# Arrays and Strings

# Accident Problem

Each year the Department of Traffic Accidents receives accident count reports from a number of cities and towns across the country. Given details of ‘n’ days, develop an algorithm and write a program to determine the average number of accidents and for each day, print the difference between the number of accidents on that day and average. For example, if the number of days is 50 and the values are 10, 12, 15, 13, 5 then average is 11 and the difference of values are 1, 1, 4, 2, 6

# Accident problem

| Input | Output | Logic Involved |
| --- | --- | --- |
| Value of ‘n’, the number of days | Average and ‘n’ values that is the difference between average and value | Find average and difference |

# Algorithm

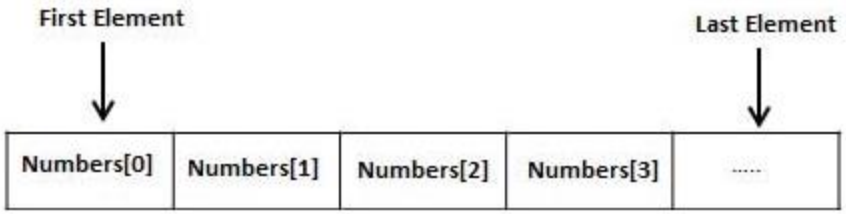
1. Read the value of ‘n’
2. Read the number of accidents happened in ‘n’ days
3. Find average
4. For each value print the difference between average and the value

# New Stuff...

- We can find the difference between average and the number of accidents on a particular day only after reading all numbers from the user
- So data has to be stored
- Same type of data is to be stored
- Number of items not known prior
- Best choice would be using arrays in C
- Array - Can store a fixed-size sequential collection of elements of the same type

# Arrays in C

- Consist of contiguous memory locations
- lowest address corresponds to the first element
- highest address to the last element
- Array indices start with zero
- The elements have indices from 0 to ‘n-1’
- Similar to list in Python but homogenous



# Declaration of Arrays in C

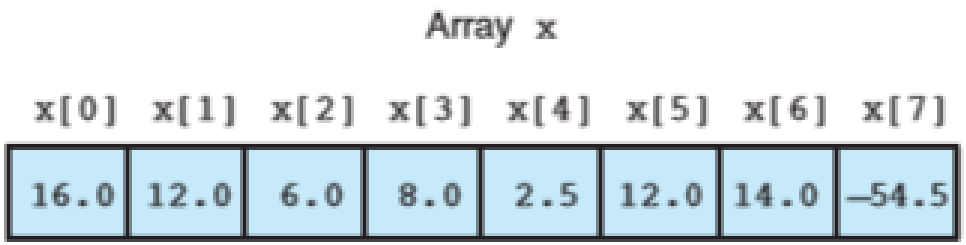
- type arrayName [ arraySize ];
- double balance[10];

**Initializing Arrays**

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

(or)

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};



# Difference between Initialization and Assignment

- Assignment
- int a;
- a = 5;
- a=10;

Intialization

- int a = 5;
- char gender=‘M’;

## Code Fragment That Manipulates Array x

| Statement | Explanation |
|---|---|
| `i = 5;` | |
| `printf("%d %.1f", 4, x[4]);` | Displays `4` and `2.5` (value of `x[4]`) |
| `printf("%d %.1f", i, x[i]);` | Displays `5` and `12.0` (value of `x[5]`) |
| `printf("%.1f", x[i] + 1);` | Displays `13.0` (value of `x[5]` plus 1) |
| `printf("%.1f", x[i] + i);` | Displays `17.0` (value of `x[5]` plus 5) |
| `printf("%.1f", x[i + 1]);` | Displays `14.0` (value of `x[6]`) |
| `printf("%.1f", x[i + i]);` | Invalid. Attempt to display `x[10]` |
| `printf("%.1f", x[2 * i]);` | Invalid. Attempt to display `x[10]` |
| `printf("%.1f", x[2 * i - 3]);` | Displays `-54.5` (value of `x[7]`) |
| `printf("%.1f", x[(int)x[4]]);` | Displays `6.0` (value of `x[2]`) |
| `printf("%.1f", x[i++]);` | Displays `12.0` (value of `x[5]`); then assigns `6` to `i` |
| `printf("%.1f", x[--i]);` | Assigns `5` (`6 - 1`) to `i` and then displays `12.0` (value of `x[5]`) |
| `x[i - 1] = x[i];` | Assigns `12.0` (value of `x[5]`) to `x[4]` |
| `x[i] = x[i + 1];` | Assigns `14.0` (value of `x[6]`) to `x[5]` |
| `x[i] - 1 = x[i];` | Illegal assignment statement |

