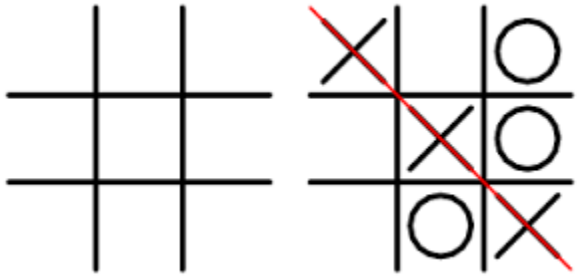
Multi-D

Tic Tac Toe Problem

Tic-tac-toe is a paper-and-pencil game for two players, *X* and *O*, who take turns marking the spaces in a 3×3 grid. Player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row wins the game.



Tic Tac Toe Problem Contd...

Given the board configuration of the tic tac toe game, determine if the board is in either of the following states: empty, player1 wins, player2 wins, draw or intermediate. The board is said to be in initial state if all the cells contain '-1', player1 uses '1' as his coin and player2 uses '2' as his coin. The game is draw when the board is full and no one has won the game. The game is in intermediate state when no one has won and board is not full

Tic Tac Toe problem

Input	Output	Logic Involved
Current board configuration	State of the board as win, draw, initial or intermediate	Find average and difference

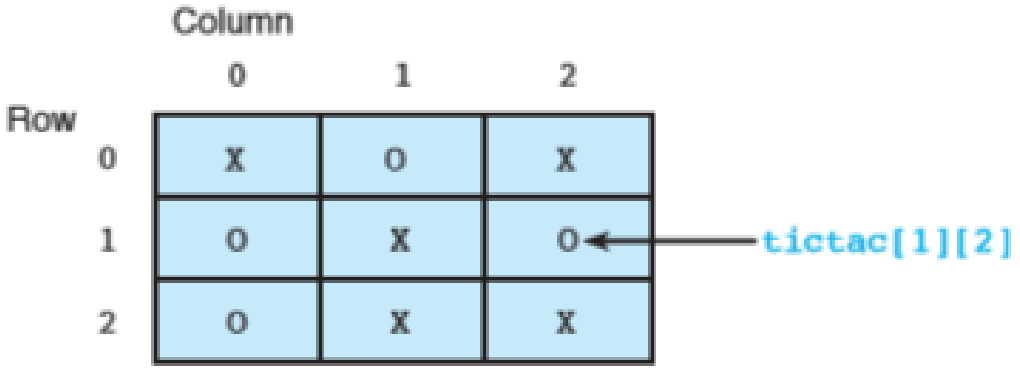
Algorithm

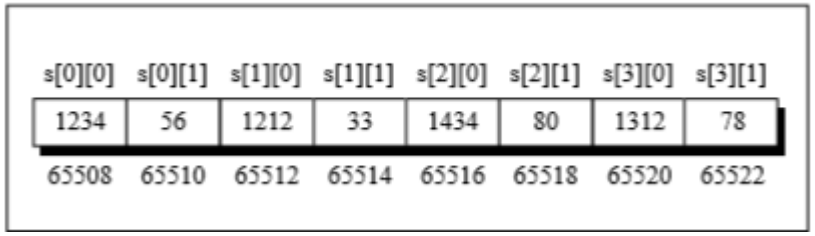
- Represent the board in memory
- Get the elements in first row, second row and so on
- Process the elements
- If all are -1 then print ‘empty’
- If ‘1’ is placed row wise, column wise or diagonally then print ‘Player 1’ wins
- If ‘2’ is placed row wise, column wise or diagonally then print ‘Player 2’ wins
- If all cells are full and no one has won the game then print ‘Draw’
- Otherwise print intermediate

New Stuffs...

