ESC101: Introduction to Computing

Multi dimensional Arrays

Multidimensional Arrays

Multidimensional arrays are defined like this:

double mat[5][6]; OR **int mat[5][6];** OR **float mat[5][6];** etc.

The definition states that mat is a 5 X 6 matrix of doubles (or ints or floats). It has 5 rows, each row has 6 columns, each entry is of type double.

2.1	1.0	-0.11	-0.87	31.5	11.4
-3.2	-2.5	1.678	4.5	0.001	1.89
7.889	3.333	0.667	1.1	1.0	-1.0
-4.56	-21.5	1.0	-1.0	5.0	-5.78
45.7	26.9	-0.001	1000.09	15.1	1.0

Accessing matrix elements-I

1. **The (i,j)th member of mat is accessed as mat[i][j]. Note the slight difference from the matrix notation in maths.**
2. **The row and column numbering each start at 0 (not 1).**
3. **The following program prints the input matrix.**

```
void print_matrix(float mat[5][6]) {
    int i,j;
    for (i=0; i < 5; i=i+1) {
      for (j=0; j < 6; j = j+1) {
      printf("%f ", mat[i][j]);
      }
      printf("\n");
    }
}
```

/* prints the ith row i = 0..4. */

/* In each row, prints each of the six columns j=0..5 */

/* prints a newline after each row */

Accessing matrix elements-II

1. **Code for reading the matrix.**
2. **The address of the i,j th matrix element is &mat[i][j].**
3. **This works without parentheses since the array indexing operator [] has higher precedence than &.**

```
void read_matrix(float mat[5][6]) {
   int i,j;
   for (i=0; i < 5; i=i+1) {
        for (j=0; j < 6; j = j+1) {
      scanf("%f ", &mat[i][j]);
        }
    }
}
```

/* read the ith row i = 0..4. */

/* In each row, read each of the six columns j=0..5 */

scanf with %f option will skip over whitespace.

So it doesn't matter whether the entire input is given in 5 rows of 6 floats in a row or all 30 floats in a single line.

Practice Problem

- We are provided with a 3x3 matrix. We should output whether it is an identity matrix

Input : 1 0 0 Output: It is an identity matrix

0 1 0

0 0 1

Input: 2 1 0 Output: It is not an identity matrix

0 0 1

0 1 0

Solution for Practice problem

```
#include <stdio.h>
int main()
{
        int flag = 1;
        int mat[3][3];
        for(int i=0; i<3; i++)
                for(int j=0; j<3; j++)
                        scanf("%d",&mat[i][j]);
        //code to check whether matrix is identity

       return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
          //input read in mat
        for(int i=0; i<3; i++)
        {
                for(int j=0; j<3; j++)
                {
                        if(i!=j)
                        {
                                if(mat[i][j] != __ )
                                {
                                        flag = 0;
                                        break;
                                }
                        }
                        else
                                if(mat[i][i]!= __ )
                                {
                                        flag = 0;
                                        break;
                                }
                }
        }
        if(flag == 1)
                printf("matrix is identity\n");
        else
                printf("matrix is not identity\n");
        return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
          //input read in mat
        for(int i=0; i<3; i++)
        {
                for(int j=0; j<3; j++)
                {
                        if(i!=j)
                        {
                                if(mat[i][j] != 0 )
                                {
                                        flag = 0;
                                        break;
                                }
                        }
                        else
                                if(mat[i][i]!= __ )
                                {
                                        flag = 0;
                                        break;
                                }
                }
        }
        if(flag == 1)
                printf("matrix is identity\n");
        else
                printf("matrix is not identity\n");
        return 0;
}
```

Solution for Practice problem

```
#include <stdio.h>
int main()
{
          //input read in mat
        for(int i=0; i<3; i++)
        {
                for(int j=0; j<3; j++)
                {
                        if(i!=j)
                        {
                                if(mat[i][j] != 0 )
                                {
                                        flag = 0;
                                        break;
                                }
                        }
                        else
                                if(mat[i][i]!= 1 )
                                {
                                        flag = 0;
                                        break;
                                }
                }
        }
        if(flag == 1)
                printf("matrix is identity\n");
        else
                printf("matrix is not identity\n");
        return 0;
}
```

Coin Collection

You have an $n \times n$ grid with a certain number of coins in each cell of the grid. The grid cells are indexed by (i, j) where $0 \leq i, j \leq n - 1$.

For example, here is a 3x3 grid of coins:

	0	1	2
0	5	8	2
1	3	6	9
2	10	15	2

Coin Collection: Problem Statement

- You have to go from cell (0, 0) to (n-1, n-1).
- Whenever you pass through a cell, you collect all the coins in that cell.
- You can only move right or down from your current cell.

Goal: Collect the maximum number of coins.

