INFORMATICA 2010/11 - CdL in FISICA

SECONDO APPELLO – 25/07/2011: 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 calcola e stampa due valori: la somma di tutti gli elementi pari e maggiori di zero, e la somma di tutti gli elementi dispari e minori di zero. La sequenza termina quando l'utente inserisce due numeri uguali consecutivi, che sono considerati entrambi 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 pariPos=0, dispariNeg=0;
  int n=0, prev=0;
  int i=0;

  do {
    i++;
    prev = n;
    printf("Inserisci un numero: ");
    scanf("%d", &n);
    if(n > 0 && n % 2 == 0) pariPos += n;
    if(n < 0 && n % 2 != 0) dispariNeg += n;
  } while(i < 2 || prev != n);

  printf("La somma dei valori inseriti pari e maggiori di 0 e': %d\n", pariPos);
  printf("La somma dei valori inseriti dispari e minori di 0 e': %d\n", dispariNeg);
  return 0;
}
```

Esempio di soluzione ricorsiva:

```
void pariPosMaggioredispariNeg(int* pariPos, int* dispariNeg, int last, boolean primo) {
  int n;
  printf("Inserisci un numero: ");
  scanf("%d", &n);
  if(n > 0 && n % 2 == 0) *pariPos += n;
  if(n < 0 && n % 2 != 0) *dispariNeg += n;

  if(primo || n!=last)
    pariPosMaggioredispariNeg(pariPos, dispariNeg, n, false);
  return;
}
```

```
int main() {
  int pariPos=0, dispariNeg=0;
  printf("Inizio inserimento.\n");
  pariPosMaggioredispariNeg(&pariPos, &dispariNeg, 0, true);
  printf("La somma dei valori inseriti maggiori di 0 e': %d\n", pariPos);
  printf("La somma dei valori inseriti minori di 0 e': %d\n", dispariNeg);
  return 0;
}
```

ESERCIZIO 2

Scrivere una funzione C che ha come parametro formale una stringa, e restituisce il carattere che compare più frequentemente in essa. Il programma deve ignorare la differenza tra caratteri alfabetici maiuscoli e minuscoli, e nel caso più caratteri compaiano con frequenza massima deve restituirne il maggiore secondo l'ordinamento naturale tra caratteri. Se la stringa non ha caratteri, la funzione deve restituire il terminatore di stringa.

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

```
char toMin(char c) {
  if(c >= 'A' && c <= 'Z') return c - 'A' + 'a';
  return c;
}
char maxfreq(char* s) {
  char* start = s;
  char* cur = start;
  int amount = 0;
  char res = toMin(*start);
  int max = 0;

  /* Scorro a partire dall'inizio fino al terminatore */
  while(*start != '\0') {
    cur = start;
    amount = 0;
    /* Il cursore confronta i caratteri successivi con quello in esame */
    while(*cur != '\0') {
      if(toMin(*cur) == toMin(*start)) amount++;
      cur++;
    }
    /* Se la frequenza e' maggiore, aggiorno il risultato parziale */
    if(amount>max || (amount==max && toMin(*start) > res)) {
      max = amount;
      res = toMin(*start);
    }
    /* esamino il successivo carattere */
    start++;
  }
  return res;
}
```

ESERCIZIO 3

Sia data la seguente funzione C:

```
int barfoo(int arr[], int dim){
  int res;
  if (dim <= 0) return 0;
  res = *arr;
  *arr = barfoo(arr + 1, dim - 1);
  return res;
}
```

Sia dato inoltre il seguente `main`:

```
int main(){
  int dim = 4;
  int arr [] = {101,202,303,404};
  * (arr + dim - 1) =  barfoo(arr, dim);
  return 0;
}
```

1. Spiegare cosa fa la funzione `barfoo`.
2. Indicare il contenuto dell'array `arr` alla fine dell'esecuzione del main, giustificando la risposta.

Esempio di soluzione:

1. La procedura barfoo riceve un array e la sua dimensione e si salva il primo elemento *arr; quindi chiama ricorsivamente la stessa procedura sul sotto-array a partire dal secondo elemento, salvando il risultato nella prima posizione. Alla fine ritorna il primo elemento precedentemente salvato.

   Questo, in sostanza, ha l'effetto di uno shift verso sinistra dell'array, con il primo elemento che viene ritornato al chiamante e con l'ultimo elemento che viene sostituito da 0 (vista la condizione di terminazione della ricorsione).

2. La sua applicazione nel main ha l'effetto di sostituire l'elemento restituito, che era il primo dell'array originale, all'ultimo, che era stato rimpiazzato con 0. Nel complesso quindi l'array subisce uno shift rotatorio a sinistra e il contenuto dell'array alla fine dell'esecuzione del main e' {202,303,404,101}.

ESERCIZIO 4

Il *grafico* di una funzione definita sugli interi e a valori reali è rappresentato da una lista ordinata di *punti*, cioè di record aventi tre campi: un intero x (ascissa), un double y (ordinata), e un campo *next* che punta al prossimo punto. Nel grafico i punti sono ordinati per ascissa crescente, e non ci possono essere due punti aventi la stessa ascissa. Se fun è la funzione rappresentata dal grafico gr, la presenza del punto (x, y, p) in gr indica che $fun(x) = y$.

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

```
struct punto {
  int x;
  double y;
  struct punto * next;
};
typedef struct punto Punto;
typedef Punto * Grafico;
```

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

(ii) `void update(Grafico * fun, int argx, double argy);`
Aggiorna il grafico passato per argomento aggiungendo un punto con ascissa $argx$ e ordinata $argy$. Se $*fun$ aveva già un punto con ascissa $argx$, ne viene solo modificato il campo y con il nuovo valore. Altrimenti si inserisce un nuovo punto nel grafico, preservando l'ordinamento.

