INFORMATICA 2010/11 - CdL in FISICA

PRIMO APPELLO – 21/06/2011: 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
Scrivere un programma C che legge una sequenza di numeri interi fornita dall'utente e ne stampa il massimo, il minimo e la media aritmetica (che in generale può essere un numero decimale). La sequenza termina quando l'utente immette uno zero, che è considerato far parte della sequenza. 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; in questo caso la procedura o funzione può utilizzare istruzioni di input/output per interagire con l'utente.

Esempio di soluzione non ricorsiva:

```
int main(){
  int next, min = 0, max = 0, count = 0, sum = 0;
  double media;
  do{
    printf("Dammi un numero intero, 0 per terminare\n");
    scanf("%d", &next);
    if (next < min) min = next;
    if (next > max) max = next;
    count++;
    sum += next;
  }while (next != 0);
  media = sum / (double) count;
  printf("Massimo: %d; Minimo: %d; Media: %f \n",max,min,media);
  return 0;
}
```

Esempio di soluzione ricorsiva:

```
int readNext(int *min, int *max, int *count);
/* restituisce la somma dei numeri letti, modificando
   le variabili ricevute come parametri attuali */

int main(){
  int min = 0, max = 0, count = 0, sum;
  double media;
  sum = readNext(&min, &max, &count);
  media = sum / (double) count;
  printf("Massimo: %d; Minimo: %d; Media: %f \n",max,min,media);
  return 0;
}

int readNext(int *min, int *max, int *count){
  int next;
  printf("Dammi un numero intero, 0 per terminare\n");
  scanf("%d", &next);
```

```
  if (next < *min) *min = next;
  if (next > *max) *max = next;
  (*count)++;
  if (next==0) return next;
  else return next + readNext(min, max,  count);
}
```

ESERCIZIO 2
Scrivere una funzione C che ha come parametri formali un array di caratteri e la sua dimensione, e restituisce la posizione del primo carattere che compare almeno tre volte nell'array, ignorando la differenza tra maiuscole e minuscole. Se nessun carattere compare almeno tre volte, la funzione restituisce -1.

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;
}

int firstPosition(char * arr, int dim){
  int i = 0, j, count;
  boolean trovato = false;
  while (i < dim && !trovato){
    count = 0;     /* conta ripetizioni di arr[i] */
    j = i+1;       /* scandisce arr a partire da i */
    while (j < dim && count != 2){
      if (m(arr[i]) == m(arr[j])) count++;
      j++;
    }
    if(count == 2) trovato = true;
    i++;
  }
  if (trovato) return i-1;
  return -1;
}
```

ESERCIZIO 3
Sia data la seguente procedura C:

```
void foobar(int x, int *y){
    x = *y;
    *y = *(y+1);
    *(y+1) = x;
}
```

Sia dato inoltre il seguente `main`:

```
int main(){
    int arr[] = {11, 22, 33, 44};
    foobar(arr[0], &arr[1]);
    foobar(arr[3], arr + 2);
}
```

1. Spiegare cosa fa la procedura `foobar`.
2. Indicare il contenuto dell'array `arr` dopo ognuna delle tre istruzioni del corpo del `main`, giustificando le risposte.
3. Cosa succede se si aggiunge come ultima istruzione del `main` la seguente?

```
    foobar(arr[3], &arr[3]);
```

Esempio di soluzione:

1. La procedura scambia il contenuto di due variabili: quella riferita dal puntatore passato come secondo argomento, e quella successiva in memoria, indirizzata sfruttando l'artimentica dei puntatori (`y+1`). Il primo argomento, passato per valore, non viene utilizzato.

2. Il contenuto dell'array dopo ogni istruzione è il seguente:

```
int main(){
    int arr[] = {11, 22, 33, 44};  /* [11, 22, 33, 44] */
    foobar(arr[0], &arr[1]);       /* [11, 33, 22, 44] */
    foobar(arr[3], arr + 2);       /* [11, 33, 44, 22] */
}
```

3. Nell'istruzione aggiunta alla fine la procedura accede a una variabile oltre la fine dell'array, pertanto il valore di `arr[3]` non è prevedibile.

ESERCIZIO 4

Dato un insieme finito X, un *multinsieme su* X è una funzione $m\colon X \to \mathbb{N}$ che associa a ogni elemento di X la *molteplicità* con cui compare nel multinsieme.
Si vuole rappresentare un multinisieme m di caratteri come una lista contenente un nodo per ogni carattere che ha molteplicità non nulla in m. Ogni nodo contiene un carattere e la sua molteplicità, oltre al puntatore al nodo successivo.
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 `MSet` il tipo di dati principale.

```
typedef struct node {
   char el;
   int mult;
   struct node * next;} NODE;
typedef NODE *MSet;
```

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

(ii) `int mult (MSet m, char c);`
Restituisce la molteplicità del carattere `c` nel multinsieme `m`. Questa funzione *deve* essere definita in modo ricorsivo.

```
int mult (MSet m, char c){
  if (m == NULL) return 0;
  if (m->el == c) return m->mult;
  return mult(m->next,c);
}
```

(iii) `void insert (MSet *p, char c, int k);`
Inserisce il carattere `c` con molteplicità `k` nel multinsieme puntato da `p`. Quindi se il carattere era già presente in un nodo, ne aumenta la molteplicità di `k`, altrimenti inserisce un nuovo nodo nella lista con il carattere `c` e con molteplicità `k`. La procedura non modifica il multinsieme se $k \leq 0$.

```
void insert (MSet *p, char c, int k){
  MSet aux;
  if (k > 0){
    if (mult(*p,c) == 0){ /* non c'e' c */
      aux = malloc(sizeof(NODE));
```

```
      aux->el = c;
      aux->mult = k;
      aux->next = *p;
      *p = aux;
    } else {
      aux = *p;
      while (aux->el != c)
        aux = aux->next;
      aux->mult += k;
    }
  }
}
```

(iv) `void minus (MSet *m1, MSet m2);`
Modifica il multinsieme `m1` sottraendo `m2`, senza modificare `m2`. Dopo l'esecuzione della procedura, `m1` sarà il multinsieme definito come

$$(m1 - m2)(x) = \begin{cases} m1(x) - m2(x) & \text{se } m2(x) < m1(x) \\ 0 & \text{altrimenti} \end{cases}$$

```
void minus (MSet *m1, MSet m2) {
  MSet aux = *m1, prec = NULL, temp;
  while(aux != NULL) {
    aux->mult -= mult(m2, aux->el);
    if(aux->mult<=0) { /* il nodo deve essere eliminato dal multinsieme */
      if(prec != NULL) /* agganciamo il precedente al successivo di aux */
        prec->next = aux->next;
      else /* oppure aggiorniamo la testa */
        *m1 = aux->next;
      temp = aux->next;
      free(aux);
      aux = temp;
    }
    else {
      prec = aux;
      aux = aux->next;
    }
  }
}
```

(v) `MSet add (MSet m1, MSet m2);`
Senza modificare i multinsiemi `m1` e `m2`, questa funzione restituisce il multinsieme ottenuto come *somma* dei due argomenti, cioè il multinsieme in cui ogni carattere `c` ha come molteplicità la somma delle molteplicità in `m1` e `m2`.

```
MSet add (MSet m1, MSet m2){
   MSet m = NULL;
   while (m1 != NULL){
      insert(&m, m1->el, m1->mult);
      m1 = m1->next;
   }
   while (m2 != NULL){
      insert(&m, m2->el, m2->mult);
      m2 = m2->next;
   }
   return m;
}
```

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