Esercizi di teoria su alberi e liste

Il seguente materiale è da consultare e studiare con una buona dose di spirito critico.
Simone Magagna

PRE=(R è un albero corretto e possibilmente vuoto)

```
int A(nodo* R)
{
  if(!R) return 0;
  if(R->left && R->right) return A(R->left) + A(R->right);
  if(R->left || R->right) return 1+A(R->left) + A(R->right);
  return 0;
}
```

POST=(restituisce il numero di nodi che hanno esattamnete un figlio)

PRE=(a è un albero corretto e possibilmente vuoto)

```
int B(nodo* a)
{
   if(!a)
   {
      return false;
   }
   if(a->left && a->right)
      return B(a->right) || B(a->left);
   return !(B(a->left) || B(a->right));
}
```

POST=(???)

PRE=(a è un albero non vuoto)

```
int C(nodo* a)
{
   if(!a->left && !a->right) return 0;
   if(!a->left && a->right) return C(a->right);
   if(!a->right && a->left) return C(a->left);
   return 2+C(a->left)+C(a->right);
}
```

POST=(restituisce il doppio del numero dei nodi che hanno entrambi i figli)

PRE=(L è una lista corretta possibilmente vuota)

```
bool D(nodo* L)
{
   if(!L)
      return true;
   return !D(L->next);
}
```

POST=(restituisce true se la lista contiene un numero pari di nodi (0 è considerato pari), false altrimenti)

PRE=(a è un albero corretto possibilmente vuoto)

```
int E(nodo* a)
{
   if(!a) return 0;
   if(!a->left && !a->right) return 1;
   return -1+E(a->left)+E(a->right);
}
```

POST=(restituisce 1 se l'albero è completo, 0 altrimenti)

PRE=(a è un albero corretto e possibilmente vuoto, b è un intero)

```
int F(nodo* a, int b)
{
   if(!b||!a)
   {
      return 0;
   }
   return (a->left!=0)+F(a->left,b-1)+F(a->right,b-1);
}
```

POST=(restituisce il numero di figli sinistri fino a b-esimo livello, se b è maggiore dei livelli dell'albero allora restituisce il numero di figli sinistri dell'intero albero)

PRE=(a è un albero corretto possibilmente vuoto)

```
int G(nodo* a)
{
  if (!a) return 0;
  if (!a->left && a->right) return G(a->right)-1;
  if (!a->right && a->left) return G(a->left)+1;
  return G(a->left)+G(a->right);
}
```

POST=(restituisce 0 se l'abero è bilanciato, altrimenti restituisce la differenza del numero di figli sx con il numero di figli dx)

PRE=(R è un albero corretto e non vuoto)

```
int H(nodo*R)
{
  if(!R->left && !R->right) return 1;
  if(!R->left) return H(R->right);
  if(!R->right) return H(R->left);
  return H(R->left)+H(R->right);
}
```

POST=(restituisce il numero delle foglie dell'albero)