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

Settembre 2000

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.

NOME:

COGNOME:

MATRICOLA:

CORSO:

==

Non scrivere qui sotto

Esercizio	Punti previsti	Punti assegnati
1	4	
2	6	
3	9	
4	6	
5	5	
Totale	30	

Esercizio 1 (punti 4)

Consideriamo il seguente codice C:

```
#include <stdio.h>

int n = 10;

void d();
void p(int *x);
void na(int **x);
void m(int *y);

main()
{
  int n = 4; int *a; int i;

  d();
  na(&a);
  p(a);
  m(&n);
  for (i=0;i<n;i++) printf("%d ",a[i]);
  printf ("\n");
}

void d()
{ scanf("%d",&n); }

void p(int *x)
{
  int i;
  for (i=0;i<n;i++) x[i] = n-i;
}

void na(int **x)
{ int i; int *xx;
  *x = (int *) calloc(sizeof(int),n);
  xx = *x;
  for (i=0;i<n;i++) xx[i]=0;
}

void m(int *y)
{ *y = n/2; }
```

Domanda: Dire qual'è l'output del programma fornendo in input il valore 12

Esercizio 2 (punti 6)

Consideriamo il tipo di dato *Dizionario* di numeri reali implementato con tabella hash **aperta** con B bucket e ordine delle liste a piacere.

Si assuma nota la funzione di hashing

function h(x:real):integer

che dato un valore reale restituisce un indice tra 0 e $B-1$.

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice la primitiva **Empty** che crea una tabella vuota;
3. Definire in pseudo-codice la primitiva **Member** che verifica se un numero è presente in tabella o no;
4. Definire in pseudo-codice la primitiva **Insert** che inserisce un nuovo elemento in tabella (non si ammettono duplicati);
5. Definire in pseudo-codice la primitiva **Delete** che cancella un elemento dalla tabella.

Note:

- Tutti i valori reali possono essere inseriti in tabella: non è lecito usarne nessuno come "tappo";
- Tentare di inserire un elemento già presente in tabella è un'operazione lecita (non è un errore) che non ha alcun effetto sulla tabella;
- Tentare di cancellare un elemento non presente in tabella è un'operazione lecita (non è un errore) che non ha alcun effetto sulla tabella.

Esercizio 2 (punti 9)

Consideriamo il tipo di dato *Successione* di caratteri implementato con liste dinamiche (con puntatori).

Domande:

1. Definire in pseudo-codice i tipi utilizzati per l'implementazione del tipo di dato;
2. Definire in pseudo-codice le primitive **Empty** che crea una successione vuota e **Conc_t** che aggiunge un elemento in testa ad una lista esistente;
3. Definire in pseudo-codice la primitiva **Reverse1** che prende in input una lista e ne produce una nuova (quindi senza modificare quella in input) corrispondente alla successione inversa di quella in input;
4. Definire in pseudo-codice la primitiva **Reverse2** che prende in input una lista e la capovolge (quindi modificare la lista in input).

Esercizio 4 (punti 6)

Consideriamo la funzione seguente in pseudo-codice che opera su successioni di interi positivi o nulli, di lunghezza massima N (dove N è una costante nota). Le successioni sono implementate con array di lunghezza $N+1$, con la convenzione che la fine della successione viene identificata da un elemento dell'array di valore negativo (sempre presente, quindi l'ultima casella dell'array non può essere utilizzata per inserire valori della successione).

```
Type succ = array [0..N] of integer;
procedure coda ( s : succ IN-OUT);
{
    i : integer;
    i ← 0;
    while s[i]≥ 0 do
    {
        s[i] ← s[i+1];
        i++;
    }
}

function is-empty ( s : succ IN-OUT) : boolean
{ return( s[0] < 0 ); }

procedure print-succ ( s : succ IN-OUT)
{
    while not is-empty(s) do
    {
        stampa( s[0] );
        coda( s )
    }
}
```

Domanda: determinare la complessità in $\Theta()$ della chiamata **print-succ(S)** in funzione della lunghezza n della successione S in input (con $n \leq N$).

NOTE:

- I passaggi di parametro si intendono tutti per riferimento;
- Non è necessario fare conti molto dettagliati esplicitando tutte le costanti. Basta considerare il costo di tutte le parti di programma mettendo eventualmente in evidenza le operazioni dominanti (una volta che si sono individuate si possono ignorare le altre).
- Non limitarsi a dare solo il risultato: bisogna spiegare come ci si arriva.

Esercizio 5 (punti 5)

Consideriamo la seguente procedura ricorsiva in C.

```
float fff(float a[]; float x; int inf; int sup)
{
  int i; int med; float m;
  if (inf>=sup)
     return(a[inf]);
  else
       { m = 0.0;
         for (i = inf; i <= sup; i++) m += a[i];
         m = m/(sup-inf+1);
         med = (sup+inf)/2;
         if (m < x) then fff(a,x,inf,med);
         else fff(a,x,med,sup);
       }
}
```

Domanda: Stimare la complessità in $\Theta()$ della chiamata `fff(A,X,1,N)` in funzione di `N`.

NOTE:

- I passaggi di parametro si intendono secondo le regole del C;
- Non è necessario fare conti molto dettagliati esplicitando tutte le costanti. Basta considerare il costo di tutte le parti di programma mettendo eventualmente in evidenza le operazioni dominanti (una volta che si sono individuate si possono ignorare le altre).
- Non limitarsi a dare solo il risultato: bisogna spiegare come ci si arriva.