MINI LECTURE 1:

ARRAY INITIALISATION & TWO DIMENSIONAL ARRAYS

COMP1002J: Introduction to Programming 2

Dr. Brett Becker (brett.becker@ucd.ie)

Beijing Dublin International College

Initialising Arrays

If I declare an array, but do not initialise any of the values, it can contain any (seemingly random) values:

```
#include <stdio.h>
#define LENGTH 20

int main() {
   int arr[LENGTH];
   int i = 0;
   for ( i = 0; i < LENGTH; i++ ) {
      printf( "%d; ", arr[ i ] );
   }
   printf("\n");
}
```

- This is the same situation for single variables.

file: arrinit0.c

Initialising Arrays

Output on my machine:

1958245226; -1637750671; 4199072; 4199072; 0; 4200544; 6422240; 6422296; 6422476; 1958269424; -356671643; -2; 1958245226; 1958245469; 4200544; 6422368; 4200638; 4200544; 0; 2134016;

- Why is this?

  These are *garbage values* – whatever values were left in memory by the last program(s) to use these addresses!

Initialising Arrays

- If I want it to have particular values, I should say what they are:

```
#include <stdio.h>
#define LENGTH 5

int main() {
   int arr[LENGTH] = { 0, 0, 0, 0, 0 };
   int i = 0;
   for ( i = 0; i < LENGTH; i++ ) {
      printf( "%d; ", arr[ i ] );
   }
   printf("\n");
}
```

file: arrinit1.c

Initialising Arrays

Output:

0; 0; 0; 0; 0;

Initialising Arrays

- If I specify the value of **any** of the array's contents when I declare it, all the others will automatically be set to 0.

```
#include <stdio.h>
#define LENGTH 100

int main() {
   int arr[LENGTH] = { 0 };
   int i = 0;
   for ( i = 0; i < LENGTH; i++ ) {
      printf( "%d; ", arr[ i ] );
   }
   printf("\n");
}
```

- This is a nice way to quickly create an array of 0s.

file: arrinit2.c

Initialising Arrays

- Output:

0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0;
0; 0; 0; 0;

Two Dimensional Arrays

- In C we are not limited to one dimensional arrays
- Arrays can have as many dimensions as we require!
- Two dimensional arrays are very common

Two Dimensional Arrays

- In C, two dimensional arrays are stored in ‘row-major order’
- In row-major order, consecutive elements of the rows of the array are contiguous in memory
- So the array $\mathrm{A} = \begin{matrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \end{matrix}$
- Is stored in the following order in memory:

Address	*x*	*x*+1*s*	*x*+2*s*	*x*+3*s*	*x*+4*s*	*x*+5*s*
Value	a_{00}	a_{01}	a_{02}	a_{10}	a_{11}	a_{12}

- Where *x* is the address of a_{00}, *s* is `sizeof(<type>)`, and `<type>` is the type of the array elements.

Two Dimensional Arrays

- In C, two dimensional arrays are stored in ‘row-major order’

- This also means that a[1][0] refers to row 1, column 0.

Two Dimensional Arrays

```
int cols = 10;
int rows = 10;
int myArray[rows][cols];
```

- We can then access array elements like so:

- Assign the value 1 to ‘row’ 4, ‘column’ 5:

```
myArray[4][5] = 1;
```

- Assign the value at ‘row’ 0, ‘column’ 3 to y:

```
int y = myArray[0][3];
```

Two Dimensional Arrays

```
#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, ", i,j, a[i][j] );
      }
           printf("\n");
   }
   return 0;
}
```

matrix1.c

Two Dimensional Arrays

Output:

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,

Two Dimensional Arrays

We can use ‘nested loops’ to process two dimensional arrays:

```
for ( i = 0; i < 3; i++ ) { // this loop iterates
                            //through rows
     for ( j = 0; j < 3; j++ ) { //this loop iterates
                                 //through columns
        printf("%d  ", mat[i][j]);
     }
     printf("\n");
  }
```

matrix2.c