Politecnico di Milano - Dipartimento di Elettronica e informazione

Prof. Mauro Negri

Fondamenti di Informatica **6 settembre 2012**

III appello

Matricola_____ Cognome _____ Nome _____

Durata prova: 1 ora 30 minuti

Istruzioni

LE RISPOSTE DEVONO ESSERE SCRITTE SU QUESTO PLICO (anche sul retro)

Non separare questi fogli.

L'uso di cellulari, libri, eserciziari, appunti o calcolatrici durante lo svolgimento della prova comporta l'annullamento della prova.

Non è possibile uscire dall'aula durante la prova tranne in casi eccezionali.

Si può utilizzare la matita e pertanto sono penalizzati compiti illeggibili o pasticciati.

Esercizio 1 (14 punti) _____

Esercizio 2 (4 punti) _____

Esercizio 3 (12 punti) _____

Punteggio totale (30 punti) _____

Esercizio 1.

a) Scrivere una funzione C **(sul retro della precedente pagina)** di nome DOPPIE che legge da terminale una sequenza di caratteri di lunghezza arbitraria e terminata dal carattere newline '\n'; la funzione supponga di ricevere in ingresso solo lettere minuscole dalla 'a' alla 'z' (alfabeto italiano) e spazi bianchi oltre al carattere terminatore. La funzione identifica le doppie presenti nella sequenza (coppie di lettere uguali adiacenti) e per ogni coppia conta quante volte essa sia presente nella sequenza; la funzione supponga che non esistano mai nella sequenza più di due lettere uguali adiacenti (una tripla "dittta" ad esempio non apparirà mai nella sequenza).
La funzione restituisce come parametro un vettore che memorizza nei suoi elementi le doppie trovate nella sequenza e la loro frequenza.
Ad esempio, la stringa "il gatto di pezza gioca con la bella donzella" memorizzerà i seguenti valori per le 3 coppie incontrate:

tt	1
zz	1
ll	2

Si noti che l'ordinamento delle coppie all'interno del vettore può essere qualsiasi.

b) Infine completare il seguente main, inserendo al punto1) la definizione della variabile globale VET che rappresenta il vettore che sarà caricato dalla funzione DOPPIE e al punto 2) completando l'invocazione della funzione.

```
int main ()
{
1) definizione VET
#define MAX 21
typedef struct {
        char coppia[3];
        int occorrenze;
} s_VET;
s_VET VET[MAX];
int i;

2)      DOPPIE(VET);
//per visualizzare – non richiesto
for (i = 0; i < MAX; i++)  printf("%s compare %d volte\n", VET[i].coppia, VET[i].occorrenze);
}
```

.

Soluzione parte a)

```
1. La sequenza può contenere solo lettere minuscole alfabeto italiano
2. Le lettere uguali adiacenti possono essere al massimo 2
3. Non definire un vettore per contenere la sequenza letta dal terminale
perché comporta la definizione di una lunghezza massima della stringa.
L'esercizio richiede lunghezza arbitraria ed è possibile evitare tale vettore.
```

```
#include <stdio.h>

//tipo strutturato
Vedi punto b)

void DOPPIE(s_VET VET[])
{ //vettore temporaneo per la fase di scansione
int V[MAX]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};
```

```
 char cp,cd; int i;

 for (i=0;i<MAX;i++) {VET[i].coppia[0]= 'a'+i; VET[i].coppia[1]= 'a'+i;
                          VET[i].occorrenze=0;
                          //opzionale
                          VET[i].coppia[2]='\0';
                      }


 cp= getchar(); if (cp=='\n') {printf("\nseq.vuota");return;}

 cd=getchar();
 if  (cd=='\n') {printf("\nseq.con 1 elemento");return;}
              else if ((cp==cd)&&(cp!=' ')) per non contare come doppie due blank adiacenti
                                   V[cp-'a']++;
 cp=cd;
 cd=getchar();

 while(cd!='\n')
 {     //verifica se il carattere attuale e' uguale al successivo
       if ((cd == cp) &&(cp!=' ')) V[cp-'a']++;
      cp=cd; cd=getchar();
 }

 for (i=0;i<MAX;i++) VET[i].occorrenze = V[i];
}
```

Note:

1) Era possibile evitare vettore temporaneo e lavorare direttamente su VET

Esercizio 2.