Consider the example grid

5	8	2
3	6	9
10	15	2

There are many ways to go from (0,0) to (n-1,n-1)

5	8	2
3	6	9
10	15	2

Total = 35

5	8	2
3	6	9
10	15	2

Total = 25

5	8	2
3	6	9
10	15	2

Total = 31

5	8	2
3	6	9
10	15	2

Total = 30

5	8	2
3	6	9
10	15	2

Total = 23

5	8	2
3	6	9
10	15	2

Total = 36

Max = 36

Building a Solution

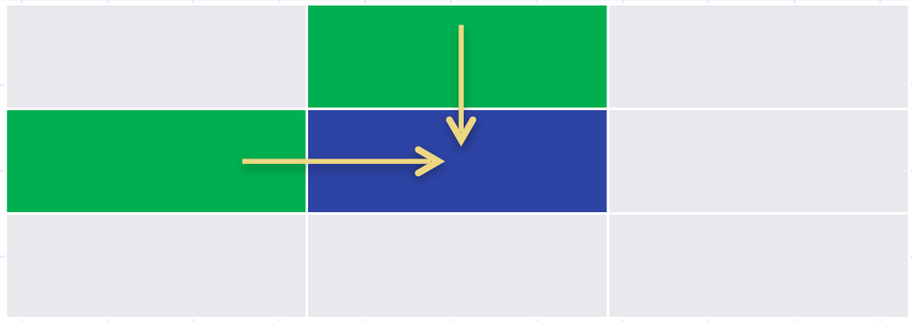
- We cannot afford to check every possible path and find the maximum.
  - Why?

In an $n \times n$ grid, how many such paths are possible?

- Instead we will iteratively try to build a solution.

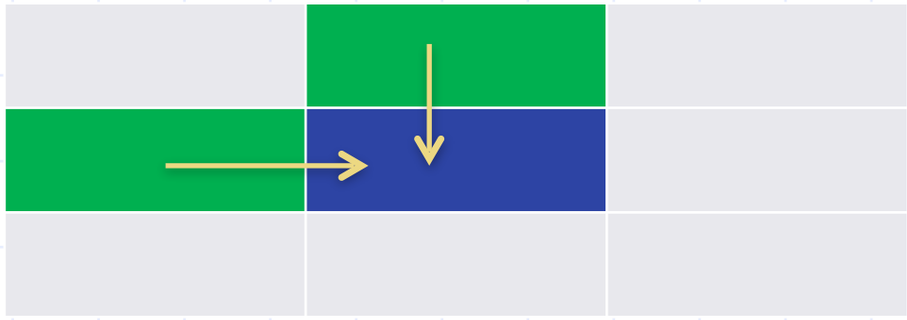
Solution Idea

- Consider a portion of some matrix



- What is the maximum number of coins that I can collect when I reach the blue cell?
  - This number depends only on the maximum number of coins that I can collect when I reach the two green cells.
  - Why? Because I can only come to the blue cell via one of the two green cells.

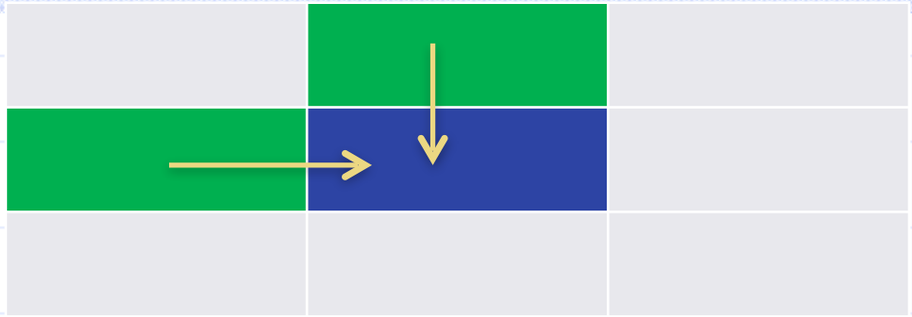
Solution Idea (dynamic programming)



Max-coins (bluecell) =

max(Max-coins (greencell1),
Max-coins (greencell2))
+ No. of coins (bluecell))

Solution Idea



- Let a(i,j) be the number of coins in cell(i,j)
- Let coin(i,j) be the maximum number of coins collected when travelling from (0,0) to (i,j).
- Then,

$$coin(i,j) = max(coin(i,j-1), coin(i-1,j)) + a(i,j)$$

Implementation

- Use an additional two dimensional array, whose (i,j)-th cell will store the maximum number of coins collected when travelling from (0,0) to (i,j).
- Fill this array one row at a time, from left to right.
- When the array is completely filled, return the (n-1, n-1)-th element.

