Task # 0

Consider the following C++ code fragment and answer the following questions:

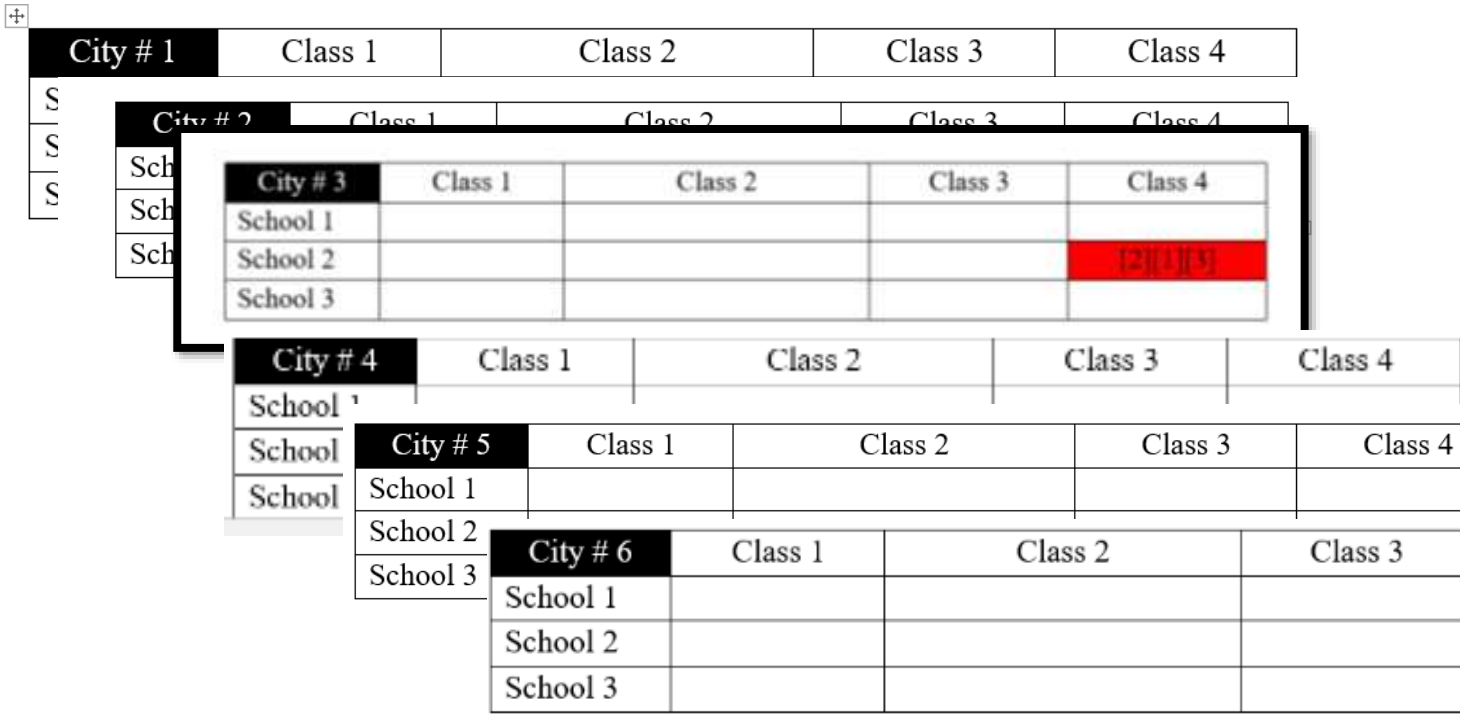
const int City = 6, School = 3, Class = 4;

int enrolled_Students[City][School][Class];

A. What is represented by enrolled_Students [2][1][3]? Explain it.

enrolled_Students is a 3d matrix with dimensions of 6x3x4, means 2 cities have 3 schools with 4 classes. While ***enrolled_Students [2][1][3]*** is for accessing the data this represents the data of 3rd city's 2nd schools' 4th class data. The numbers in subscript operator ([]) shows the indexes of arrays thus +1 is for telling in an actual data.

Furthermore, by diagram can be explained as (the desired data will be highlighted by red):



B. Compute the logical address of enrolled_Students [2][1][3]. Suppose the base address of the table is 0x100(h) in C++? Show complete steps. Also show the addressable location with the help of a diagram to support your calculations. Show final address in hexadecimal form. Marks will not be awarded for the direct answer.

Hint: dimensions of enrolled_Students are 6 x 3 x 4

According to the given dimensions we have 6 different layers of 12 elements thus;

Address required =

base address +4(# of city(school*classes) + # of school (total # classes) + classes);

multiplied by 4 because size of each will be 4 as integer array, first finding the second part then converting to the hexa-decimal value then adding to the base:

- ⇨ 4(2(12)) + 1(4) + 3)
- ⇨ 4(24+4+3)
- ⇨ 4(31)
- ⇨ 124

Converting to the hexa-value:

$(124)_{10} = (7C)_{16}$

Now adding the values.

Address of enrolled_Students [2][1][3] = 100 + 7C

$= (17C)_{16}$

$= 0x17C_{(h)}$

Below in the figure there are 6 different colors showing each layer of 2d array. In memory the addresses are save in row wise. Each 6 color can be considered as the layers and each layer then have three rows, 4 columns, 12 elements. **The red color is showing the required address**. Each box has 4 bytes and there is 4 bytes' increment when move to next because the array used in example is int, thus increments will result addition of four bytes.

0x100(h)	0x104(h)	0x108(h)	0x10C(h)	0x110(h)	0x114(h)	0x118(h)	0x11C(h)	0x120(h)	0x124(h)
0x128(h)	0x12C(h)	0x130(h)	0x134(h)	0x138(h)	0x3C(h)	0x140(h)	0x144(h)	0x148(h)	0x14C(h)
0x150(h)	0x154(h)	0x158(h)	0x15C(h)	0x160(h)	0x164(h)	0x168(h)	0x16C(h)	0x170(h)	0x174(h)
0x178(h)	0x17C(h)	0x180(h)	0x184(h)	0x188(h)	0x18C(h)	0x190(h)	0x194(h)	0x198(h)	0x19C(h)
0x1A0(h)	0x1A4(h)	0x1A8(h)	0x1AC(h)	0x1B0(h)	0x1B4(h)	0x1B8(h)	0x1BC(h)	0x1C0(h)	0x1C4(h)
0x1C8(h)	0x1CC(h)	0x1DO(h)	0x1D4(h)	0x1D8(h)	0x1DC(h)	0x1E0(h)	0x1E4(h)	0x1E8(h)	0x1EC(h)
0x1F0(h)	0x1F4(h)	0x1F8(h)	0x1FC(h)	0x200(h)	0x204(h)	0x208(h)	0x20C(h)	0x210(h)	0x214(h)
0x218(h)	0x21C(h)	-	-	-	-	-	-	-	-

Table 1 The values here are in hexa-value and each block shows 4 bytes as int array

The final required address is 0x17C(h)