Computer Programming
Final Examination, Spring 2013

National University of Computer and Emerging Sciences

Marks: 60 **Time:** 3 hrs.

Q1. Implement a C++ class called `Array` which satisfies all the requirements of the following **main** program. Following is a partial definition of this class. [**20**]

```
class Array{
      int * a;//dynamic array
      int cap;//capacity (total space in the array)
      int n;  //size (number of elements inserted)
   public:
      // Implementation here
};

int main()
{
      int temp[]={12,4,6,11,5};
      Array A;     // make an empty array of capacity 0

      Array B(temp,5,10);
      /*
      Make an array with size 5 and capacity 10. If no capacity is specified or capacity
      is less than size, then set capacity equal to size, and allocate the array of
      that capacity.  */

      int n, input;
      cin>>n;
      for(int i=0;i<n;i++){
            cin>>input;
            A.insert(i);
            /*
               Insert puts the new integer at the next available index
               in the array. If size exceeds capacity then, a new array
               of double capacity is allocated, data is copied into it
               and the older array is de-allocated.     */
      }
      A.reverseInRange(3,7);
      /*
            The order of all elements in the given range is reversed, keeping the remaining
      Elements
            un-disturbed. For example, if the array contained 2,6,8,4,1,13,7,5, then after
            this call it will become: 2,5,8,7,1,13,4,6 (notice the elements in bold have been
            reversed)    */

      return 0;
}
```

```
	// default constructor
	Array():cap(0),a(nullptr),n(0) {}

	// overloaded constructor
	Array(const int arr[], int size, int capacity=-1):n(size),cap(capacity)
	{
		// handle edge cases
		if (cap==-1 || cap < n)
		{
			cap = n;
		}

		// copy array
		a = new int[cap];
		for (int i=0; i<n; i++)
		{
			a[i] = arr[i];
		}
	}

	// dtor
	~Array()
	{
		if (a!=nullptr)
		{
			delete []a;
		}
	}

	void insert(int input)
	{
		// when capacity is zero, create an array of size 1
		if (cap==0)
		{
			a = new int[1];
			n = cap=1;
			a[0] = input;
			return;
		}

		// resize when necessary
		if (n+1>cap)
		{
			int * temp = a;
			cap *= 2;
			a = new int[cap];
			for (int i=0; i < n; i++)
			{
				a[i] = temp[i];
			}
			delete temp;
		}
		a[n] = input;
		n++;
	}




	void reverseInRange(int start, int end)
```

```
	{
		// allocate space for worst case when all elements are to be reversed
		int * temp = new int[cap];
		int count = 0;

		// store the elements to be reversed in a temp array
		for (int i=0; i < n; i++)
		{
			if (a[i]>=start && a[i]<=end)
			{
				temp[count++] = a[i];
			}
		}

		// overwrite the elements from temp in reverse order
		for (int i=0; i < n; i++)
		{
			if (a[i]>=start && a[i]<=end)
			{
				a[i] = temp[--count];
			}
		}

		delete [] temp;

	}
```

Q2. Write the template function(s) **find(array_A,size_of_A,element_to_search)** such that it finds the **element_to_search** in the array **array**_Aand returns the index of its first occurrence. If the element is not found, it returns -1. Make sure that the following code executes successfully.**[10]**

```
int main() {
	int a[]={2,3,5,8};
	char b[]={'H', 'e', 'l', 'l', 'o'};
	char* c[] = {"cd", "lion", "zoo"};

	cout<< "Index = " <<find(a,4,3);
	cout<< "Index = " <<find(b,5,'l');
	cout<< "Index = " <<find(c,3,"lion");

	return 0;
}
```

```
template <typename T>
int find(T array_A[],int size_of_A,T element_to_search) {
	for (int i=0; i<size_of_A; i++) {
		if (array_A[i] == element_to_search) {
			return i;
		}
	}
	return -1;
}
```

```
int find(char* array_A[],int size_of_A,char* element_to_search) {
      for (int i=0; i<size_of_A; i++) {
            if (strcmp(array_A[i],element_to_search)==0) {
                  return i;
            }
      }
      return -1;
}
```