# Cannot assign one array to other

```
int ia[] = {0, 1, 2}; // ok: array of int's
int ia2[] = ia; // error: cannot initialize one array with another
int main()
{
const int array_size = 3;
int ia3[array_size];
// ok: but elements are uninitialized!
ia3 = ia; // error: cannot assign one array to another
return 0;
}
```

```c
#include<stdio.h>
#include<stdlib.h>
void main()
{
int acc_Details[20],total=0;
int num_Of_Days,counter;
float mean;
scanf("%d",&num_Of_Days);
for (counter=0;counter<num_Of_Days;counter++)
{
scanf("%d",&acc_Details[counter]);
total+=acc_Details[counter];
}
mean = total/(float)num_Of_Days;
printf("%.2f\n",mean);

for (counter=0;counter<num_Of_Days;counter++)
{
printf("%.2f\n",fabsf(mean-acc_Details[counter]));
}
```

# Huffman Coding Problem

**Huffman code** is a particular type of optimal prefix code for characters. It is commonly used for lossless data compression. It is a variable-length code derived from frequency of occurrence. Given a string develop an algorithm and write a C program to determine frequency of occurrence of each character in the string.

# Huffman Coding problem

| Input | Output | Logic Involved |
| --- | --- | --- |
| A string S | Frequency count of each letter in S | Convert all letters to uniform case and check if it is a particular letter and increment corresponding count |

# Algorithm

1. Read a string
2. Make all letters in the string to be in lowercase
3. Process character by character
4. If the character is an alphabet then increment count of it
5. Print count of all alphabets

# Strings in C

- No data type string
- Represented as array of characters

# Character Arrays Are Special

- Can be initialized with either a list of comma-separated character literals enclosed in braces or a string literal
- Two forms are not equivalent
- String literal contains an additional terminating null character(NULL)

# Get string input

char name[10];
scanf(“%s”,name);  -> raju  -> {‘r’,’a’,’j’,’u’,’\0’};
printf(“%s”,name);  -> raju

char a[]={‘r’,’a’,’j’,’u’};
char b[]={‘r’,’a’,’m’,’u’};
printf(“%s”,a);  -> raju

To add a null:
char a[]= {‘r’,’a’,’j’,’u’,’\0’);  ->raju
printf(“%s”,a);

# New compilers also allow..

char a[50]=“I like C Programming”;

char b[]=“I like C Programming”;

# Strcpy

- used to copy a string and can be as strcpy(destination, source)
- Will not perform any boundary checking, and thus there is a risk of overrunning the strings. It will not check the size of the two char array.
- Add **string.h** header file.

```
#include <stdio.h>
#include <string.h>
void main() {
   char str1[20] = "C programming";
   char str2[20];
   // copying str1 to str2
   strcpy(str2, str1);
   printf("%s", str2); // C programming
}
```

# Strcmp

- used to compare two strings and can be used as strcmp(str1, str2)
- If the first string is greater than the second string a number greater than null is returned.
- If the first string is less than the second string a number less than null is returned.
- If the first and the second string are equal a zero (null0  is returned.

# Strcpy

```
#include<stdio.h>
#include<string.h>
void main(){
char a[10];
scanf("%s",a);
printf("\n%s",a);
char b[]={'a','b','c','\0'};
printf("\n%s",b);
printf("\n%d",strcmp(a,b));
return 0;
}
```

```
Output
:

abc
abc
0
```

# Strcpy

```
#include<stdio.h>
#include<string.h>
void main(){
char a[10];
scanf("%s",a);
printf("\n%s",a);
char b[]={'a','b','c','\0'};
printf("\n%s",b);
printf("\n%d",strcmp(a,b));
return 0;
}
```

```
Output
:

raju
abc
17
```

# Strcpy

```
#include<stdio.h>
#include<string.h>
void main(){
char a[10];
scanf("%s",a);
printf("\n%s",a);
char b[]={'a','b','c','\0'};
printf("\n%s",b);
printf("\n%d",strcmp(a,b));
return 0;
}
```

```
Output
:

aaa
abc
-1
```