Tradurre il seguente programma C in assembler (ipotizzare che lo stack sia stato allocato e SP correttamente inizializzato.

```
…
typedef struct {float a; int b;} t_s;
t_s X, Y, Z[2]; int W;

t_s S(t_s  *P2)
   {int vett[5]; (*P2).b= vet[1]; return 10;}
int main ()
   {  W = S(&Y);  }
```

Istruzioni assembly

[W:] ADD oper1,oper2	somma in oper2
[W:] MOV oper1, oper2	copia in oper2
[W:] RTS	ritorno da funzione
[W:] JTS oper	invocazione sottoprogramma
[W:] EXIT	termina esecuzione programma

Modalità di indirizzamento operandi delle istruzioni:

- Ri o SP — contenuto del registro Ri o SP
- (Ri) o (SP) — contenuto della parola di memoria il cui indirizzo è nel registro Ri o SP
- #X — valore del simbolo X

Ipotesi: il contenuto di R1,…, R6 (non R0 e SP) non va salvato prima dell'uso dei registri.

```
X: .RES 2
Y: .RES 2
Z: .RES 4
W: .RES 1
MAIN: ADD #-1, SP
      MOV #Y, (SP)
      ADD #-1, SP
      JTS S
      ADD #2, SP
      MOV (SP), W
      EXIT
S:    MOV R0, (SP)
      ADD #-1, SP
      MOV SP, R0
      ADD#-5,SP

      MOV R0, R1          (*p2).B=vet[1]
      ADD #1, R1          R1 punta a vet[1]

      MOV R0, R2
      ADD #3, R2
      MOV (R2), R2        R2 punta a y.b
      ADD #1, R2

      MOV R0,R1           PREDISPOSIZIONE VALORE 10 DI RITORNO
      ADD #4,R1
      MOV #10, (R1)

      ADD # 6, SP
      MOV (SP), R0
      RTS
```

Esercizio 3. Date le seguenti definizioni che descrivono una lista dinamica nella quale ogni elemento punta al successivo e al precedente e le variabili che puntano rispettivamente al primo e all'ultimo elemento:

```
typedef struct el     {  int  valore; struct el  * prossimo; struct el  * precedente; }   ElemLista;
typedef struct {ElemLista  *primo; ElemLista  *ultimo;} t_pointer;
t_pointer   L = {NULL,NULL};
```

Realizzare la seguente funzione C di nome **INS**, che riceve come parametri la lista e due numeri interi NUM e POS (i numeri NUM e POS possono assumere un qualsiasi valore intero). La funzione crea un elemento dinamico inserendo NUM nel campo valore e procede a cercare la posizione POS di inserimento dell'elemento nella lista, partendo dal "primo" che occupa la posizione 1 (se POS vale P significa che l'elemento può essere inserito nella lista se e solo se è possibile inserirlo come P-esimo elemento della lista, altrimenti l'inserimento non deve avvenire e l'elemento deve essere eliminato). La funzione ritorna come parametri la lista aggiornata e 1 se l'aggiornamento è avvenuto o 0 se non è stato possibile inserire l'elemento.

Soluzione /*Ipotesi del testo:
POS è un intero qualsiasi
La soluzione presentata funziona supponendo di avere nella lista almeno (POS-1) elementi, ma è stato ammessa durante l'esame l'ipotesi di avere almeno POS elementi nella lista.

Condizioni verificate dalla soluzione proposta
Pos<1 : errore poiché pos minima è 1
Pos=1: si inserisce sempre, tuttavia nel caso di lista vuota ricordarsi che il nuovo elemento è anche l'ultimo
Pos>1: verifica che pos corrisponda al più alla lunghezza della lista +1, altrimenti non è possibile inserire. Se inseribile allora esistono due casi:
a) inserimento in posizione intermedia della lista
b) inserimento in ultima posizione (non esiste successivo)

Soluzione */

```
#include <stdio.h>
#include <stdlib.h>
typedef
    struct el {  int   valore; struct el  *prossimo; struct el *precedente; }
    ElemLista;
typedef struct {ElemLista  *primo;  ElemLista   *ultimo; } t_pointer;
t_pointer    pointer={NULL,NULL};

int INSERT(t_pointer *p, int val, int pos)
{ ElemLista *temp, *prec; int precpos;

 if (pos<1) return 0;

 if (pos==1)
  { temp=(ElemLista*) malloc(sizeof(ElemLista));
    if (temp==NULL)return 0;
          else
     { temp->valore=val; temp->prossimo=p->primo; p->primo=temp;
       temp->precedente=NULL;
        // verifica esistenza di altro elemento nella lista
       if (temp->prossimo==NULL)  p->ultimo=temp;
       else temp->prossimo->precedente=temp;
      return 1;
     }
   }
 else
       //pos>1 verifica esistenza posizione
  {prec = p->primo; precpos=1;
   while((prec!=NULL) && (precpos <(pos-1)))
                     {precpos++; prec=prec->prossimo;}
   if (prec==NULL)
              //lista contiene meno elementi di quelli richiesti da pos
     return 0;
   else
     {temp= malloc(sizeof(ElemLista));
      if (temp==NULL)return 0;
      else //inserimento
         {temp->valore=val;  temp->prossimo = prec->prossimo;
          prec->prossimo=temp;
          temp->precedente=prec;
          if (temp->prossimo==NULL) p->ultimo=temp;
          else temp->prossimo->precedente=temp;
         }
     }
    return 1;
   }
}
```

```
void main()
{ int v,p;
   printf("inserire valore e posizione"); scanf("%d",&v);scanf("%d",&p);
   printf("\n%d",INSERT(&pointer, v,p));
}
```