- Represent the board in memory using a 2 d array
- Traverse the board using nested loop
- Memory is only a 1 d structure
- But high level languages supports arrays of multiple dimensions
- A kind of ordering is done internally to support multi dimensional arrays
- Either row major or column major ordering is done
- ‘C’ does row major ordering for 2 D arrays

Representation of Tic Tac Toe Board





Row Major Ordering of 2 D Arrays

- Elements in the first row are placed followed by elements of second row and so on
- Contiguous memory allocation is done and address of first byte of memory is stored in the name of the array
- Address of nth element in an array named as a (i.e.) a[n], is determined as: (a+n*b) where b is the number of bytes allocated for the data type

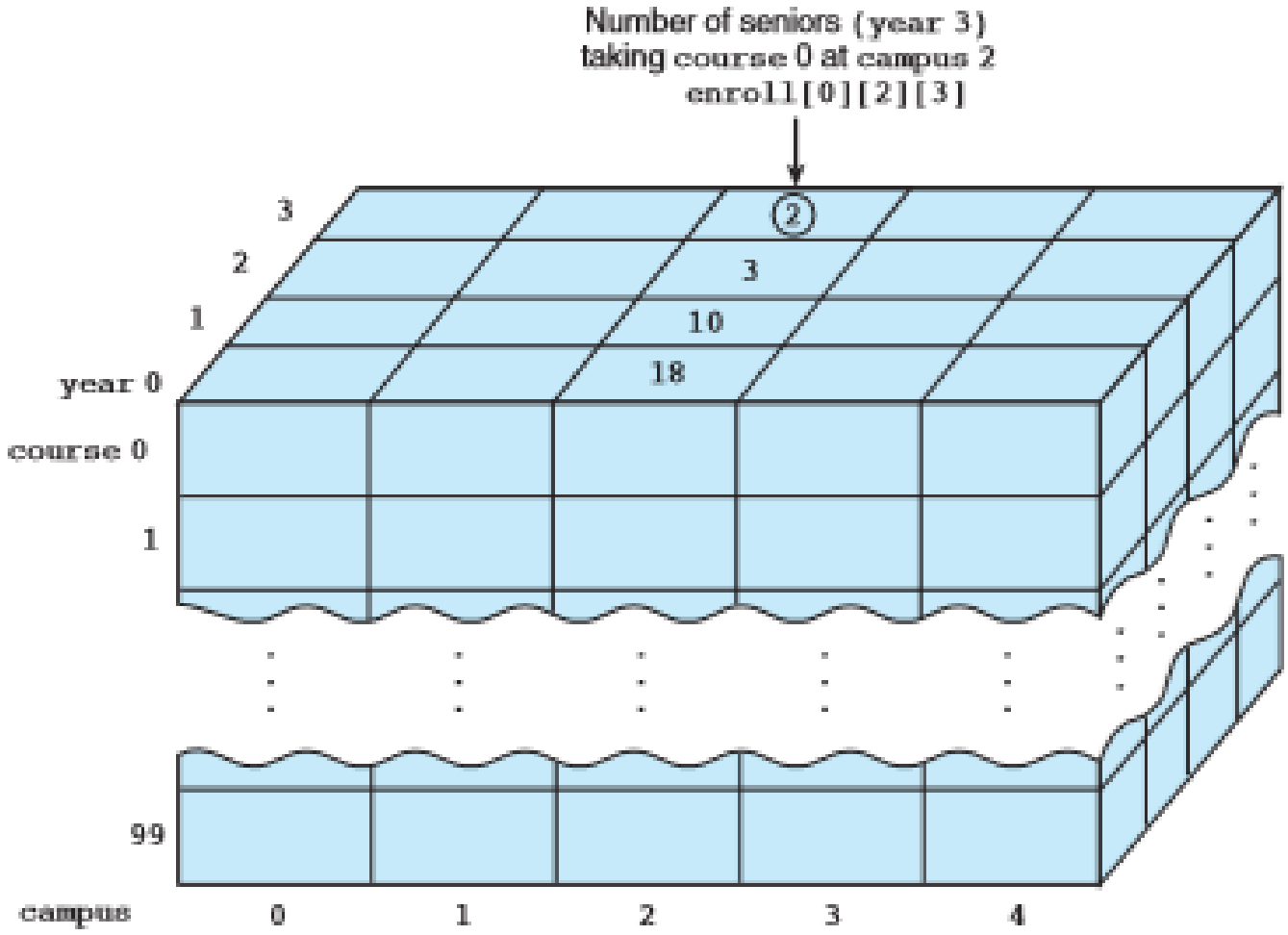
Initialization of a Character 2 D Arrays

```
char tictac[3][3] = { {' ', ' ', ' '}, {' ', ' ', ' '},
                      {' ', ' ', ' '} };
```

