NOME: COGNOME: MATRICOLA:

[1] **Punti 2** – Domanda a risposte multiple (sono possibili 0 o più risposte)

**L'espressione `extern int n;`**

a. Non provoca allocazione di memoria
b. Rappresenta una dichiarazione di una variabile intera
c. Definisce la variabile n e la rende accessibile ad altri file (mediante ridichiarazione)
d. Necessita di almeno un file in cui sia definita la variabile n con collegamento esterno

[2] **Punti 3** – Domanda a risposta aperta

Cosa stampa il seguente programma: _____

```
void f(int i, int *p, int &ri, int *&rp) {
int *q = new int ;
*q=20;
*p=++*q;
ri =  30;
i=40;

rp = q;
}

int main() {
  int a = 1, b=2;
  int *punt;
  punt = &b;
  int *prp;
prp = &a;
f(a, punt, b, prp);
cout << a << " " << *punt << " " << b << " " << *prp << " "
<< endl;}
```

[3] **Punti 3** – Domanda a risposta aperta

Cosa stampa il seguente programma: _____

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


int funz(int a)
{
  if (a == 0 || a ==1) return a;
   return funz(a%2)+funz(a/2);}

int main()
{
  cout<< funz(7)<<endl ;
  return 0 ;
}
```

NOME: COGNOME: MATRICOLA:

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

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

int main()  {

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

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

[5] **Punti 4** – Scrittura di codice
Date le seguenti dichiarazioni per un binary search tree con chiave intera, scrivere il codice della primitiva `bst_insert` per l'inserimento di un nodo `n`, assumendo che i nodi siano ordinati in *ordine decrescente* (dal più grande al più piccolo):

```
typedef int tipo_key;

struct bnode {
  int key;
  bnode* left;
  bnode* right;
  bnode* parent;
};

void bst_insert(bst& b, bnode* n){









}
```

NOME: COGNOME: MATRICOLA:

[6] **Punti 4 -** Scrittura di codice
Data una sequenza di valori interi memorizzati in una lista doppia dichiarata sotto, si scriva la *funzione* `int max(elem*)` che dato un elemento della lista (posizione non nota) conta il numero di valori nella sequenza maggiori del valore dell'elemento in ingresso. La funzione deve implementare l'algoritmo con un solo ciclo.

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

int max(elem* e){









}
```

[7] **Punti 13 -** Scrittura di codice
Data la segue dichiarazione

```
typedef char* user;
```

a. **Punti 2** – Scrittura di codice
Si assuma una coda di elementi di tipo `user`. Scrivere il tipo di dato `elem` (elemento della coda) e il tipo di dato `coda`

NOME: COGNOME: MATRICOLA:

b. **Punti 4** – Scrittura di codice
Date due code di utenti c1 e c2 in ordine crescente, scrivere la funzione `coda merge(coda c1, coda c2)` che restituisca una terza coda che contenga i valori delle due code disposti in ordine crescente. La funzione usa il tipo di dato `coda` sopra definito e le seguenti primitive definite sulle code:

```
coda enqueue(coda, user);
user dequeue(coda&);
bool isEmpty(coda);
```

```
coda merge(coda c1, coda c2){

}
```

c. **Punti 2** – Scrittura di codice
Si assuma un grafo di elementi di tipo `user`. Scrivere il tipo di dato `nodo` (elemento del grafo) e il tipo di dato `grafo`

NOME: COGNOME: MATRICOLA:

d. **Punti 5** – Scrittura di codice
Scrivere la *funzione* `bool path(nodo n1, user u)` che restituisce TRUE se esiste un cammino che collega il nodo `n1` con un nodo che ha come contenuto l'utente `u`, FALSE altrimenti.
La funzione usa il tipo di dato `coda` definito su elementi di tipo `nodo` e le seguenti primitive definite sulle code `newQueue`, `enqueue`, `dequeue` e `isEmpty`.

```
bool path(nodo n1, user u){









}
```