INFORMATICA 2010/11 - CdL in FISICA

QUARTO APPELLO – 19/09/2011

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 chiede all'utente un intero positivo k, e quindi chiede una sequenza di interi che termina quando la somma dei valori assoluti dei numeri letti supera k (né k né l'ultimo numero letto fanno parte della sequenza). Alla fine il programma stampa la media aritmetica, il minimo e il massimo dei valori 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 (che naturalmente può utilizzare istruzioni di input/output per interagire con l'utente).

ESERCIZIO 2
Una stringa è *bilanciata* se contiene lo stesso numero di caratteri alfabetici (minuscoli o maiuscoli) e di cifre decimali (0,1, ..., 9). Una stringa è *alternata* se è bilanciata e in essa ogni cifra decimale è seguita da un carattere alfabetico e viceversa, ogni carattere alfabetico è seguito da una cifra decimale.

Scrivere una funzione C (ed eventuali funzioni ausiliarie) che ha come unico parametro formale una stringa, e restituisce 0 se la stringa non è bilanciata, 1 se è bilanciata ma non alternata, e 2 se è alternata. Per esempio, la funzione deve restituire 0 per le seguenti stringhe: "`44 gatti`", "`soloCaratteri`", "`342`"; deve restituire 1 per le seguenti: "`19 settem.  2011`", "`3a2e4a.`"; e infine 2 per le seguenti: "`4h3j2k5k`", "`U2`", "`...C6H2K6`".

Indicare in un commento gli schemi di programmi utilizzati.

ESERCIZIO 3
Sia data la seguente funzione ricorsiva C:

```
void foobarfoo(int a[], int * b, int dima, int dimb){
  int p;
  if (dima == 0 || dimb == 0) return;
  p = *a;
  a[0] = *b;
  b[0] = p;
  /*  ===> STATO DELLA PILA <=== */
  foobarfoo(b+1,&a[1],--dimb, dima-1);
}
```

Sia dato inoltre il seguente `main`:

```
int main(){
  int a [] = {1,2,3};
  int b [] = {10,20};
  foobarfoo(a,b,3,2);
}
```

1. Mostrare lo stato della pila (e in particolare il contenuto degli array a e b) per ogni chiamata della funzione `foobarfoo` nel punto indicato.
2. Spiegare cosa fa, intuitivamente, la funzione `foobarfoo` quando viene invocata su due array di lunghezza n e m, per $n = m$ e per $n \neq m$.

ESERCIZIO 4

Una *tabella hash* è una struttura dati usata comunemente in programmazione per memorizzare e gestire insiemi di dati in modo efficiente. Si vuole scrivere un insieme di funzioni e procedure che realizzano le operazioni basiche di una semplice tabella hash per memorizzare stringhe.

La tabella è un array di liste concatenate, i cui elementi sono le stringhe. Per ogni stringa da inserire nell'insieme si determina l'indice della lista in cui inserirla con una *funzione hash*. Nel caso specifico, l'indice sarà determinato così: se la tabella ha `DIM` liste, la stringa `str` verrà inserita in quella di indice `strlen(str) % DIM`.

Le definizioni dei tipi di dato sono le seguenti:

```
#define DIM 10  % numero di liste nella tabella hash

typedef struct el {
  char * str;
  struct el * next;
} Nodo;

Nodo * Tabella [DIM];
```

Si definiscano le seguenti funzioni C che operano su oggetti di tipo `Tabella`, rispettando il prototipo proposto:

(i) `boolean member(char * str, Nodo * Tabella []);`
Restituisce `true` se `str` è un elemento della tabella, `false` altrimenti.

(ii) `boolean add(char * str, Nodo * Tabella []);`
Inserisce la stringa `str` nella tabella, solo se non era già presente. Restituisce `true` se l'inserimento ha avuto sucesso, `false` se la stringa era già presente.

(iii) `boolean del(char * str, Nodo * Tabella []);`
Cancella la stringa `str` dalla tabella. Restituisce `true` se la cancellazione ha avuto sucesso, `false` altrimenti.

(iv) `Nodo * elements(Nodo * Tabella []);`
Restituisce una lista contenente tutte le stringhe presenti nella tabella. La tabella non deve essere modificata.

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