Question 1 [20 points]

Write a C++ function called **deleteEmployee**.It receives: an array, A, of pointers to dynamically created objects of the type Employee, the count *n*, representing the number of Employee pointers currently stored in A, and an integer key, representing the ID of one of the Employees in the array A (all Employee IDs are unique). The function finds and deletes the Employee whose ID matches key. After deletion the function must also copy back (towards left) all the pointers in A to the right of the deleted employee. The function should return the new value of n after a successful deletion. If no employee is found with ID equal to the key, the function should return -1. Following is the definition of the struct Employee:

```
struct Employee{
     char * name; //name of the employee created at runtime
     int ID; //Id number to be compared with the key
     float salary;
};
```

Please note the following:

- You do not need to take any inputs, or output anything to the screen.
- You do not need to create the array A. It is passed to your function.
- Make sure that there are no memory leaks.
- Pay attention to the arguments to the function and their types.
- Don't write the main program. All we need is the code for deleteEmployee.

Solution:

```
int deleteEmployee(Employee ** &A, int n, int key){

    for(int i=0;i<n;i++){
        if(A[i].ID==key){
            delete []A[i].name;
            delete A[i];
            Employee ** temp = new Employee * [n-1];
            for(int j=0,k=0;j<n-1;k++){
                if(k!=i)
                temp[j++] = A[k];
            }
            delete []A;
```

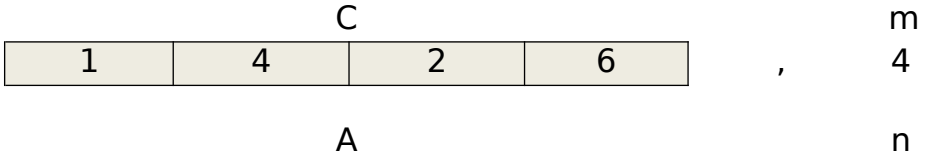
```
                A=temp;
                return n-1;
            }
        }

        return -1;
    }
```

Question 2 [20 points]

In the following, an ID is simply an integer between 0 and *n*-1.

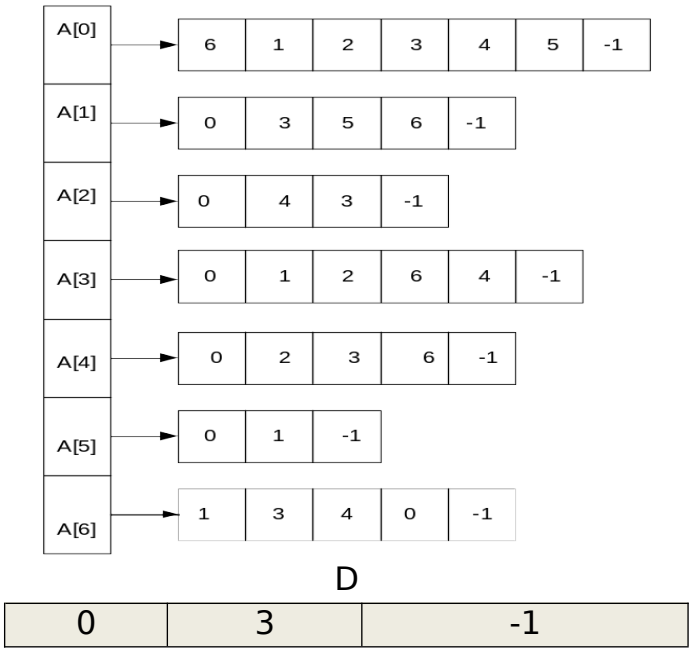
A social network stores the friends' lists of all its users in a two dimensional dynamic array called A, with row A[*i*] containing a pointer to an array with IDs of all the users that are friends of the user *i*, and the end of the array is marked with -1. There are *n* users on the network, hence the pointers A[0] to A[*n*-1] point to the arrays of their friends, respectively. Your task is to write a C++ function called **computeCommonFriends**, that receives array A, and the number of users *n*, and another array, C, containing IDs of *m* different user, where *m* is also an input to the function. The function should then compute and return another dynamic array, D, with IDs of all the common friends of the *m* users mentioned in array C. The size of D should be exactly the same as the number of common friends, plus one extra space containing -1, to mark the end.

An example input and its corresponding output are shown below. In the example users 0 and 3 are the common friends of users 1, 4, 2 and 6, passed in array C.



Name: _____ Roll No: _____
Section:_____

7



Solution:

```
int * Intersection(int * a, int * b){
	int i=0, j=0, k, common=0;
	while(a[i]!=-1){
		j=0;
		while(b[j]!=-1){
			if(a[i]==b[j]){
				common++;
				break;
			}
			j++;
		}
		i++;
	}

	int * res = new int[common+1];
	res[common]=-1;

	k=0;
	while(a[i]!=-1){
		j=0;
		while(b[j]!=-1){
```

```
                if(a[i]==b[j]){
                        res[k++] = a[i];
                        break;
                }
                j++;
        }
        i++;
    }

    return res;
}

int * computCommonFrinds(int **A, int n, int * C, int m){
    int * D ,temp;
    D = Intersection(A[C[0]],A[C[1]]);

    for(int i=2;i<m;i++){
        temp = Intersection(D,A[C[i]]);
        delete []D;
        D = temp;
    }

    return D;
}
```

