Programmazione - Compito del 22/4/2022

Istruzioni

- prima di iniziare, scrivete nome, cognome e matricola su *ogni foglio*. Se state sostendo un'integrazione, indicatelo sotto la matricola (es. integrazione 6 crediti).
- Dall'inizio della prova avete a disposizione un'ora (per quelli di voi che hanno diritto al 30% di tempo aggiuntivo 1:18h, 30 minuti per chi fa l'integrazione da 6-7 crediti) per consegnare il compito alla cattedra. Consegne in ritardo non verranno accettate. Una volta consegnato, non potete uscire dall'aula fino al termine del turno. Se state facendo un'integrazione, dovete completare solamente gli esercizi con indicato (integrazione $\geq$ x crediti).
- Potete usare solamente penne, matita e gomma. Nient'altro può essere presente sul banco. Non potete usare fogli aggiuntivi, nemmeno per la brutta copia. Appoggiate zaini e giacche a lato della stanza. Scrivete in modo leggibile, parti non comprensibili non verranno corrette.
- Vi chiedo di non parlare e non alzare lo sguardo dal vostro foglio. Una volta finito il compito non utilizzate il cellulare. Visto che siete seduti vicini gli uni agli altri, sarò intransigente, segnandomi la minima trasgressione e riservandomi successivamente come agire.

Cognome ..... Nome ..... Matricola .....

1. Utilizzando l'aritmetica dei puntatori, scrivere la formula per ottenere l'*indirizzo* dell'elemento `C[7][9]` nell'array seguente: **3 punti** (integrazione $\geq$ 6 crediti)

```
int C[18][13];
```

Risposta: C+7*13+9
Se volete provarlo al calcolatore C dovrebbe essere un puntatore ad intero, quindi ad esempio si può eseguire un cast (int *) C

2. Discutere pregi e difetti dell'utilizzo della ricorsione. **2 punti**

Risposta: La ricorsione è costosa sia in termini di memoria occupata (ad ogni invocazione di una funzione si deve aggiungere al record di attivazione i parametri della funzione, il suo punto di ritorno e le variabili locali) che di tempo di esecuzione (la gestione dello stack richiede tempo). La ricorsione può essere preferibile quando non ci sono evidenti vincoli di efficienza del codice e la soluzione ricorsiva è molto più leggibile dell'equivalente soluzione iterativa.

3. Se compiliamo ed eseguiamo il seguente codice, qual è il risultato (cosa viene stampato se non ci sono errori)? Motivare la risposta. **4 punti** (integrazione $\geq$ 6 crediti)

```
1  #include <stdio.h>
2
3  float x = 3.3;
4
5  int main(void) {
6      int y = 4;
7      {
8      if (x<3)
9          if (y>4)
10             printf("%d ", y);
11     else
12         printf("%f ", x);
13         int y = 5;
14         {
15             int z = x * 3;
16             printf("%d", z);
17         }
18     }
19 }
```

Risposta: il codice stampa 9.
Il comando else nella riga 11 si rifersice all'if della riga 9 (vedi diapositiva 19 lezione 2), quindi, poiché la condizione in riga 8 è falsa, le righe 8-12 non vengono eseguite. Nella riga 15 alla variabile intera z viene assegnato il valore 3.3*3=9.9, dopo aver eseguito una conversione automatica da float ad intero: la conversione da float ad intero elimina la parte frazionaria del numero (.9). Infine la riga 16 stampa z dopo la conversione, ovvero 9.

Cognome ..... Nome ..... Matricola .....

4. Aggiungere una sola istruzione di assegnamento nella funzione sub in modo che, eseguendo il frammento di codice seguente, venga stampato "progr". Se ciò non è possibile, spiegare il perché. Notate che l'assegnamento `s="progr";` all'interno di `sub` renderebbe il codice non compilabile. **3 punti**

```
1  void sub(char *s) {
2      s ... ;
3  }
4  char s[20] = "programmazione";
5  sub(s);
6  printf("%s", s);
```

Risposta: la riga 2 diventa s[5]='\0';
La printf nella riga 6 stampa una stringa fino a quando incontra il carattere \0.

