1. [3 punti] - Data una matrice M1, NxM, che rappresenta un'immagine in memoria come un array contiguo, e un'altra immagine M2, HxK, anch'essa rappresentata come un array contiguo in memoria, correggere i bug della seguente funzione che copia M2 in M1:

```
void BitBlt(unsigned char *dest, int N, int M, unsigned char *src, int H, int K, intx, int y)
{
   for (i = 0; i < K; i++)
   {
      for (j = 0; j < H; j++)
     {
        dest[i + j * N] = src[i * H + j];
      }
   }
}
```

2. [1 punto] - Dato l'esercizio precedente, ottimizzare la funzione di copia.

3. [3 punti] - Data una ipotetica funzione “int pow(int n, int e)” che ritorna una potenza di un numero elevato per un esponente, indicare i pro e i contro di ucon la sare una funzione suddetta firma e una macro tipo “#define pow(n, e)”.

4. [2 punti] - Data la precedente define, indicare cosa accade se eseguo il codice seguente:

```
int n = 4;
int e = 2;
std::cout << pow(++n, e) << std::endl;
```

5. [2 punti] - Che output produce il seguente codice?

```
class A
{
   public:
   A()
   {
       std::cout << 1;
   }

   ~A()
   {
       std::cout << 2;
    }
};

class B : public A
{
    public:
    B()
    {
        std::cout << 3;
    }
```

```
    ~B()
    {
        std::cout << 4;
    }
}

int main()
{
    B b;
    return 0;
}
```

6. [3 punti] – Considerando la seguente definizione di classe, indicare quale costruttore e funzione viene chiamata per ogni riga presente nella funzione main.

```
class Car
{
public:

    Car(float speed, int kw);
    Car(Car &car);
    Car(Car *car);
    Car &operator=(const Car &car);
    virtual ~Car();

protected:
   float m_dSpeed;
   int m_iKw;
};

int main(int argc, const char * argv[])
{
    Car car1(180, 200);
    Car *pCar1 = &car1;
    Car car2 = car1;
    Car *pCar2 = new Car(pCar1);
    Car *pCar3 = pCar1;
    Car car3 = *pCar2;
    pCar1 = pCar3;
    car1 = *pCar1;
    return 0;
}
```

7. [2 punto] - Effettuare il refactoring SOLO dell'interfaccia di questa classe spiegate a lezione: forward declaration, inline, passaggi constantness, passaggio x riferimento, passaggio per valore, etc…

```
#include "Faa.h"

class Fii
{
public:
    Fii();
```

```
    ~Fii();
private:
    float *vec;
};
class Foo : public Fii
{
public:
   Foo();

   void Data(Faa *f);
   int FastSum(int l, int r);
   int Sum(int l, int r);
   int GetNumber();
   const Fii GetObject();

   friend Foo operator+(int a, Foo &f);
   Foo &operator+=(const Foo &f);

   ~Foo();

   private:
   int num;
   Fii *obj;
};
```

8. [1 punto] – Spiegare cosa è un template specializzato e darne un esempio.

9. [1 punto] – Spiegare cosa è una specializzazione parziale di un template e darne un esempio.

10. [2 punto] – Indicare quale funzione, a sinistra, viene invocata dalla chiamata a funzione sotto indicate:

```
int i;
double d;
float ff;
complex<double> c;
```

1. template<typename T1, typename T2> void f(T1, T2);
2. template<typename T> void f(T);
3. template<typename T> void f(T, T);
4. template<typename T> void f(T*);
5. template<typename T> void f(T*, T);
6. template<typename T> void f(T, T*);
7. template<typename T> void f(int, T*);
8. template<> void f<int>(int);
9. void f(int, double);
10. void f(int);

a. f(i);
b. f<int>(i);
c. f(i, i);
d. f(c);
e. f(i, ff);
f. f(i, d);
g. f(c, &c);
h. f(i, &d);
i. f(&d, d);
j. f(&d);
k. f(d, &i);
l. f(&i, &i);

11. [1 punti] - Indicare che tipo di risultato torna la seguente funzione dato un numero positivo o negativo qualunque.

```
int UnknowFunc(int n)
{
    int k = 0;
    while (n)
    {
        k++;
        n /= 10;
    }
    return k;
}
```

12. [1 punti] - Scrivere l'output di questa funzione:

```
void Foo()
{
  for (int i = 4; i > 0; --i)
  {
      std::cout << i << "\t" << (i & 6) << "\t" << (1 << i) << std::endl;
  }
}
```

13. [1 punti] - Implementare una funzione che esegue il reverse dei caratteri di una stringa (es: input "Ciao", output "oaiC") data la seguente firma; la funzione DEVE utilizzare solo memoria sullo stack.

```
void reverse(char *str);
```

14. [1 punti] - Data la precedente funzione, implementarne una versione SENZA l'ausilio dello stack.