Q3. Suppose we have a 2D matrix of characters with dimensions **row** x **col**. You have to write a function to copy the given matrix to another matrix (with same dimensions **row** x **col**) with all **rows** and **columns** containing the occurrence of a given character removed. The rows and columns at the end should be filled with '-'. In the following example, **Matrix2** is formed after removal of character 'a' from **Matrix1**. **[15]**

b	c	d	f	-	-
s	t	u	w	-	-
-	-	-	-	-	-
-	-	-	-	-	-

Example 1:

b	c	d	e	f	h
t	u	x	a	g	h
y	i	l	m	n	a
s	t	u	v	w	x

Input Matrix **Output Matrix**

Example 2:

b	c	d	e	f	h
t	u	x	a	g	a
y	i	l	m	n	h
s	t	u	v	w	a
b	c	d	f	-	-
y	i	l	n	-	-
-	-	-	-	-	-
-	-	-	-	-	-

Input Matrix **Output Matrix**

Assuming ***col*** is a pre-defined integer constant, implement it using following function prototype:

void RemoveFromMatrix(char Matrix1[][col],const char rchar, char Matrix2[][col],const int rows) ;

```
void RemoveFromMatrix(char Matrix1[][col],const char rchar, char Matrix2[][col],const int
rows)  {
	int i=0, j=0;
	int *rowNo = new int [rows];
	int colNo[col]={0};

	//to find the indexes of rows and col where 'a' is found
	for(int i=0;i<rows;i++)
		for(int j=0;j<col;j++)
			if(Matrix1[i][j]==rchar){
				rowNo[i]=1;
				colNo[j]=1;
			}
			int iRow=0;
			int iCol=0;

			for(int i=0;i<rows;i++){
				if (rowNo[i]!=1){
					iCol=0;
					for(int j=0;j<col;j++){
						if(colNo[j]!=1){
							Matrix2[iRow][iCol]=Matrix1[i][j];
							iCol++;
						}
					}
					iRow++;
				}
			}

			delete [] rowNo;
}
```

Q4. Given the main function below, implement a class called `Tool`. It should store the strength of the tool as a number. Implement three more classes called `Rock`, `Paper`, and `Scissors`, which inherit from `Tool`. These classes will need a public function bool `fight(Tool&)` that compares their strengths in the following way:

- Rock's strength remains unchanged whenever it is the first opponent.
- Scissors strength is increased two and half times (temporarily) whenever it is the first opponent.
- Paper's strength is doubled (temporarily) whenever it is the first opponent.

The one with the highest strength wins the fight.

Implement the necessary constructors, getters and setters. You may also include any extra auxiliary functions and/or fields in any of these classes. Following is the main program that should work with your code.**[15]**

```
int main(){
	Tool *tools[3];
	Scissors s1(5);
	Paper p1(7);
	Rock r1(15);
```

```
	tools[0]=&r1;
	tools[1]=&p1;	tools[2]=&s1;

	int choice1=0, choice2=0;

	cout<<"1: Rock (r)\n";
	cout<<"2: Paper (p)\n";
	cout<<"3: Scissors (s)\n";
	cout<<"Enter the choices for the two contestants to fight and -1 to end:\n";
	cin>>choice1;

	// 3 2
	// should play Scissors against paper

	while(choice1!=-1){
		cin >> choice2;
		cout << tools[choice1-1]->getWhoAmI()<<" vs "<< tools[choice2-1]->getWhoAmI() <<
endl;
		// output: Scissors vs Paper
		if (tools[choice1-1]->fight(*tools[choice2-1])) {
			cout << tools[choice1-1]->getWhoAmI()<<" won! "<< endl;
		} else {
			cout << tools[choice1-1]->getWhoAmI()<<" lost! "<< endl;
		}
		// Scissors won!
		cout<<"Enter the choices for the two contestants to fight and -1 to end:\n";
		cin >> choice1;
	}
	return 0;
   }
```

```
class Tool{
private:
	int strength;
public:
	Tool(int s =0):strength(s){}
	virtual int fight(Tool &)=0;
	virtual char* getWhoAmI()=0;
	int getStrength(){ return strength; }
};
// Note that in this universe, paper can win against scissors if paper is the first contestant
class Scissors:public Tool{
public:
	Scissors(int s):Tool(s){};
	int fight(Tool &obj){ return getStrength()*2.5>=obj.getStrength(); }
	char* getWhoAmI(){ return "Scissors"; }

};

class Paper:public Tool{
public:
	Paper(int s):Tool(s){};
	int fight(Tool &obj){ return getStrength()*2>=obj.getStrength(); }
	char* getWhoAmI(){ return "Paper"; }

};

class Rock:public Tool{
public:
	Rock(int s):Tool(s){};
```

```
      int fight(Tool &obj){ return getStrength()>=obj.getStrength(); }
      char* getWhoAmI(){ return "Rock"; }

};
```