Question 3 [10 + 10points]

Write a function template called **mismatch**(…) that compares two given arrays of any type, and returns the index where the first mismatch occurs. If the two arrays are identical the function returns -1. For example, for the following arrays a and b, the mismatch occurs at index 3.

```
int a[] = {10, 20, 30, 40, 50};  int b[] = {10, 20, 30, 90, 80}
```

(a) Provide all code necessary to make sure that the following program executes successfully (without any syntactic or logical errors).

```
int main() {
      int a[] = {10, 20, 30, 40, 50};
      int b[] = {10, 20, 30, 90, 80, 70};
      cout <<mismatch( a, 5, b, 6 ) << endl;
      float f1[] = {1.1, 3.2, 5.3};
      float f2[] = {1.1, 3.2, 5.3};
      cout << mismatch( f1, 3, f2, 3 ) << endl;
      char* c[] = {“ab”, “bcd”}; char* d[] = {“ab”, “yz”};
      cout <<mismatch( c, 2, d, 2 ) << endl;
      return 0;
}
```

Solution:

```
template<class T>
int mismatch(T A[], int n1, T B [], int n2){
        for(int i=0,;i<n1 && i<n2;i++){
                if(A[i]!=B[i])
                        return i;
        }

        if(i<n1)
                return n2;
        if(i<n2)
                return n1;

        return -1;
}
template<>
int mismatch(char * A[], int n1, char* B[], int n2){
        for(int i=0,;i<n1 && i<n2;i++){
                if(strcmp(A[i],B[i])!=0)
                        return i;
        }
        if(i<n1)
                return n2;
        if(i<n2)
                return n1;
        return -1;
}
```

(b) Consider the following piece of code.

```
int main(){
      Fraction fr1[2], fr2[3];
      ...
      cout << mismatch(fr1, 2, fr2, 3 ) << endl;  return 0;
}
```

The class Fraction is defined partially as follows,

```
class Fraction{
private:
      int num; //numerator
      int den; //denominator
public:
      Fraction();
      ~Fraction();
};

bool Fraction::opertor!=(Fraction f){
      int a,b;
```

```
            a= gcd(num,den);
            b=gcd(f.num,f.den);
            if(num/a==f.num/b && den/a == f.den/b)
                  return false;
            else
                  return true;
      }
```

Add all code necessary to the class Fraction so that your function template from part (a) works correctly in the code given above.

Question 4[**20 points**]

You have been hired by a furniture store owner to develop an inventory program. He sells two main types of furniture: Beds and Chairs. Each bed or chair can be either small, medium or large. Each chair can be of different height. This height is measured in feet. There is also a special type of chair available called a Personalized Chair. Personalized chairs can have a name printed on the back.
The store owner also knows C++ and has taken upon himself to construct the user interface. However, he has given you a basic driver program shown below which uses all the classes and functions that are required. He also has given you a sample output of the code.
Write C++ classes in a proper hierarchy which enable the driver given below to compile and produce the given output. You cannot change the driver program at all. Your code also must not have any memory leaks.

```
#define SMALL 0
#define MEDIUM 1
#define LARGE 2
int main()
{
      const int size = 5;
      Furniture ** inventory = new Furniture * [size];
      inventory[0] = new Bed(SMALL);
      inventory[1] = new Chair(MEDIUM);
      inventory[2] = new Chair(MEDIUM,3);
      inventory[3] = new Bed(LARGE);
      inventory[4] = new PersonalizedChair(SMALL,2,"Ali");



for (int i = 0; i < size; i++)
      {
            inventory[i]->printDescription();
            delete inventory[i];
      }
      delete [] inventory;

      return 0;
```

```
}
```

Output:

```
A small bed
A medium 2 feet high chair
A medium 3 feet high chair
A large bed
A small 2 feet high chair for Ali
```

Solution

```
class Furniture{
	int size;
public:
	Furniture(int s=0){
		size=s;
	}
	virtual void printDescription(){
		if(size==0)
			cout<<"A SMALL ";
		if(size==1)
			cout<<"A Medium ";
		if(size==2)
			cout<<"A Large ";
	}
	virtual ~Furniture();
}
};

class Chair:public Furniture{
	int height;
public:
	Chair(int s, int h=2):Furniture(s){
		height = h;
	}
	virtual void printDescription(){
		Furniture::printDescription();
		cout<<height<<" feet high chair"<<endl;
	}
	virtual ~chair();
};

class Bed:public Furniture{
public:
	Bed(int s):Furniture(s){}
	virtua void print Description(){
		furniture::printDescription();
```

```
                cout<<" Bed"<<endl;
        }
};

class PersonalizedChair:public Chair
{
        char * name;
public:
        PersonalizedChair(int s, int h, char * n):Chair(s,h){
                int l = strlen(n);
                name = new char[l+1];
                strcpy(n,name);
        }
        virual void printDescription(){
                Chair::printDescription();
                cout<<" for "<<name<<endl;
        }
        virtual ~PersonalizedChair(){
                delete []name;
        }
};
```

THE END