Exercise 1 [15%]

Use induction to prove the following: $$\sum_{i=1}^{n} i \cdot i! = (n+1)! - 1$$

$$\sum_{i=1}^{n} i.i! = (n+1)! - 1$$

for $n = 1$

$1(1)! = 1(1) = 1$ eq $(1+1)! - 1 = 2! - 1 = 2 - 1 = 1$

Assume it is true for $n = K$

(a) $\rightarrow 1(1)! + 2(2)! + 3(3)! + \dots + K(K)! = (K+1)! - 1$

we need to show

$\rightarrow 1(1)! + 2(2)! + 3(3)! + \dots + (K+1)(K+1)! \overset{?}{=} (K+2)! - 1$

we add $(K+1)(K+1)$ to both sides of (a)

(a) $\rightarrow 1(1!) + 2(2!) + 3(3!) + \dots + K(K!) + (K+1)(K+1)!$

$$= (K+1)! - 1 + (K+1)(K+1)!$$

$$= (K+1)! + (K+1)(K+1)! - 1$$

$$= (K+1)! [1 + (K+1)] - 1$$

$$= (K+1)! (K+2) - 1$$

$$= \underline{\underline{(K+2)! - 1}}$$

Induction complete.

Exercise 2 [15%]

Consider the two classes given below: `Course` and `Professor`. Find three different mistakes in the code, explain what the problems are and propose a fix.

```
class Course {
  public:
    Course(string profName, int courseNumber,
            int prerequisite ) {
      prof.name = profName;
      cNumber = courseNumber;
      prereq = prerequisite;
    }
  private:
    Professor prof;
    int cNumber;
    int prereq;
}
```

```
class Professor {
  public:
    Professor( string n ) {
      name = n;
    }
  Private:
    string name;
}


boolean SamePrereq(Course C1,Course C2) {
  return (C1. prereq == C2. prereq); }


int main () {
  Professor P(“Seraja”);
  Course C1(P->name, 456, 123);
  Course C2(P->name, 457, 123);
  boolean result = SamePrereq(C1,C2);
}
```

Mistake 1 `prof.name = profName;`	Explanation Since the `name` member is the Professor class’ private data, it is prohibited to assign `prof.name` with `profName` directly
	Fix Use a getter function.

Mistake 2
`Course C1(P->name, 456, 123);`
or
`Course C2(P->name, 457, 123);`
	Explanation
	`Course C2(P->name, 457, 123);`

	Fix
	`Course C1(P.name, 456, 123);`
	`Course C2(P.name, 457, 123);`

Mistake 3
`boolean SamePrereq(Course C1,Course C2) {`
`  return (C1. prereq == C2. prereq); }`
	Explanation
	SamePrereq is unauthorized to access private members of class Course

	Fix Declare SamePrereq as a friend function to class Course Or use a getter or make it a member function of Course

Exercise 3 [25%]

Given the following function:

```
#define MAX_VAL 150
void magic(int n)
{
   if (n<=0) return;
   if (n> MAX_VAL) return;
   cout << n;
   magic(2*n);
   cout << n;
   return;
}
```

a. What is printed to the console when `magic(5)` is called? [5%]

5 10 20 40 80 80 40 20 10 5

b. Is this function safe to use with any value of n? Explain why? [5%]

Yes, no more recursive calls when n goes beyond MAX_VAL.

c. Rewrite the above function using tail recursion. [15%]

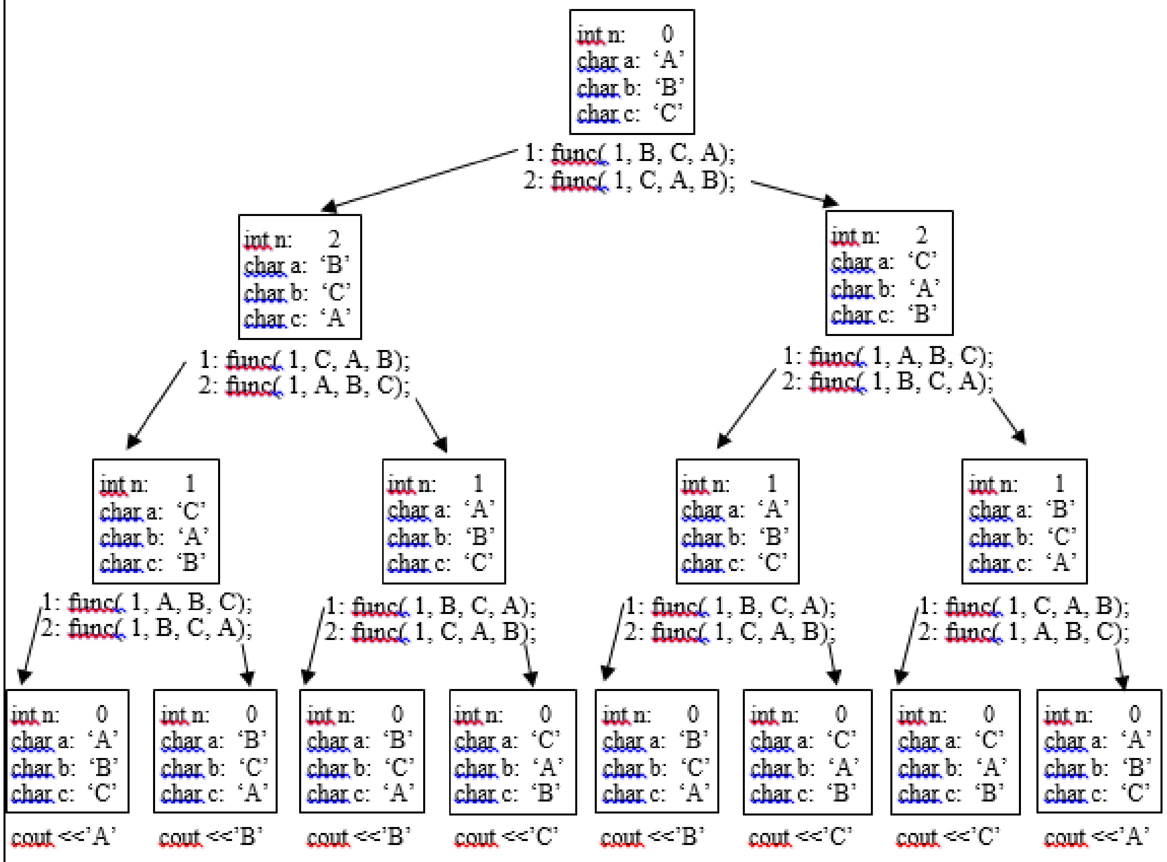
```
void call_magic(int n)
{
      magic(n,1);
}

void magic(int n, int dir)
{
   if (n<=0) return;
   if (n> MAX_VAL) { dir = -1; n = n/2;}
   printf(“%d ”,n);
   magic(n+(dir*n), dir);
}
```

