NOME: COGNOME: MATRICOLA:

[1] **Punti 2** – Domanda a risposte multiple (sono possibili 0 o più risposte)
Dati le seguenti dichiarazioni di file:

```
//file1.cc
struct ss{int a;} ;
void fun(ss b) { … }

//file2.cc
struct ss {char c ; int z; };
void fun(ss);
int main() {ss k; fun(k); }
```

a. Il compilatore segnala errore perché rileva due dichiarazioni incompatibili della struttura `ss`
b. Il compilatore segnala errori perché in `file2.cc` non è stata usata la parola chiave `extern` per riferirsi alla dichiarazione di `ss`
c. Il compilatore segnala errore perché l'identificatore `ss` non ha collegamento esterno
d. Nessuna delle precedenti risposte

[2] **Punti 3** – Domanda a risposta aperta
Cosa stampa il seguente programma: _____1 29 1 1 _____

```
void f(int a, int &b, int *&c, int *d) {
     a*=2;
     b+=a;
     c=d;
     *c = 1;}

int main() {
     int a = 7, b = 15;
     int *c = &b;
     int *d = &a;
     f(a, b, c, d);
     cout <<a<< " " <<b<< " " <<*c<< " " <<*d<< " " << endl;}
```

[3] **Punti 3** – Domanda a risposta aperta
Cosa stampa il seguente programma: _____f(4)=3 _____

```
#include <iostream>

using namespace std ;

int f(int);

int main(){
     cout<<"f(4)="<<f(4)<<endl;
     return 0;
}
```

NOME: COGNOME: MATRICOLA:

```
int f(int n){
     if ((n-1)<=0)
        return n;
     return f(n-1)+f(n-2);
}
```

[4] **Punti 3** – Domanda a risposta aperta
Cosa stampa il seguente programma: _____ 1 4 3 6 _____

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

int main()  {

   int v[4]={1,2,3,4};
   int *p = v+1;
   int i;

   *(p+2)=6;
   *p=(*p)+2;
   for(i=0;i<4;i++)
      cout<<v[i]<< " ";
}
```

[5] **Punti 4 -** Scrittura di codice
Data una sequenza di valori interi memorizzati in una lista doppia di elementi di tipo `elem`, si scriva la *funzione* `bool max(elem* e)` che, dato un elemento `e` della lista (la posizione nella lista non è nota), restituisca `true` se il valore registrato in `e` è il valore massimo, `false` altrimenti.

```
struct elem
{
       int inf;
       elem* pun;
       elem* prev;
};

bool max(elem* e){

elem* curr = e->prev;
while(curr!=NULL){
     if(curr->inf>e->inf)
       return false;
     curr=curr->prev;}
curr=e->pun;
whie(curr!=NULL){
     if(curr->inf>e->inf)
       return false;
     curr=curr->pun;}
return true;}
```

NOME: COGNOME: MATRICOLA:

[6] **Punti 8 -** Scrittura di codice
Data la seguente dichiarazione di lista e primitive

```
struct elem
{
        int inf ;
        elem* pun ;
} ;

typedef elem* lista ;

int head(lista); //restituisce il contenuto della testa
lista tail(lista); //restituisce la coda della lista
```

a. **Punti 4**
Si scriva la *procedura ricorsiva* `void stampa_inv(lista)` che, dato in input la testa della lista, stampa l'elenco degli elementi in ordine inverso

```
void stampa_inv(lista l){


  if(tail(l)!=NULL)
       stampa_inv(tail(l));
  cout<<head(l)<<endl;


}
```

b. **Punti 4**
Si scriva la funzione `lista sposta(lista& l, int soglia)` che sposta dalla lista l tutti gli elementi i cui valori sono sotto il valore soglia e restituisce la lista degli elementi spostati. Ad esempio data la lista [1,2,3,4] e il valore di soglia 3 la funzione restituisce la testa della lista [1,2] mentre l diventa [3,4].

```
lista sposta(lista& l, int soglia){
lista l1=NULL, app=l;
lista last_l1=NULL, prev_l=NULL;

while(app!=NULL){
  if(head(app)<soglia){
       if(app==l)
            l=tail(l);
       else
            prev_l->pun=tail(app);
       if(l1==NULL)
            l1=last_l1=app;
       else{
            last_l1->pun=app;
            last_l1=app;}
```

NOME: COGNOME: MATRICOLA:

```
  }
  else
       prev_l=app;
  app=tail(app);}
last_l1->pun=NULL;
return l1;
}
```

[7] **Punti 9 -** Scrittura di codice
Dato la segue dichiarazione

```
struct libro{
  char* autore;
  char* titolo;
};
```

a. **Punti 3**
Si scriva la funzione `int compare(libro, libro)` che implementa la seguente relazione d'ordine:
compare(l1,l2)=0 se autore e titolo coincidono
compare (l1,l2)<0 se l'autore di l1 precede l'autore di l2 e oppure l'autore è lo stesso e il titolo di l1 precede il titolo di l2
compare(e1,e2)>0 altrimenti

```
int compare(libro l1, libro l2){


  if((strcmp(l1.autore,l2.autore)==0 &&
       (strcmp(l1.titolo,l2.titolo)==0)||
       strcmp(l1.titolo,l2.titolo)<0)
       return strcmp(l1.titolo,l2.titolo);
  return strcmp(l1.autore,l2.autore);


}
```

b. **Punti 2** – Scrittura di codice
Si assuma un BST con chiave di tipo `libro`. Scrivere il tipo di dato `bnode` (nodo del BST) e il tipo di dato `bst` (puntatore alla radice del BST)

```
struct bnode {
  libro key;
  bnode* left;
  bnode* right;
  bnode* parent;};

typedef bnode* bst;
```

NOME: COGNOME: MATRICOLA:

c. **Punti 4**
Scrivere la *procedura* `void aggiorna_libro(bst& b,libro* l, char* a)` che aggiorna il contenuto del libro `l` sostituendo l'autore con la stringa contenuta in `a`. La funzione deve usare la funzione `compare` (punto a).

```
void aggiorna_libro(bst& b,libro* l, char* a) {

       bst app=b;
       bnode* new;
      while (app!= NULL) {
        if (compare_key(*l,get_key(app))==0){
          strcpy(l->autore,a);
          new=bst_newNode(*l);
          bst_delete(b,app);
          bst_insert(b,new);
         }
        if (compare_key(k,get_key(app))<0) {
             app = get_left(app);
        } else {
             app = get_right(app);
        }

      }
```

}