National University of Computer and Emerging Sciences, Lahore Campus

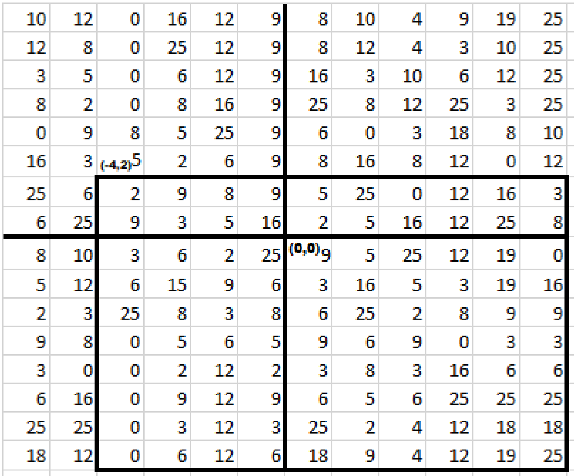
Course Name:	Object Oriented Programming	**Course Code:**	CS1004
Degree Program:	BS (CS, SE, DS)	**Semester:**	Spring 2023
Exam Duration:	60 Minutes	**Total Marks:**	20
Paper Date:	27-Feb-2023	**Weight**	15
Section:	ALL	**Page(s):**	4
Exam Type:	Midterm-I		

Student : Name:_______________________________ **Roll No.**________________ **Section:**_______

Instruction/Notes: Attempt all questions. Answer in the space provided. **Answers written on rough sheet will not be marked**. Do not use pencil or red ink to answer the questions**.** In case of confusion or ambiguity make a reasonable assumption.

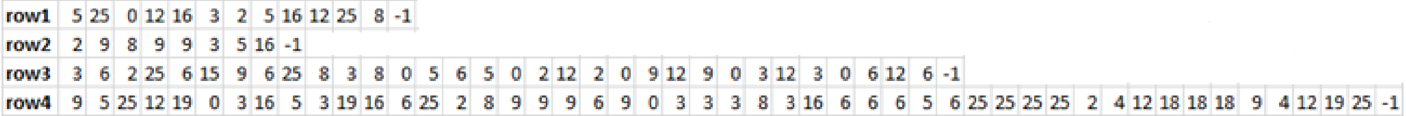
Question 1: [CLO 1] (Marks: 10)

A robot is designed to examine a crop field to detect and count the number of insects. The crop field is divided into four parts where each part has a different crop. The robot's camera (receptive field) can detect pests in a fixed size 10x10 square feet region at a time. Additionally, the software in the robot can detect and count number of pests in every 1x1 square foot region. You can imagine that the crop field is just an XY plane where origon (0,0) is at the mid of the field. Crop 1 is in 1st quadrant, crop 2 is in 2nd quadrant and so on. In order to minimize the use of pesticide, the experts need to determine the number of pests in each 1x1 region. Your task is to get the data from the robot and transform it such that the experts can decide the amount of pesticide for each crop in every 1x1 region. Define a C++ function that gets the top left corner (w.r.t origon) of the robot's receptive field and a two dimensional array p_count of 10x10 size as parameters. The array p_count contains the number of pests in 10x10 region of the crop field. The job of this function is to dynamically create and return a two dimensional array having four rows where each row of this array contains the number of pests in 1x1 regions for a particular crop. You need to store a sentinel value '-1' at the end of each row after storing the information from the given grid, indicating the end of list. Consider the following example.



10	12	0	16	12	9	8	10	4	9	19	25
12	8	0	25	12	9	8	12	4	3	10	25
3	5	0	6	12	9	16	3	10	6	12	25
8	2	0	8	16	9	25	8	12	25	3	25
0	9	8	5	25	9	6	0	3	18	8	10
16	3	(-4,2)5	2	6	9	8	16	8	12	0	12
25	6	2	9	8	9	5	25	0	12	16	3
6	25	9	3	5	16	2	5	16	12	25	8
8	10	3	6	2	25	(0,0)9	5	25	12	19	0
5	12	6	15	9	6	3	16	5	3	19	16
2	3	25	8	3	8	6	25	2	8	9	9
9	8	0	5	6	5	9	6	9	0	3	3
3	0	0	2	12	2	3	8	3	16	6	6
6	16	0	9	12	9	6	5	6	25	25	25
25	25	0	3	12	3	25	2	4	12	18	18
18	12	0	6	12	6	18	9	4	12	19	25

Here the highlighted 10x10 region has the top left corner (-4,2). There are 12 1x1 regions of crop1, 8 regions of crop 2, 32 regions of crop 3 and 48 such regions of crop 4. The resultant two dimensional array must have sizes 13, 9, 33, 49 for row1, row2, row3 and row 4 respectively. The resultant array must be as follows.



row1	5	25	0	12	16	3	2	5	16	12	25	8	-1																																				
row2	2	9	8	9	9	3	5	16	-1																																								
row3	3	6	2	25	6	15	9	6	25	8	3	8	0	5	6	5	0	2	12	2	0	9	12	9	0	3	12	3	0	6	12	6	-1																
row4	9	5	25	12	19	0	3	16	5	3	19	16	6	25	2	8	9	9	9	6	9	0	3	3	3	8	3	16	6	6	6	5	6	25	25	25	25	2	4	12	18	18	18	9	4	12	19	25	-1