Exercise 4 [10%]

Consider the following program that includes a recursive function, `func( )`. Trace a run of this program using the box method. What output will this program display?

```
void func( int n, char a, char b, char c ) {
   if( n > 0 ) {
     func( n - 1, b, c, a );
     func( n - 1, c, a, b );
   } else
     cout << a;
}
void main( ) {
   func( 3, ‘A’, ‘B’, ‘C’ );
   cout << endl;
 }
```

Trace a run using the box method: (10pts)



int n: 0
char a: ‘A’
char b: ‘B’
char c: ‘C’

1: func(1, B, C, A);
2: func(1, C, A, B);

int n: 2
char a: ‘B’
char b: ‘C’
char c: ‘A’

1: func(1, C, A, B);
2: func(1, A, B, C);

int n: 2
char a: ‘C’
char b: ‘A’
char c: ‘B’

1: func(1, A, B, C);
2: func(1, B, C, A);

int n: 1
char a: ‘C’
char b: ‘A’
char c: ‘B’

1: func(1, A, B, C);
2: func(1, B, C, A);

int n: 1
char a: ‘A’
char b: ‘B’
char c: ‘C’

1: func(1, B, C, A);
2: func(1, C, A, B);

int n: 1
char a: ‘A’
char b: ‘B’
char c: ‘C’

1: func(1, B, C, A);
2: func(1, C, A, B);

int n: 1
char a: ‘B’
char b: ‘C’
char c: ‘A’

1: func(1, C, A, B);
2: func(1, A, B, C);

int n: 0
char a: ‘A’
char b: ‘B’
char c: ‘C’
cout <<’A’

int n: 0
char a: ‘B’
char b: ‘C’
char c: ‘A’
cout <<’B’

int n: 0
char a: ‘B’
char b: ‘C’
char c: ‘A’
cout <<’B’

int n: 0
char a: ‘C’
char b: ‘A’
char c: ‘B’
cout <<’C’

int n: 0
char a: ‘B’
char b: ‘C’
char c: ‘A’
cout <<’B’

int n: 0
char a: ‘C’
char b: ‘A’
char c: ‘B’
cout <<’C’

int n: 0
char a: ‘C’
char b: ‘A’
char c: ‘B’
cout <<’C’

int n: 0
char a: ‘A’
char b: ‘B’
char c: ‘C’
cout <<’A’

Output displayed by this program: (1pts)

ABBCBCCA

Exercise 5 [20%]

1. A binary search uses a ______ strategy.
   - **a) divide-and-conquer**
   - b) sequential
   - c) determine-the-pivot
   - d) smallest-to-largest
2. How many bases cases does a recursive binary search of a sorted array have?
   - a) 0
   - b) 1
   - **c) 2**
   - d) 3
3. A recursive solution that finds the factorial of *n* always reduces the problem size by _____ at each recursive call.
   - **a) 1**
   - b) 2
   - c) half
   - d) one-third
4. When each module performs one well-defined task, we say that it is ___.
   - a) loosely coupled
   - b) highly coupled
   - **c) cohesive**
   - d) not easily reused
5. A(n) ______ is a C++ construct that enables a programmer to define a new data type.
   - **a) class**
   - b) method
   - c) data field
   - d) object
6. For the method `remove(anEntry)` of the ADT Bag, what would be the output of the method?
   - a) `anEntry`
   - b) nothing
   - **c) `true` or `false`**
   - d) the previous position of `anEntry` in the bag
7. A(n) ______ is a class that inherits the members of another class.
   - a) base class
   - b) superclass
   - c) abstract class
   - **d) subclass**
8. Data structures are part of an ADT's ______.
   - a) definition
   - **b) implementation**
   - c) specifications
   - d) usage
9. Encapsulation combines an ADT's data with its operations to form a(n) ______.
   - a) exception
   - b) method
   - **c) object**
   - d) variable
10. A function can indicate that an error has occurred by ______ an exception.
    - **a) throwing**
    - b) catching
    - c) implementing
    - d) declaring

Exercise 6 [15%]

Consider the following definition of class Shape:

```
Class Shape {
  protected:
        int x, y;
  public:
  void move_to(int xx, int yy) { x= xx; y= yy; }
  virtual void draw(void)=0;
};
```

a) `move_to` is a(n) ____member__________________________ function.

b) `draw` is a(n) ________pure virtual______________________ function.

c) class `Shape` is a(n) ____abstract base__________________________ class.

d) Implement class `Circle` which is derived from the `Shape` class.

e) Implement `operator==` for the `Circle` class.