INFORMATICA 2010/11 - CdL in FISICA

TERZO APPELLO – 8/09/2011: SOLUZIONI PROPOSTE

Scrivere **in stampatello** COGNOME, NOME e MATRICOLA su ogni foglio consegnato

N.B.: In tutti gli esercizi viene valutata anche la leggibilità del codice proposto. Non è consentito l'uso di istruzioni che alterino il normale flusso dell'esecuzione (come `continue`, `break` (tranne che in uno `switch`) e istruzioni di `return` all'interno di cicli che ne provochino l'uscita forzata).
Invece di usare interi come variabili booleane, utilizzare il tipo `boolean` definito da

`typedef enum {false, true} boolean;`

Le procedure/funzioni in generale non devono contenere alcuna istruzione di input/output (ad es. `scanf, printf, getchar, putchar,...`)

ESERCIZIO 1
La *distanza* tra due numeri reali x e y è data dal valore assoluto della loro differenza. La *lunghezza* di una sequenza di numeri reali è la somma delle distanze tra ogni numero (escluso l'ultimo) e il successivo.

Scrivere un programma C che legge una sequenza di numeri reali fornita dall'utente, e ne calcola e stampa la lunghezza. La sequenza termina quando l'utente immette due valori successivi con distanza zero. Il programma non deve utilizzare né un array né una struttura dati dinamica per memorizzare la sequenza. Il programma *può* (ma non deve) essere strutturato in modo da utilizzare una procedura o funzione ricorsiva (che naturalmente può utilizzare istruzioni di input/output per interagire con l'utente).

Esempio di soluzione non ricorsiva:

```
int main(){
  float num, diff, prec, lung=0;
  printf("Dammi un numero, anche decimale\n");
  scanf("%f", &prec);
  do {
    printf("Dammi un numero, anche decimale\n");
    scanf("%f", &num);
    diff = prec - num;
    if(diff<0) diff *= -1;
    lung += diff;
    prec = num;
  } while (diff!=0);
  printf("Lunghezza: %f \n",lung);
  return 0;
}
```

Esempio di soluzione ricorsiva:

```
float readNext(float last, float lung){
  float num, diff;
  printf("Dammi un numero, anche decimale\n");
  scanf("%f", &num);
  diff = last - num;
  if(diff<0) diff *= -1;
  lung += diff;
  if (diff==0) return lung;
  else return readNext(num, lung);
}
int main(){
  float num, lung;
  printf("Dammi un numero, anche decimale\n");
  scanf("%f", &num);
```

```
  lung = readNext(num, 0);
  printf("Lunghezza: %f \n",lung);
  return 0;
}
```

ESERCIZIO 2
Scrivere una funzione C che ha come unico parametro formale una stringa, e restituisce `true` se e solo se tutti i caratteri della stringa diversi dallo spazio sono distinti. Nel confronto tra caratteri ignorare le differenze tra caratteri alfabetici minuscoli e maiuscoli. Per esempio, la funzione deve restituire `true` per le seguenti stringhe: "`aiuole`", "`un po' di fame`", "`*#sgRtZ!?`"; invece deve restituire `false` per le seguenti: "`aiuola`", "`un po' di Dame`", "`*#sgRtZ!??`".

Esempio di soluzione (con funzione ausiliaria per convertire caratteri in minuscoli):

```
int m(char x){ /* converte il carattere in minuscolo */
  if (x >= 'A' && x <= 'Z') return x - 'A' + 'a';
  return x;
}
boolean distinti(char* str) {
  boolean res=true;
  char act, *cur, *pos=str;
  while(res && *pos != '\0'){
    if (*pos != ' '){
      cur = pos+1;
      act = m(*pos);
      while(res && *cur != '\0') {
        res &= (act != m(*cur));
        cur++;
      }
    }
    pos++;
  }
  return res;
}
```

ESERCIZIO 3
Sia data la seguente funzione ricorsiva C:

```
int bamboo(int arr[], int dim, boolean flag, int val){
  int res;
  if (dim <= 0) return val;
  if (flag) {
    *arr = bamboo(arr+1, dim-1, false, *arr);
    return 0;
  } else {
    bamboo(arr+1, dim-1, true, *arr);
    res = arr[0];
    *arr = val;
    return res;
  }
}
```

Sia dato inoltre il seguente `main`:

```
int main(){
  int dim = 5;
  int arr [] = {11,22,33,44,55};
  bamboo(arr, dim, true, 0);
}
```

1. Mostrare il contenuto dell'array `arr` al momento di ogni chiamata della funzione `bamboo` successiva alla prima, e il contenuto di `arr` alla fine dell'esecuzione del `main`.