Note: You are supposed to provide a generic code that should work accurately for any given top left corner.

```
int ** transform_data(int p_countn[][10], int topLeft_x, int topLeft_y) {
	int n[4], col_L, row_U, x, y;
	//row_U counts rows above the origion
	if (topLeft_y > 0 && topLeft_y - 10 >= 0)
		row_U = 10;
	if (topLeft_y > 0 && topLeft_y - 10 <= 0)
		row_U = topLeft_y;
	if (topLeft_y <= 0)
		row_U = 0;

	//col_L counts colums on the left of origion
	if (topLeft_x < 0 && topLeft_x+10 <=0)
		col_L = 10;
	if (topLeft_x < 0 && topLeft_x + 10 >= 0)
		col_L = -1 * topLeft_x;
	if (topLeft_x>=0)
		col_L = 0;

	//counts number of elements in each quadrant
	n[0] = (10 - col_L) * row_U;
	n[1] = col_L * row_U;
	n[2] = col_L * (10 - row_U);
	n[3] = (10 - col_L) * (10 - row_U);

	int** data = new int* [4];
	for (int i = 0;i < 4;i++)
		data[i] = new int[n[i]+1];//allocate dynamic memory for each quadrant
	//copy data of each quadrant in different rowws of data
	int k = 0;
	for (x = 0;x <row_U;x++)
		for (y = col_L;y < 10;y++)
			data[0][k++] = p_countn[x][y];
	data[0][k] = -1;
	k = 0;
	for(x= 0; x< row_U;x++)
		for(y=0;y<col_L;y++)
			data[1][k++] = p_countn[x][y];
	data[1][k] = -1;
	k = 0;
	for (x = row_U; x < 10;x++)
		for (y = 0;y < col_L;y++)
			data[2][k++] = p_countn[x][y];
	data[2][k] = -1;
	k = 0;
	for (x = row_U; x < 10;x++)
		for (y = col_L;y < 10;y++)
			data[3][k++] = p_countn[x][y];
	data[3][k] = -1;

	return data;

}
```

Question 2: [CLO 2] **(Marks: 2*5)**

For each of the following part, identify the logical error(s) if any and write the output produced by the code segment. In case the output cannot be determined write G

CODE SEGMENT	OUTPUT

`int* sum(int a, int b) {`
`  int sum = a + b;`
`  return ∑`
`}`
`int main() {`
`  int* addr = sum(5, 6);`
`  cout << *addr;`
`}`
	Output: 11 or garbage
	Error: Accessing local variable in main that is destroyed. addr is a dangling pointer.

`int main() {`
`  int arr[5] = { 1, 4, 5, 6, 7 };`
`  int *ptr = &arr[2];`
`  cout << (*ptr)+1 << endl;`
`  cout << *(ptr-1)<<endl;`
`  cout << *(ptr + 2)<<endl;`
`}`
	Output:
	6
	4
	7
	No error

`int * fun1(int* ptrs) {`
`  int* ptr = new int[7];`
`  for (int i = 0; i < 7; i++)`
`  ptr[i] =  ptrs[i];`
`  return ptr;`
`}`
`void display(int *ptr) {`
`  if (ptr != nullptr) {`
`  for (int i = 0; i < 5; i++)`
`  cout << ptr[i] <<" ";`
`  }`
`}`
`int main() {`
`  int arr[5] = {1,2,3,4,5};`
`  int* ptrs = arr;`
`  int *pt =fun1(ptrs);`
`  display(ptrs);`
`  cout << *(pt+1);`
`  return 0;`
`}`
	Output: 1 2 3 4 5 2
	Error: in fun1 index out of bound + memory leak in main function.

`int main() {`
 	
`char** s = new char* [2];`
`char* name1 = new char[20];`
`char* name2 = new char[20];`
 	
`strcpy_s(name1,20, "Hello");`
`strcpy_s(name2, 20, "World");`
`s[0] = name1;`
`s[1] = name2;`
 	
`delete[] name1;`
`delete[] name2;`
`for (int i = 0;i < 2;i++)`
`  cout << s[i] << endl;`
`delete[] s;`
`s = nullptr;`
`  return 0;`
	Output: Garbage
	Error: Dangling pointers [0] and s[1]
	
	Correct 2
	Partial 1
	Incorrect 0

`}`	

`class ABC{`
`private:`
`  int a;`
`public:`
`  ABC(){`
`  cout<<"ABC() Called.\n";`
`  a = 10;`
`  }`
`  ABC(int x){`
`  cout<<"ABC( int ) Called.\n";`
`  a = x;`
`  }`
`  void Print(){`
`  cout<<"a = "<<a<<endl;`
`  }`
`  void SetValue(int y){`
`  a = y;`
`  }`
`  void SomeFunction(){`
`  a = a * 5;`
`  }`
`};`
`void main(){`
`  ABC obj1;`
`  ABC obj2(20);`
`  obj1.Print();`
`  obj2.Print();`
`  cout<<"- - -\n";`
`  obj1.SetValue(2);`
`  obj2.SomeFunction();`
`  obj1.Print();`
`  obj2.Print();`
`}`
	Output: `ABC() Called.`
	`ABC(int) Called.`
	`a= 10`
	`a=20`
	--- 	
	a= 2
	a= 100