5. Scrivere una funzione che, dato un array `A` di `dim` interi ordinati in modo decrescente ed un intero `x`, determini se esiste una coppia di indici (i,j) in `A`, con i≠j per cui $A[i] + A[j] = x$. Nel main deve essere possibile stampare tale coppia di indici oppure (-1, -1) se $A[i] + A[j] \neq x$ per ogni (i,j). Vi è richiesto di scrivere solamente la funzione. **8 punti**

```
1  void trova_indici_somma_v1(int A[], int dim, int x, int *i, int *j) {

3      for(*i=0; *i<dim; *i+=1)
4          for(*j=0; *j<dim; *j+=1)
5              if( (*i!=*j) && (A[*i]+A[*j]==x) )
6                  return;
7      *i=-1; *j=-1;
8  }
```

Passare le variabili i,j come puntatori permette di mantere i loro valori dopo l'invocazione della funzione. La funzione controlla tutte le coppie (A[i],A[j]), uscendo dalla funzione non appena ne esiste una con i≠j e A[i]+A[j]=x. Si può evitare di controllare tutte le coppie utilizzando la seguente funzione:

```
1  void trova_indici_somma_v2(int A[], int dim, int x, int *i, int *j) {

3      for(*i=0; *i<dim; *i+=1)
4          for(*j=*i+1; *j<dim; *j+=1)
5              if( (A[*i]+A[*j])==x )
6                  return;
7      *i=-1; *j=-1;
8  }
```

La funzione controlla se la somma di tutte le coppie (A[i],A[j]) con $i < j$, sia uguale a x. Notate che la condizione $i < j$ non è restrittiva: nella funzione precedente, v1, si controllava sia (A[i],A[j]) che (A[j],A[i]), ma poichè la somma è commutativa, basta controllare una delle due coppie. Ciò si ottiene ad esempio ponendo la condizione $i < j$. Notate che così si ottiene anche di non controllare le coppie i=j.

Le soluzioni sopra funzionano indipendentemente dal fatto che l'array sia ordinato oppure no. Si poteva pensare una soluzione che sfruttasse ciò, ve la lascio di seguito per "curiosità":

Cognome ..... Nome ..... Matricola .....

```
1  void trova_indici_somma_v3(int A[], int dim, int x, int *i, int *j) {

3    int somma_coppia;
4    *i=0; *j=dim-1;

6    while (*i < *j) {
7      somma_coppia = A[*i] + A[*j];
8      if (somma_coppia < x)
9        *j -= 1;
10     if (somma_coppia > x)
11       *i += 1;
12     if (somma_coppia == x) {
13       return;
14     }
15   }
16   *i = -1; *j = -1;
17 }
```

Gli indici i,j vengono inizializzati al primo ed all'ultimo elemento di A. Se $A[i] + A[j] < x$ dobbiamo far si che $A[i] + A[j]$ aumenti. Se $i = 0$ allora non possiamo diminuire i; se $i > 0$ allora esiste un $j' > j$ per cui $A[i-1] + A[j'] > x$, ma allora anche $A[i-1] + A[j] > x$, per cui non possiamo aumentare i e non ci rimane che tentare di diminuire j. Un ragionamento simile si applica alle righe 10-11.

6. Il valore di un elemento del triangolo magico è la somma del numero della riga precedente che si trova immediatamente sopra e del numero della stessa riga immediatamente a sinistra, se questi sono definiti (altrimenti è 1).

$$\begin{array}{ccccccc} 1 & & & & & & \\ 1 & 1 & & & & & \\ 1 & 2 & 2 & & & & \\ 1 & 3 & 5 & 5 & & & \\ 1 & 4 & 9 & 14 & 14 & & \end{array}$$

Scrivere una funzione ricorsiva che, dato un indice di riga x ed un indice di colonna y, restituisce il valore (x, y) del triangolo magico. Per esempio `magico(4,2)` restituisce $9 = 5 + 4$. **8 punti** (integrazione ≥ 6 crediti)

```
1  int tartaglia(int x, int y) {

3    if ( (x==0) && (y==0) )
4        return 1;
5    else if ( (y<0) || (x<0) || (y>x) )
6        return 0;
7    else
8        return tartaglia(x-1,y) + tartaglia(x,y-1);

10 }
```

Cognome ..... Nome ..... Matricola .....

7. Data la seguente funzione, scrivere PRE e POST e discuterne la correttezza. **4 punti** (integrazione $\geq$ 6 crediti)

```
1  int f(int m, int n) {
2
3      if(n==m)
4          return m;
5      else
6          return m*f(m-1, n);
7  }
```

Risposta: una possibile soluzione
POST: Calcola $m * (m-1) * (m-2) * \ldots * n$
PRE: $m \geq n \geq 1$
La dimensione del problema è esprimibile come m-n; poiché nella PRE assumiamo $m \geq n$, $m - n \geq 0$ decresce al descrescere di m (poiché n rimane fissato).
Caso base: se $n = m$ allora la post diventa m, per cui è verificata.
Caso induttivo: la POST è verificata per f(m-1,n), ovvero $f(m-1, n) = (m-1) * (m-2) * \ldots * n$, ma allora sostituendo la POST di f(m-1,n) nella riga 6 ottengo $f(m, n) = m * (m-1) * (m-2) * \ldots * n$, per cui POST(f(m-1,n)) $\implies$ POST(f(m,n)) e quindi la POST è verificata per ogni m,n.