Prova scritta di Algoritmi e Strutture Dati (1^o anno)

Ottobre '99

NOTE:

- L'ordine in cui vengono svolti gli esercizi non è rilevante (in altre parole: se avete problemi a svolgerne uno potete passare tranquillamente ai successivi e farlo dopo).
- I punti previsti per ogni esercizio si riferiscono ad uno svolgimento completamente corretto.
- Nei calcoli di complessità (esercizi 5 e 6) non fare conti troppo dettagliati esplicitando tutte le costanti, ma non limitarsi neppure a dare solo il risultato: bisogna spiegare come ci si arriva.

NOME:

COGNOME:

MATRICOLA:

==

Non scrivere qui sotto

Esercizio	Punti previsti	Punti assegnati
1	4	
2	3	
3	3	
4	8	
5	7	
6	5	
Totale	30	

Esercizio 1 (punti 4) Consideriamo il seguente codice C:

```
#include <stdio.h>
#define MAX 10

int A[MAX];
int pp(int x, int y);
void qq(int x);
int ss(int X[MAX]);

main()
{
   int i,j;
   for ( i=0 ; i<MAX ; i++ )
      scanf("%d",&A[i]);
   for ( i=0 ; i<MAX ; i++ )
      qq(A[i]);
   printf( "\n");
   printf( "%d\n", ss(A) );
}

int pp(int x, int y)
{
   int r,i;
   r = 1;
   for (i=1;i<=y;i++)
     r = r*x;
   return(r);
 }

void qq(int x)
{
   x = -x;
   printf("%d ",x);
}

int ss(int X[MAX])
{
   int s,i;
   s = 0;
   for (i=0;i<MAX;i++)
     s = s + pp(-1,i) * A[i];
   return(s);
}
```

Domande:

1. Descrivere in generale cosa fa la funzione pp
2. Dire qual'è l'output per il seguente input: 1 2 3 4 5 6 7 8 9 10

Esercizio 2 (punti 3)

Consideriamo il seguente codice C:

```
#include <stdio.h>

int a = 100;

int ppp (int * x, int * y);

int fff (int x, int y);

main()
{
  int a = 10;
  int b = 2;
  int c;
  c = fff(a,b);
  printf ("%d, %d, %d\n", a, b, c);
  c = ppp(&a, &b);
  printf ("%d, %d, %d\n", a, b, c);
}

int ppp (int * x, int * y)
{ int a;
  a = (*x) * (*y);
  *x = a;
  *y = -a;
  return(2*a);
}

int fff (int x, int y)
{
  return( x + y * a);
}
```

Domanda: qual'è l'output stampato dalle printf?

Esercizio 3 (punti 3)

Consideriamo le espressioni definite induttivamente come segue:

Lett $= \{A, B, C, \ldots, Z\}$, l'insieme delle lettere maiuscole.

- (base) Lett $\subset$ Exp
- (passo 1) a $\in$ Exp $\Rightarrow$ @ a $\in$ Exp
- (passo 2) x, y $\in$ Exp $\Rightarrow$ x % y $\in$ Exp
- (passo 3) p, q $\in$ Exp $\Rightarrow$ (p & q) $\in$ Exp

Domande:

1. (A & B % @ @ C) appartiene ad Exp ?
2. Fornire una rappresentazione ad albero consistente con la definizione induttiva per l'espressione @ X % (Y % (Z & W) & @ K)

Nota: nella rappresentazione ad albero non è necessario che compaiano le parentesi.

Esercizio 4 (punti 8)

Consideriamo il tipo di dato **Dizionario** di coppie di interi implementato mediante albero binario di ricerca (con puntatori).

Sulle coppie di interi si considera l'ordinamento lessicografico definito come segue:

$$(x, y) < (w, z) \Leftrightarrow [(x < w) \text{ or } ((x = w) \text{ and } (y < z))]$$

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice la funzione booleana **Minore** che implementa il confronto tra due coppie secondo l'ordinamento di cui sopra;
3. Utilizzando la funzione al punto precedente implementare la primitiva **Insert** che aggiunge una coppia al dizionario, tenendo presente che non si ammettono duplicati in un insieme;
4. Disegnare l'albero che si ottiene partendo dall'albero vuoto e inserendo tramite la Insert le coppie della sequenza di input:

   (10,20) (2,1) (8,8) (31,8) (0,44) (50,80) (3,3) (18,99) (10,10)

   Segnare la coppia contenuta in ogni nodo dell'albero.
5. Definire in pseudo-codice una procedura di visita che permetta di scrivere in output le coppie contenute nel dizionario ordinate dalla maggiore alla minore (sempre secondo l'ordinamento lessicografico).

Esercizio 5 (punti 7)

Consideriamo il seguente programma in pseudo-codice:

```
main()
{
    aa : array[1..n] of integer;
    i, j, r : integer;
    i ← 1;
    while i ≤ n do
    {
        r ← 0;
        per j = 1 ... i :
            r ← r + aa[j];
        scrivi(r);
        i ← 2 * i;
    }
}
```

Domanda: determinare la complessità del programma in $\Theta()$ in funzione della dimensione n dell'array aa, nell'ipotesi che la procedura "scrivi" funzioni in tempo costante. Per semplicità si può assumere che n sia una potenza di 2.

Esercizio 6 (punti 5)

Consideriamo alberi ternari implementati con puntatori: ogni nodo contiene un campo info di tipo intero e tre campi sx, ctr e dx di tipo albero. Consideriamo la procedura seguente:

procedure swap(t : albero ternario)
{
    $t1$: albero ternario
    if t non è vuoto **then**
    {
        $t1 \leftarrow t->sx$;
        $t->sx \leftarrow t->dx$;
        $t->dx \leftarrow t1$;
        swap($t->sx$);
        swap($t->dx$);
    }
}

Domanda: stimare la complessità di una generica chiamata swap(t) della procedura in $\Theta()$ in funzione del numero dei nodi n dell'albero t