Implementation: boundary cases

- To fill a cell of this array, we need to know the information of the cell above and to the left of the cell.
- What about elements in the top most row and left most column?
  - Cell in top row: no cell above
  - Cell in leftmost column: no cell on left
- Before starting with the other elements, we will fill these first.

```
int coin_collect(int a[][100], int n){
  int i,j, coins[100][100];

  coins[0][0] = a[0][0]; //initial cell

  for (i=1; i<n; i++) //first row
    coins[0][i] = coins[0][i-1] + a[0][i];

  for (i=1; i<n; i++) //first column
    coins[i][0] = coins[i-1][0] + a[i][0];

  for (i=1; i<n; i++) //filling up the rest of the array
    for (j=1; j<n; j++)
      coins[i][j] = max(coins[i-1][j], coins[i][j-1])
                    + a[i][j];

  return coins[n-1][n-1]; //value of last cell
}
```

```
int max(int a, int b){
  if (a>b) return a;
  else return b;
}

int main(){
  int m[100][100],i,j,n;

  scanf("%d", &n);
  for (i=0; i<n; i++)
    for (j=0; j<n; j++)
      scanf("%d", &m[i][j]);

  printf("%d\n", coin_collect(m,n));
  return 0;
}
```

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

5	13	15
8	0	0
18	0	0

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

5	13	15
8	0	0
18	0	0

5	13	15
8	19	28
18	0	0

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

5	13	15
8	0	0
18	0	0

5	13	15
8	19	28
18	0	0

5	13	15
8	19	28
18	34	0

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

5	13	15
8	0	0
18	0	0

5	13	15
8	19	28
18	0	0

5	13	15
8	19	28
18	34	0

5	13	15
8	19	28
18	34	36

Consider the example grid

5	8	2
3	6	9
10	15	2

Incremental stages of Coin 2D array

5	0	0
0	0	0
0	0	0

5	13	15
8	0	0
18	0	0

5	13	15
8	19	28
18	0	0

5	13	15
8	19	28
18	34	0

5	13	15
8	19	28
18	34	36

5	8	2
3	6	9
10	15	2

Total = 36

Passing two dimensional arrays as parameters

Write a program that takes a two dimensional array of type double [5] [6] and prints the sum of entries in each row.

```
int i,j; double rowsum;
for (i=0; i < 5; i=i+1) {
    rowsum = 0.0;
     for (j=0; j < 6; j = j+1) {
    rowsum = rowsum+mat[i][j];
     }
    printf("%lf ", rowsum);
  }
}
```

Question?

But suppose we have read only the first 3 rows out of the 5 rows of mat. And we would like to find the marginal sum of the first 3 rows.

Answer:

That's easy, we can take an additional parameter nrows and run the loop for i=0.. (nrows-1) instead of 0..5.

The slightly generalized program would be:

```
void marginals(double mat[5][6], int nrows) {
   int i,j; double rowsum;
   for (i=0; i < nrows; i=i+1) {
       rowsum = 0.0;
        for (j=0; j < 6; j = j+1) {
       rowsum = rowsum+mat[i][j];
       }
      printf("%lf ", rowsum);
    }
}
```

In parameter double mat[5][6], **C completely ignores the number of rows** 5.
It is only interested in the number of cols: 6.

Let's see more examples…

We declared mat to be of type double [5][6].
Does this mean that nrows should be <= 5?
We are not checking for it!

The following program is exactly identical to the previous one.

```
void marginals(double mat[ ][6], int nrows)
 {
   int i,j; int rowsum;
   for (i=0; i < nrows; i=i+1) {
       rowsum = 0.0;
        for (j=0; j < 6; j = j+1) {
       rowsum = rowsum+mat[i][j];
        }
       printf("%lf ", rowsum);
    }
}
```

1. Why? because C does not care about the number of rows, only the number of cols.
2. And why is that? We'll have to understand 2-dim array

Example…

This means that the above program works with a k X 6 matrix where k could be passed for nrows.

Example calls for marginals

```
void marginals(double mat[ ][6], int nrows);
void main() {
      double mat[9][6];
     /* read the first 8 rows into mat */
     marginals(mat,8);
}
```

```
void marginals(double mat[ ][6], int nrows);
void main() {
     double mat[9][6];
    /* read 9 rows into mat */
     marginals(mat,10);
}
```

UNSAFE

The 10th row of mat[9][6] is not defined. So we may get a segmentation fault when marginals() processes the 10th row, i.e., i becomes 9.

As with 1 dim arrays, allocate your array and stay within the limits allocated.

Number of columns

Why is the number of columns required?

- The memory of a computer is a 1D array!
- 2D (or >2D) arrays are "flattened" into 1D to be stored in memory
- In C (and most other languages), arrays are flattened using Row-Major order
  - In case of 2D arrays, knowledge of number of columns is required to figure out where the next row starts.
  - Last n-1 dimensions required for nD arrays

Row Major Layout

mat[3][5]

0,0	0,1	0,2	0,3	0,4
1,0	1,1	1,2	1,3	1,4
2,0	2,1	2,2	2,3	2,4

Layout of mat[3][5] in memory

0,0	0,1	0,2	0,3	0,4	1,0	1,1	1,2	1,3	1,4	2,0	2,1	2,2	2,3	2,4

mat[0] mat[1] mat[2]