# strcat

- concatenates a string onto the end of the other string and the resultant string is returned
- strcat() will not perform any boundary checking, and thus there is a risk of overrunning the strings.

```
printf("Enter you age: ");
scanf("%s", age);
result = strcat( age, " years old." ) == 0 )
printf("You are %s\n", result);
```

# strcat

```
#include <stdio.h>
#include <string.h>
void main() {
   char str1[10] = "Hello ";
   char str2[] = "VIT";
   strcat(str1, str2);
   printf("\n %s",str1);
   printf("\n %s",str2);
}
```

# strlen

- returns the length of a string
- All characters before the null termination

```
name = "jane";
result = strlen(name); //Will return size of four.
```

# Two-dimensional Arrays

type arrayName [ x ][ y ];

Where **type** can be any valid C data type and **arrayName** will be a valid C identifier.

A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.

# Initializing Two-Dimensional Arrays

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
   {0, 1, 2, 3} ,   /*  initializers for row indexed by 0 */
   {4, 5, 6, 7} ,   /*  initializers for row indexed by 1 */
   {8, 9, 10, 11}   /*  initializers for row indexed by 2 */
};
```

The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

# Accessing Two-Dimensional Array Elements

```
#include <stdio.h>
int main () {
   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;
   /* output each array element's value */
   for ( i = 0; i < 5; i++ ) {
      for ( j = 0; j < 2; j++ ) {
         printf("a[%d][%d] = %d\n", i,j, a[i][j] );
      }
   }
   return 0;
}
```

# Accessing Two-Dimensional Array Elements

When the above code is compiled and executed, it produces the following result −

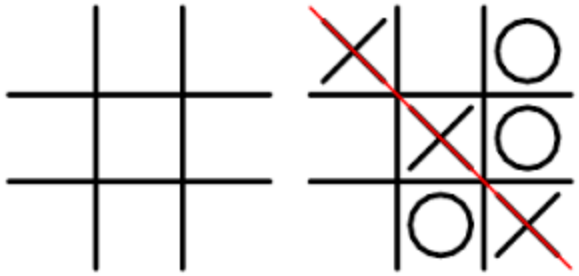
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8

# Holiday Assignment

## Tic Tac Toe Problem

# 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 |

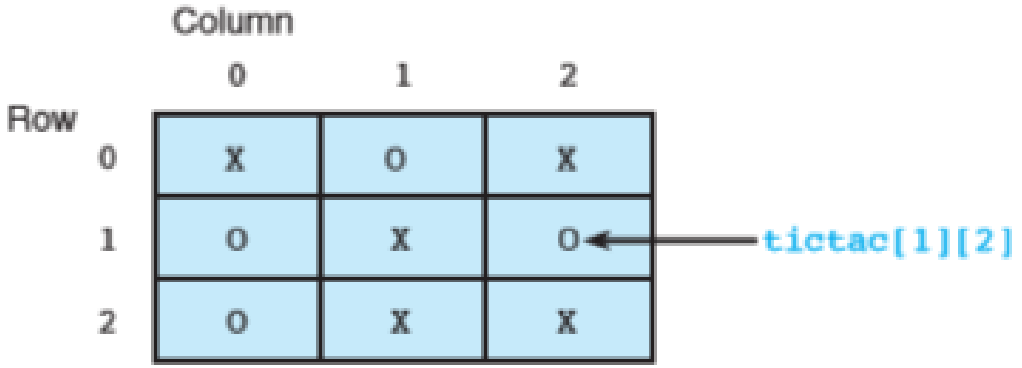
# Algorithm

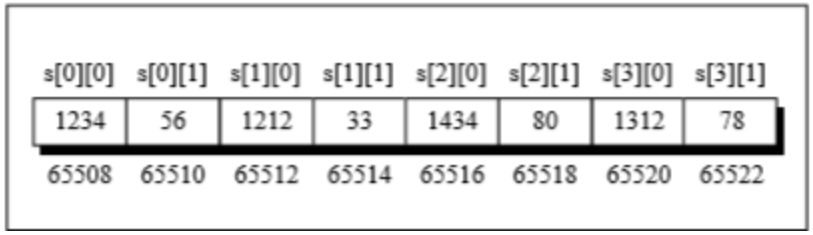
- Represent the board in memory
- Get the elements in first row, second row and so on
- Process the elements
- If all are -1 then print ‘empty’
- If ‘1’ is placed row wise, column wise or diagonally then print ‘Player 1’ wins
- If ‘2’ is placed row wise, column wise or diagonally then print ‘Player 2’ 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

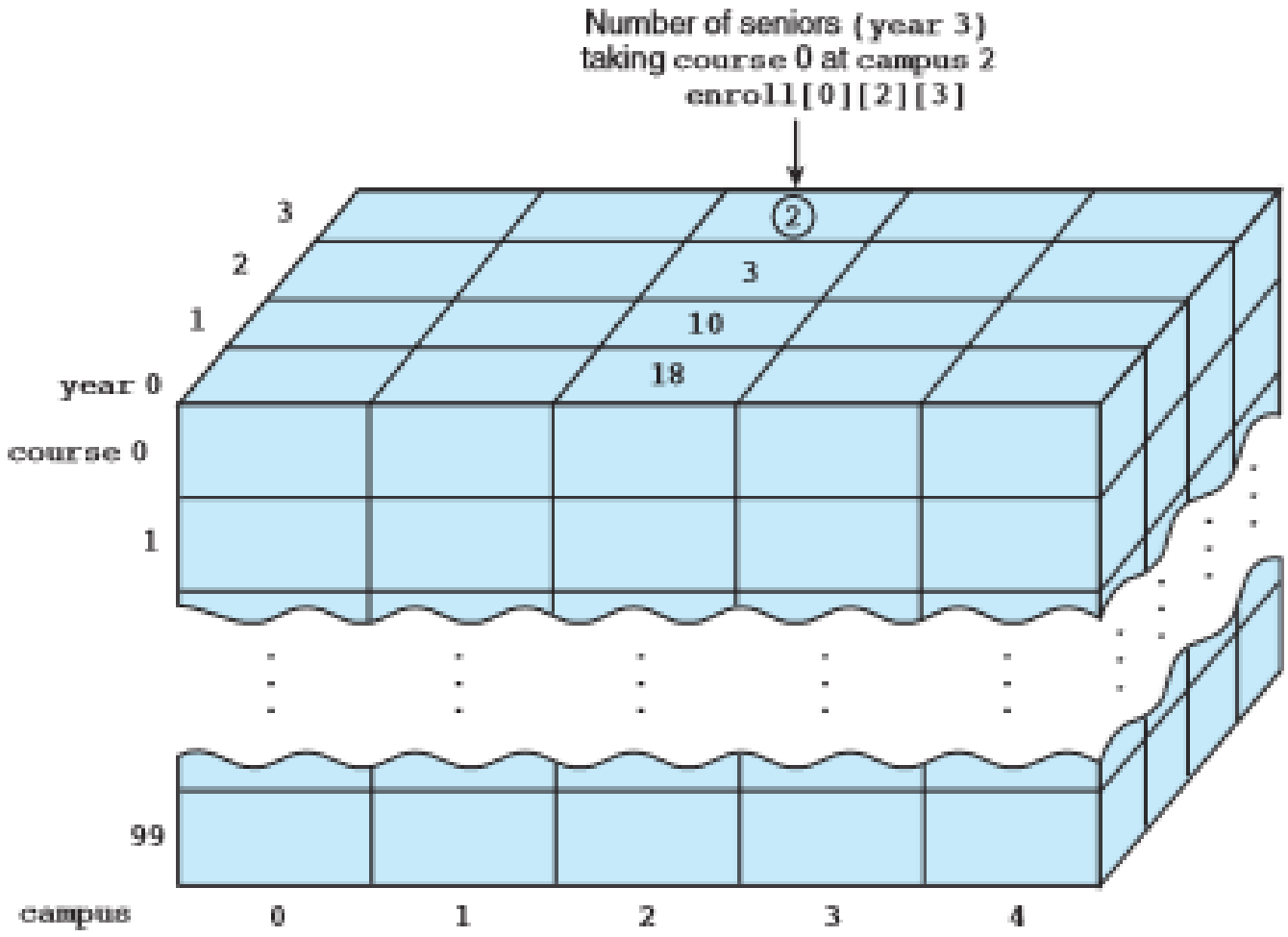
# Initialization of a Character 2 D Arrays

```
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");
```