2. Spiegare cosa fa, intuitivamente, la funzione `bamboo` quando viene invocata con flag `true` su un array di lunghezza n, per $n = 0$, $n = 1$, n pari maggiore di 0, e n dispari maggiore di 1.

Esempio di soluzione:

1. L'array è immutato al momento di ogni chiamata della funzione bamboo, infatti questo viene modificato solo dalle istruzioni successive che non vengono eseguite finché la ricorsione non inizia a chiudersi. L'array alla fine dell'esecuzione del main è $\{22, 11, 44, 33, 55\}$.

2. La funzione bamboo, invocata con flag true, scambia gli elementi di posizione 0-1, 2-3, 4-5 ecc... ignorando l'ultimo elemento se la dimensione è dispari. Con lunghezza 0 la funzione ritorna il valore passato come argomento `val` mentre con lunghezza 1 la funzione ritorna 0. In entrambi i casi l'array resta invariato.

ESERCIZIO 4

Dato un insieme finito X, un *insieme pesato (su X)* è una funzione $w\colon X \to \mathbb{R}$ che associa a ogni elemento di X il suo *peso*, che è un valore reale.
Si vuole rappresentare un insieme pesato w di caratteri come una lista ordinata contenente un nodo per ogni carattere che ha peso non nullo in w. Ogni nodo contiene un carattere e il suo peso (un `double`), oltre al puntatore al nodo successivo. La lista deve essere ordinata *per peso crescente degli elementi*.
N.B.: Si considerino diversi i caratteri minuscoli e maiuscoli.

(i) Si definiscano i tipi di dati necessari per implementare in `C` la rappresentazione indicata. Si chiami `WeightedSet` il tipo di dati principale.

```
typedef struct node {
  char el;
  double peso;
  struct node * next;
} NODE;
typedef NODE *WeightedSet;
```

Si definiscano quindi le seguenti funzioni o procedure C che operano su oggetti di tipo `WeightedSet`, rispettando il prototipo proposto:

(ii) `double weight (WeightedSet w, char c);`
Restituisce il peso del carattere `c` nell'insieme pesato `w`. Questa funzione *deve* essere definita in modo ricorsivo.

```
double weight (WeightedSet w, char c) {
  if(w==NULL) return 0;
  if(w->el==c) return w->peso;
  else return weight(w->next, c);
}
```

(iii) `boolean remove (WeightedSet *p, char c);`
Elimina il nodo con il carattere `c` dall'inisieme pesato puntato da `*p`, se tale nodo esiste: in questo caso restituisce `true`. Altrimenti non modifica l'insieme pesato e restituisce `false`.

```
boolean remove (WeightedSet *p, char c) {
  WeightedSet temp;
  if(p!=NULL) {
    if(*p==NULL) return false;
    else {
      if((*p)->el == c) {
        temp = *p;
        *p = (*p)->next;
        free(temp);
        return true;
      }
```

```
      else {
        return remove(&((*p)->next), c);
      }
    }
  }
  return false;
}
```

