1. Record di attivazione A

Scrivi una funzione in **C** che dato in ingresso un intero x e un intero y, restituisce la potenza di x alla y. Scrivi tre versioni: una non ricorsiva, una ricorsiva senza tail recursion e una ricorsiva con tail recursion.

Specifica esattamente i parametri che passi alla procedura, il tipo di passaggio utilizzato e il loro significato.

Scrivi anche un main di esempio in cui chiami la funzione con 3^2 e assegni il risultato ad una variabile locale al main. Non usare alcuna variabile globale.

Disegna il record di attivazione per tutte e tre le versioni fino alla massima estensione del record di attivazione. Nel caso di tail recursion, spiega quali ottimizzazioni hai adottato o potresti adottare.

2. Record Attivazione B

Considera questo codice C:

```
int f(int* k) {
    int x = *k;
    int *y = &x;
    { int a = 0;
      y = &a;
    }
    return *y;
}
```

E' corretto? Scrivi il record di attivazione per una chiamata f(&a) con a variabile intera già sullo stack.

3. Cyclone

A cosa servono i puntatori @zeroterm? Come si usano? Fai un piccolo esempio.

4. Puntatori opachi del C

Come funzionano i puntatori opachi del C? Fai un piccolo esempio.

5. C++

Considera il seguente codice?

```
class Person {
 private:
   int age() { return 1; }
 public:
   int hands() { return 2; }
   virtual int hair() { return 3; }
};

class Student: private Person {
 public:
   int hair() { return 5; }
};

class Professor: public Person {
public:
  int hands() { return 6; }
  int hair() { return 7; }
};
```

```
int main() {
 1.    Person p;
 2.    cout << p.age() << endl;
 3.    cout << p.hands()<< endl;

 4.    Student s;
 5.    cout << s.age() << endl;
 6.    cout << s.hands()<< endl;
 7.    cout << s.hair()<< endl;

 8.    Professor pr;
 9.    cout << pr.age() << endl;
 10.   cout << pr.hands()<< endl;
 11.   cout << pr.hair()<< endl;

 12.   Student * ps = &p;
 13.   cout << ps->hair()<< endl;
```

```
14.   Person * pp = &s;
15.   cout << pp->hair()<< endl;

16.   Person * pp2 = &pr;
17.   cout << pp2->hair()<< endl;

18.   Professor * ppr = &pr;
19.   cout << ppr->hair()<< endl;

20.   Professor * ppr2 = &p;
21.   cout << ppr2->hair()<< endl;

22.   Professor pr3 = p;
23.   cout << pr3.age() << endl;
24.   cout << pr3.hands()<< endl;
25.   cout << pr3.hair()<< endl;

26.   Student s3 = p;
27.   cout << s3.age() << endl;
28.   cout << s3.hands()<< endl;
29.   cout << s3.hair()<< endl;
      }
```

Quale è l'output/effetto prodotto da ogni linea numerata del main? Se contiene un errore scrivi errore, spiega l'errore e ignora la linea (ed eventuali istruzioni che dipendono da essa).

6. Dinamic Binding in Java

Date le seguenti dichiarazioni:

```
class Person{

   public int m(double d){return 1;}

   public int f(double d){return 2;}

}
```

```
class Student  extends Person{

   public int m(int d){return 3;}

   public int f(double d){return 4;}

}
```

```
Object op = new Person();

Person ps = new Student();
```

Quale è il valore delle seguenti tre espressioni spiegando bene (cioè anche il processo di early e late binding dove necessario) il perché (anche se le ritieni errate):

```
op.m(2);

ps.m(2);

ps.m(2.0);

ps.f(2.5);
```

7. Java

Dato il seguente metodo:

```
public void print(Vector<Person> shape) {…}
```

Assumi che Student estende Person. Vector implementa List ed è esteso da Stack. ArrayList implementa List.

Quali di queste chiamate sono corrette? Se non lo sono perchè e come potrei modificare la segnatura del metodo per renderle corrette (se possibile)?

```
1. print(new Vector<Person>());

2. print(new Vector<Student>());
```

```
3. print(new Stack<Person>());

4. print(new Stack<Student>());

5. print(new ArrayList<Person>());

6. print(new List<Person>());

7. print(new ArrayList<Student>());
```

8. Semantica assiomatica

Scrivi un piccolo programma che calcola la potenza di x alla y (vedi esercizio 1), scrivi le precodizioni e le postcondizioni e cerca di dimostrare la correttezza.

9. Testing [6 crediti]

Dato il seguente programma:

```
foo(int x, int y, int z){
     if (x > 2){
        if ((y <x && y >z) || ( y ==5 && z == y+1)){
                z++;
        } else{
                z--;
        }
    }
}
```

Disegna il grafo del programma e trova i casi di test per la copertura delle istruzioni, delle decisioni, delle condizioni e l'MCDC