strlen(str)+1);
  for (i = 0; i <= strlen(str); ++i) /* copia anche il terminatore di stringa */
    if (str[i] != c)
      app[j++] = str[i];
  str = (char *) malloc(strlen(app)+1);  /* str non serve più */
strcpy(str,app);
free(app);
return str;
}

main () /* programma principale di test, non richiesto dal compito */
{
   char *s = "sschissa se ss s sfunzionasssssssssss";
   printf("%s\n",s);
   printf("%s\n",cancella(s,'s'));
}
```

Soluzione esercizio 8

```
void modifica(char **v, int n){
  int i,j;
  char *app;
  for (i = 0; i < n-1; ++i)
    for(j = i+1; j < n; ++j)
      if ( (v[i][ strlen(v[i])-2 ] == v[j][0]) &&
           (v[i][ strlen(v[i])-1 ] == v[j][1]) )
        {
          v[i] = (char *) realloc(v[i],strlen(v[i])+strlen(v[j])-2+1);
          strcat(v[i],v[j]+2);
          break;
} }
```

Soluzione esercizio 9

La funzione inizialmente conta il numero di spazi presenti tra il nome e il cognome, in modo da poter allocare correttamente la stringa risultante (ris). In particolare, la dimensione della stringa risultante è pari alla dimensione della stringa nome cognome privata degli spazi, più la dimensione del soprannome più‘ cinque (2 per le parentesi, 2 per gli spazi e uno per il terminatore di stringa). Nei tre cicli successivi vengono scritti nella stringa il nome, il soprannome e il cognome rispettivamente. Le parentesi, gli spazi e il terminatore di stringa vengono inseriti nella stringa da apposite istruzioni di assegnazioni.

```
#include <stdio.h>
```

```
char* AggiungiSoprannome(char* nome_cognome, char* soprannome)
{
  char *ris; /* stringa risultato */
  int i,  /* indice per scorrere il soprannome */
      j,  /* indice per scorrere il risultato */
      k;  /* indice per scorrere nome e cognome */
  int num_spazi = 0; /* numero di spazi tra nome e cognome */
  for (k = 0; nome_cognome[k] != '\0'; k++)
    if (nome_cognome[k] == ' ')
        num_spazi++;
  ris = (char*) malloc(strlen(nome_cognome) - num_spazi +
                       strlen(soprannome) + 5);
     /* 5 caratteri extra: 2 parentesi, 2 spazi, 1 terminatore */
  for (k = 0; nome_cognome[k] != ' '; k++)
    ris[k] = nome_cognome[k];
  j = k;
  ris[j++] = ' ';
  ris[j++] = '(';
  for (i = 0; soprannome[i] != '\0'; i++, j++)
    ris[j] = soprannome[i];
  ris[j++] = ')';
  ris[j++] = ' ';
  for ( ; nome_cognome[k] == ' '; k++)
    ;
  for ( ; nome_cognome[k] != '\0'; k++, j++)
    ris[j] = nome_cognome[k];
  ris[j] = '\0';
return ris; }

main()
{ /* programma principale (non richesto dal testo) */
  char stringa_nome[31], stringa_soprannome[21], ch;
  int i = 0;
  printf("Inserisci la stringa con nome e cognome\n");
  while ( (ch = getchar()) != '\n')
    stringa_nome[i++] = ch;
  stringa_nome[i] = '\0';
  i = 0;
  printf("Inserisci la stringa con il soprannome\n");
  while ( (ch = getchar()) != '\n')
    stringa_soprannome[i++] = ch;
  stringa_soprannome[i] = '\0';
 printf("%s\n",AggiungiSoprannome(stringa_nome, stringa_soprannome));
```

```
}
```

Soluzione esercizio 10

Si propongono per questo esercizio due soluzioni diverse: una iterativa (funzione modifica()) ed una ricorsiva (funzione modifica2()).

```
#include <stdio.h>

struct elem{
   int valore;
  struct elem * next;
};

void modifica(struct elem *l)
{
  int prec, copia;
  prec = l->valore;
  do
    {
      l = l->next;
      copia = l->valore;
      l->valore = l->valore + prec;
      prec = copia;
    }
  while (l->next);
}

int modifica2(struct elem *inizio)
{
  if (inizio->next->next == NULL)
    inizio->next->valore += inizio->valore;
else {
      inizio->next->valore += inizio->valore;
    }
}

struct elem* crea(int n)
{ /* funzione ausiliaria (non richiesta): crea una lista
     con i valori n, n-1, ..., 2, 1 */
  int i;
  struct elem *l, *app;
  app = (struct elem *) malloc(sizeof(struct elem));
```

```
  app->next = NULL;
  for (i = 1; i <= n; ++i)
    {
      app->valore = i;
      l = app;
      app = (struct elem *) malloc(sizeof(struct elem));
      app->next = l;
}
free(app);
return l; }

void stampa(struct elem *l)
{/* funzione ausiliaria (non richiesta): stampa una lista */
  while (l != NULL)
    {
      printf("%d ",l->valore);
l=l->next; }
  printf("\n");
}

main(){/* programma principale di prova (non richiesto) */
  int i;
  struct elem *p;
  p = crea(10);
  stampa(p);
  modifica(p);
  stampa(p);
  modifica2(p);
  stampa(p);
}
```

Soluzione esercizio 11

Le assunzioni fatte nel testo ci garantiscono con certezza che la stringa risultante ha la stessa lunghezza delle due stringhe in ingresso. E‘ quindi possibile allocare subito tale stringa della sua lunghezza definitiva. Successivamente, si esegue un ciclo in cui viene calcolata ogni cifra del risultato. Tale ciclo parte dall'ultima locazione delle stringhe (cioè le cifre meno significative dei numeri) per poter propagare correttamente il riporto verso le cifre più significative. La conversione tra i caratteri '0', '1', . . . , '9' e i valori interi 0, 1, . . . , 9, viene gestito tramite sottrazione e somma del valore del carattere '0'.

```
#include <stdio.h>
```

```
char* Somma(char* s1, char* s2){
  char* ris;
  int n = strlen(s1);
  int i, carry = 0, somma;
  ris = (char*) malloc(n + 1);
  ris[n] = '\0';
  for (i = n-1; i >= 0; i--)
    {
      somma = (s1[i] - '0') + (s2[i] - '0') + carry;
      carry = somma / 10;
      ris[i] = somma % 10 + '0';
}
return ris;
}

main(){ /* programma principale di prova (non richiesto) */
  char a[101], b[101];
  printf("Inserisci il primo numero : ");
  scanf("%s",a);
  printf("Inserisci il secondo numero (%d cifre): ", strlen(a));
  scanf("%s",b);
  printf("La somma di %s e %s e' : %s\n",a,b,Somma(a,b));
}
```