(iv) `void addWeight (WeightedSet *p, char c, double r);`
Aggiunge il peso `r` al carattere `c` nell'insieme pesato puntato da `p`. Quindi se il carattere era presente in un nodo, ne aumenta il peso di `r`, spostando il nodo per preservare l'ordinamento, se necessario, oppure eliminando il nodo se il peso risultante è nullo. Altrimenti inserisce un nuovo nodo nella lista (rispettando l'ordinamento) con il carattere `c` e con peso `r`. La procedura non modifica l'insieme pesato se $r = 0$.[1]

```
/* Metodo ausiliario per inserire un nodo nella lista in modo ordinato */
void insert(WeightedSet *p, char c, double r) {
  WeightedSet temp;
  NODE* nuovo;
  if(p!=NULL) {
    if(*p!=NULL) {
      temp = *p;
      if(weight(*p,c)==0) {
        if(temp->peso<=r) {
          nuovo = malloc(sizeof(NODE));
          nuovo->el = c;
          nuovo->peso = r;
          nuovo->next = temp;
          *p = nuovo;
        }
        else {
          while(temp->next!=NULL && temp->next->peso > r)
            temp = temp->next;
          nuovo = malloc(sizeof(NODE));
          nuovo->el = c;
          nuovo->peso = r;
          nuovo->next = temp->next;
          temp->next = nuovo;
        }
      }
    }
    else {
      nuovo = malloc(sizeof(NODE));
      nuovo->el = c;
      nuovo->peso = r;
      nuovo->next = NULL;
      *p = nuovo;
    }
  }
}

/* Soluzione piu' semplice, che modifica la lista
   anche se non strettamente necessario */
void addWeight (WeightedSet *p, char c, double r) {
  double newweight;
  /* se r==0 la lista non va modificata */
  if(r==0) return;
  /* calcolo il nuovo peso per il carattere c */
  newweight  = weight(*p, c) + r;
  /* elimino il nodo con c, se esiste */
```

[1] Per un refuso, il testo riportava "se $r \leq 0$": le corrispondenti soluzioni sono state considerate corrette.

```
  remove(p, c);
  /* inserisco il nodo con c, con il nuovo peso,
     se quest'ultimo e' diverso da zero */
  if (newweight != 0)
    insert(p, c, newweight);
}

/* Soluzione piu' complessa, che modifica la lista
   solo se strettamente necessario */
void addWeight (WeightedSet *p, char c, double r) {
  double newweight, oldweight;
  WeightedSet temp;
  if(r==0) return; /* se r==0 la lista non va modificata */
  oldweight = weight(*p, c);
  if(oldweight == 0) {
    /* se il carattere non era nella lista, lo inserisco in modo ordinato */
    insert(p, c, r);
    return;
  }
  newweight = oldweight + r;
  if(newweight==0){
    /* se il nuovo peso e' zero, cancello l'elemento dalla lista */
    remove(p, c);
    return;
  }
  if ((*p)-> el == c){ /* il carattere c e' nel primo elemento */
    if ((*p)-> next == NULL || (*p)->next->peso >= newweight){
      /* se la lista rimane ordinata con il nuovo peso */
      (*p)->peso = newweight;
      return;
    } else {
      /* se la lista NON rimane ordinata con il nuovo peso */
      remove(p, c); /* tolgo il carattere */
      insert(p, c, newweight); /* lo reinserisco con il nuovo peso */
      return;
    }
  }
  /* cerco l'elemento della lista con il carattere c, mantenendo un
     puntatore all'elemento precedente */
  temp = *p;
  while (temp->next->el != c)
    temp = temp->next;
  if (temp->peso <= newweight &&
      (temp->next->next == NULL || newweight <= temp->next->next->peso)){
    /* se la lista rimane ordinata con il nuovo peso */
    temp->next->peso = newweight;
    return;
  } else {
    /* se la lista NON rimane ordinata con il nuovo peso */
    remove(p, c); /* tolgo il carattere */
    insert(p, c, newweight); /* lo reinserisco con il nuovo peso */
    return;
  }
}
```

(v) `WeightedSet union (WeightedSet w1, WeightedSet w2);`
Senza modificare gli insiemi pesati `w1` e `w2`, questa funzione restituisce l'insieme pesato ottenuto come *unione* dei due argomenti, cioè l'insieme pesato in cui ogni carattere `c` ha come peso la somma dei pesi in `w1` e `w2`.

```
WeightedSet union (WeightedSet w1, WeightedSet w2) {
  WeightedSet res = NULL;
```

```
  while(w1!=NULL) {
    insert(&res, w1->el, w1->peso);
    w1=w1->next;
  }
  while(w2!=NULL) {
    addWeight(&res, w2->el, w2->peso);
    w2=w2->next;
  }
  return res;
}
```

N.B.: Nello svolgimento di questo esercizio **non** si può fare riferimento a funzioni/procedure viste a lezione o a esercitazione.