Multi dimensional Arrays

- store the enrolment data for a college
- assume that the college offers 100 (MAXCRS) courses at five different campuses
- Students from any of the four years can opt for the courses

Arrays with Several Dimensions


Number of seniors (year 3)
taking course 0 at campus 2
enroll[0][2][3]
3
2
1
year 0
2
3
10
18
course 0
1
99
campus
0
1
2
3
4

```c
#include<stdio.h>
void main()
{
        int tictactoe[3][3];
        int i_Counter = 0, j_Counter=0,empty_Cells=0,win = 0;
        //Read the state of the board
        for(i_Counter=0;i_Counter<3;i_Counter++)
        for(j_Counter=0;j_Counter<3;j_Counter++)
                scanf("%d",&tictactoe[i_Counter][j_Counter]);
        //Count the number of empty cells
        for(i_Counter=0;i_Counter<3;i_Counter++)
        for(j_Counter=0;j_Counter<3;j_Counter++)
                if(tictactoe[i_Counter][j_Counter]==-1)
                        empty_Cells++;
        if (empty_Cells==9)
                printf("Initial State");
```

```
else
{
        //Check rowwise
        for(i_Counter=0;i_Counter<3;i_Counter++)
        {
                if (tictactoe[i_Counter][0]==1&&tictactoe[i_Counter][1]==1&&tictactoe[i_Counter][2]==1)
                        {
                        printf("Player1 Wins");
                        win =1;
                        }
                else if (tictactoe[i_Counter][0]==2&&tictactoe[i_Counter][1]==2&&tictactoe[i_Counter][2]==2)
                        {
                        printf("Player2 Wins");
                        win =1;
                        }
```

```
        }
//Check Columns
else if (tictactoe[0][i_Counter]==1&&tictactoe[1][i_Counter]==1&&tictactoe[2][i_Counter]==1)
        {
        printf("Player1 Wins");
        win =1;
        }
else if (tictactoe[0][i_Counter]==2&&tictactoe[1][i_Counter]==2&&tictactoe[2][i_Counter]==2)
        {
        printf("Player2 Wins");
        win =1;
        }
```

```
}
//Check Diagonal
        if ((tictactoe[0][0]==1&&tictactoe[1][1]==1&&tictactoe[2][2]==1)||
                (tictactoe[0][2]==1&&tictactoe[1][1]==1&&tictactoe[2][0]==1))
                {
                printf("Player1 Wins");
                win =1;
                }
        else if ((tictactoe[0][0]==2&&tictactoe[1][1]==2&&tictactoe[2][2]==2)||
                (tictactoe[0][2]==2&&tictactoe[1][1]==2&&tictactoe[2][0]==2))
                {
                printf("Player2 Wins");
                win =1;
                }
        //Board not empty and no one Wins
        else if(empty_Cells==0)
                printf("Draw");
        else
                printf("Intermediate");
```


janaki@janaki-HP-248-G1:~$ ./a.out
1
2
1
2
1
2
2
1
-1
Intermediatejanaki@janaki-HP-248-G1:~$ ./a.out
1
2
1
2
1
2
2
1
2
Drawjanaki@janaki-HP-248-G1:~$ gedit tictactoe.c