NOTE SULLO SVOLGIMENTO DELLA PROVA SCRITTA:

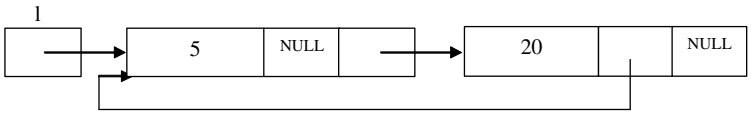
- **SPEGNERE I TELEFONINI;**
- **SCRIVERE IL PROPRIO NOME, COGNOME E NUMERO DI MATRICOLA SU OGNI FOGLIO UTILIZZATO;**
- **NON È POSSIBILE CONSULTARE NESSUN TIPO DI MATERIALE;**
- **NON È POSSIBILE UTILIZZARE CALCOLATRICI;**
- **PRIMA DI SCRIVERE LA SOLUZIONE DELL'ESERCIZIO, INSERIRE IL NUMERO DI ESERCIZIO CHE SI STA RISOLVENDO. PER ESEMPIO, SCRIVERE "ESERCIZIO N. 1" QUANDO SI STA RISOLVENDO L'ESERCIZIO N. 1;**
- NON COPIARE DAL VICINO (NON È DETTO CHE IL VICINO SIA PIÙ BRAVO DI VOI);
- NON PERMETTETE AL VICINO DI COPIARE (È SPIACEVOLE VEDERSI ANNULLARE IL COMPITO SENZA COLPE);
- **I PRIMI TRE ESERCIZI VALGONO 6 PUNTI. I RIMANENTI TRE VALGONO 4 PUNTI.**
- **ALLA FINE DELLA PROVA, RICONSEGNARE TUTTI I FOGLI UTILIZZATI.**
- **SI RICORDA CHE LO SCRITTO VALE PER TRE APPELLI CONSECUTIVI (COMPRESO L'APPELLO IN CUI LO SCRITTO È SOSTENUTO)**

1) Scrivere una funzione ricorsiva che, dato un vettore `v` di stringhe, restituisce `true` se il vettore è ordinato crescente in base alla lunghezza delle stringhe (la stringa in posizione v[i] ha lunghezza <= della stringa in posizione v[i+1]).

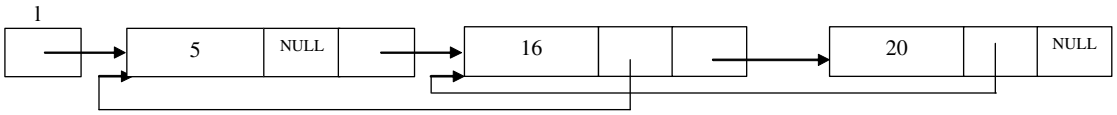
2) Si consideri una lista bidirezionale di elementi di tipo intero implementata con la struttura seguente:

```
struct elem{int info; elem* prec; elem* succ;};
```

Sia `prec` il puntatore all'elemento precedente e `succ` il puntatore all'elemento successivo nella lista.
Scrivere una funzione che, data una lista `l` ordinata in modo crescente di elementi di tipo `elem` ed un intero `n`, inserisce `n` nella lista.
Per esempio, data la lista seguente e l'intero `n`=16,



la lista è modificata come segue:



3) Scrivere una funzione che, data una matrice `mat` di $n \times m$ elementi di tipo intero ed un intero p, restituisce `true` se esistono almeno **2 elementi adiacenti** uguali a p in una riga della matrice, `false` altrimenti.
Per esempio, data la matrice `mat` (3×6), la funzione restituisce `true` se p=1 oppure p=0; la funzione restituisce `false` altrimenti:

```
        0 1 5 1 1 0
mat  =  0 0 0 0 1 0
        2 0 1 0 0 3
```

4) Data la rappresentazione $(751)_8$ in base 8, trasformarla in base 16.
Data la rappresentazione in complemento a due $(10001010)_{compl2}$, esprimere il numero in base 10.

5) Si mostri l'uscita a video del programma C++ seguente:

```
#include <iostream>
using namespace std;

class A
{ protected:
     int x;
  public:
     A(int n = 0) { x=n; cout << "A : x = " << x << endl;}
     void f() { cout << "A::f() x = " << x << endl;}
};

class B: public A
{ protected:
     int x;
  public:
     B(int n=4) { x=n; cout << "B: x = " << x << endl; }
     virtual void f() { cout << "B::f() x = " << x << endl;}
};

class C: public B
{   int x;
   public:
     C(int n=0) : B(n) { x= 10 + n;  cout << "C: x = " << x << endl; }
      void f() { cout << "C::f() x = " << x << endl;}
   };
```

```
template <class T>
void fun(T* p){
p->f();
}

int main(){
   C* pc= new C(5);
   A* pa= pc;
   B* pb= pc;
   fun(pc);
   fun<A>(pc);
   fun<B>(pc);
   return 0;
}
```

6) Si mostri l'uscita a video del programma C++ seguente con ingresso 9, 10, 11.

```
#include<iostream>
using namespace std;

class Ecc{
public:
Ecc(){ cout << "Ecc" << endl;}
virtual void print(){cout << "Ecc" << endl;}
};

class Ecc1: public Ecc {
public:
Ecc1(){cout << "Ecc1" << endl; }
void print(){cout << "Ecc1" << endl;} ;
};

void f(int x){
try{
   cout << "inizio funzione" << endl;
   if (x == 10) {
      throw Ecc1();
   }
   if(x>10) {
       throw new Ecc1();
   }
 cout << "fine try funzione" << endl;
}
catch(Ecc obj){obj.print();}
catch (Ecc* p){p->print();}
cout << "fine f" << endl;
};
```

```
int main()
{  int n;
   cin >> n;
   try {
   f(n);
   cout << "fine try main" << endl;
   }
   catch (...){ cout << "catch generico"  << endl;}
   cout << "fine main" << endl;
   return 0;
}
```