```
void update(Grafico* fun, int x, double y) {
  Punto* p;
  Grafico f;
  /* questo serve solo come precauzione, non dovrebbe succedere mai. */
  if(fun!=NULL) {
    f = *fun;
    /* se il grafico e' vuoto o il primo elemento e' piu' grande di quello passato,
    attacchiamo il nuovo elemento in testa */
    if(f==NULL || f->x > x) {
      p = (Punto*) malloc(sizeof(Punto));
      p->x = x;
```

```
      p->y = y;
      p->next = f;
      *fun = p;
    }
    else {
      /* altrimenti scorriamo il grafico fino a che non arriviamo al punto di inserimento */
      while(f->next!=NULL && f->next->x < x ) {
        f = f->next;
      }
      /* se x era gia' presente, aggiorniamo il valore, altrimenti inseriamo il nuovo elemento */
      if(f->next!=NULL && f->next->x == x) {
        f->next->y = y;
      }
      else {
        p = (Punto*) malloc(sizeof(Punto));
        p->x = x;
        p->y = y;
        p->next = f->next;
        f->next = p;
      }
    }
  }
}
```

(iii) `double simpleEval(Grafico fun, int arg);`
Restituisce il valore della funzione rappresentata da fun per l'argomento intero arg. Restituisce 0 se non esiste in fun un punto con ascissa arg. Questa funzione DEVE essere definita in modo ricorsivo.

```
double simpleEval(Grafico fun, int arg) {
  if(fun==NULL) return 0;
  if(fun->x == arg) return fun->y;
  return simpleEval(fun->next, arg);
}
```

(iv) `double length(Grafico fun);`
Restituisce la *lunghezza* del grafico, visto come una linea spezzata passante per i punti indicati.
NB: Per il calcolo della radice quadrata si può usare la funzione con prototipo `double sqrt(double)` della libreria `math.h`.

```
double length(Grafico fun){
  int dx;
  double dy;
  if (fun == NULL || fun->next == NULL) return 0;
  dx = fun->next->x - fun->x;
  dy = fun->next->y - fun->y;
  return sqrt(dx * dx + dy * dy) + length(fun->next);
}
```

(v) `double eval(Grafico fun, int arg);`
Come *simpleEval*, ma considera la funzione definita su tutti gli interi compresi tra l'ascissa del primo punto e quella dell'ultimo. Quindi:

- Se arg cade fuori dall'intervallo indicato, restituisce 0.
- Se fun ha un punto con ascissa x, restituisce la corrispondente ordinata.
- Altrimenti restituisce il valore ottenuto interpolando linearmente i valori assunti dalla funzione per i valori immediatamente minore e immediatamente maggiore di arg.

```
double eval(Grafico fun, int arg){
  Punto * aux = NULL;
  /* se il grafico e' vuoto o siamo fuori dal margine sinistro, il risultato e' 0 */
```

```
  if (fun == NULL || arg < fun->x) return 0;
  /* cerchiamo il valore tenendoci anche il precedente, aux */
  while (fun->x < arg && fun->next != NULL){
    aux = fun;
    fun = fun->next;
  }
  /* se siamo fuori dal margine destro, il risultato e' 0 */
  if (fun->x < arg) return 0;
  /* se abbiamo il valore preciso, ottimo */
  if (fun->x == arg) return fun->y;
  /* altrimenti ritorniamo il valore interpolato dalla retta
     che unisce i due punti, presa all'altezza di arg */
  return (( fun->y - aux->y )/( fun->x - aux->x )) * (arg - aux->x) + aux->y;
}
```

(vi) `Grafico saturate(Grafico fun);`
Crea e restituisce un nuovo grafico ottenuto da quello passato per argomento "riempiendo i buchi", cioè inserendo un punto per ogni valore delle ascisse compreso nell'intervallo di definizione della funzione per cui non esisteva, con il valore che si sarebbe ottenuto dalla funzione *eval* in quel punto. Il grafico restituito non deve avere alcun struct in comune con quello passato per argomento.

```
Grafico saturate(Grafico fun){
  Grafico res=NULL, f=fun;
  Punto *p, *prev=NULL;
  int x;
  double y;
  int expected;

  if(f!=NULL) {
    /* expected e' il prossimo valore di x che vogliamo:
       se c'e', bene, se non c'e' lo calcoliamo.
       All'inizio, ovviamente, vogliamo il primo. */
    expected = f->x;
    while(f!=NULL) {
      while(expected <= f->x) {
        /* i valori del nuovo punto, gia' presenti o interpolati */
        x = expected++;
        y = eval(fun, x);
        /* il nuovo punto */
        p = (Punto*) malloc(sizeof(Punto));
        p->x = x;
        p->y = y;
        p->next = NULL;
        /* il risultato finale e' il primo punto */
        if(res==NULL) res = p;
        /* e poi colleghiamo il punto al precedente, se c'e' */
        if(prev!=NULL)
          prev->next = p;
        prev = p;
      }
      /* abbiamo raggiunto f->x quindi possiamo andare avanti */
      f = f->next;
    }
  }
  return res;
}
```

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