15. [1 punto] - Una funzione fattoriale F(n) per interi positivi è definita come:

F(0) = 1
F(n) = 1*2*3*...*(n - 1)*n

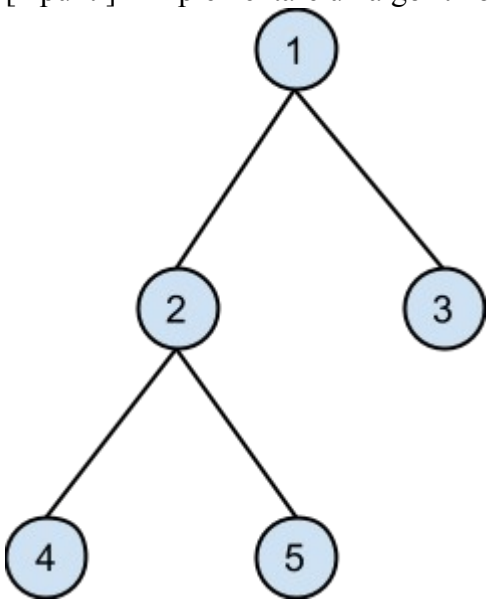
Quindi si ha per esempio che:

F(2) = 1*2 = 2
F(3) = 1*2*3 = 6

```
int factorial(int n);
```

Scrivere quindi la funzione in maniera iterativa, NON ricorsiva, che esegue F(n).

16. [1 punti] - Implementare un algoritmo di navigazione di un binary tree inorder.



La navigazione inorder di un albero binario da come risultato: 4, 2, 5, 1, 3

17. [1 punto] - Implementare un algoritmo che cancelli TUTTI i nodi di un binary tree, navigandoli in order, SENZA usare la ricorsione.
La cancellazione seguirà questa sequenza: 4, 5, 2, 3, 1 (poiché, per esempio, se cancellasse prima il 2 del 5 avremmo un memory leak);

18. [2 punto] - Implementare una funzione ReverseAfter che inverte gli elementi di una lista linkata dalla prima occorrenza del valore in input fino alla fine.

ESEMPIO: avendo in input una lista: A, B, C, D, E, F e come parametro in input D, la lista ritornata dalla funzione sarà: A, B, C, F, E, D
Implementare la funzione senza l'ausilio di nodi di supporto.

```
struct Node
{
   Node* next;
   int value;
};

void ReverseAfter(Node* head, int value);
```

19. [1 punti] - Nella seguente funzione identificare il maggior numero di errori e assunzioni errate. Inoltre elencare le varie ottimizzazioni possibili.
Da notare che ci sono almeno:
- 1 maggiore assunzione di algoritmo errato;
- 2 errori di portabilità;
- 1 errore di sintassi.

```
void myMemcpy(char* dst, const char* src, int nBytes)
{
   // Try to be fast and copy a word at a time instead of byte by byte
   int* wordDst = (int*)dst;
   int* wordSrc = (int*)src;
```

```
    int numWords = nBytes >> 2;
    for (int i=0; i < numWords; i++)
    {
        *wordDst++ = *wordSrc++;
    }
    int numRemaining = nBytes - (numWords << 2);
    dst = (char*)wordDst;
    src = (char*)wordSrc;
    for (int i=0 ; i <= numRemaining; i++);
    {
        *dst++ = *src++;
    }
}
```

20. [Facoltativo per la Lode] - Senza l'uso di altre chiamate di funzione, riscrivere il seguente codice accertandosi che il puntatore p sia sempre allineato a 16 byte per tutta la durata di un'applicazione:

```
char *p = malloc(1024);
…
free(p);
```