Introduction to C

Session 5

March 25

Dr Mark Doughty, School of Engineering and Physical Sciences,
University of Lincoln.

Memory Address of Variables

- Each variable has a location in memory when it is created
- We can determine the address of a variable with the '&' operator

```
int x = 5;
printf("%d", &x);
```

*Will print out the memory address of 'x'.

A memory address is just a number – an integer, why we use %d

```
int x = 5;
int *px = &x;
printf("%d", *px);
```

*We can assign this address to another variable – a 'pointer'

`int *px` means that 'px' is a pointer (the '*') to an int

In this context, '*' is the 'dereference' operator.

*px is 'what value is at the address pointed to by px'

A pointer is just a variable which holds a memory address rather than a value

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointers and Scope

- When a variable pointed to goes out of scope, it becomes unavailable
  - The pointer then points at nothing – it is a ‘null’ pointer (its memory address contents are removed)
  - This can be a problem if you still expect it to point to something!

```
char* message(){
    char m[] = "Hello";
    return m;
}
int main(int argc, char **argv)
{
    char *str = message();
    printf("%s", str);
    return 0;
}
```

In this example, the ‘message’ function, returns a pointer.

But as the variable ‘m’ goes out of scope when the function returns, the pointer is ‘null’.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointers and Scope

- We can pass pointers to functions to make sure they don't go out of scope:

```c
void swap(int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main(int argc, char **argv)
{
    int p = 1;
    int q = 2;
    printf("%d %d\n", p, q);
    swap(&p, &q);
    printf("%d %d", p, q);
    return 0;
}
```

Output:
1 2
2 1

'swap' accepts two integer pointers.

These pointers point to the two variables p and q.

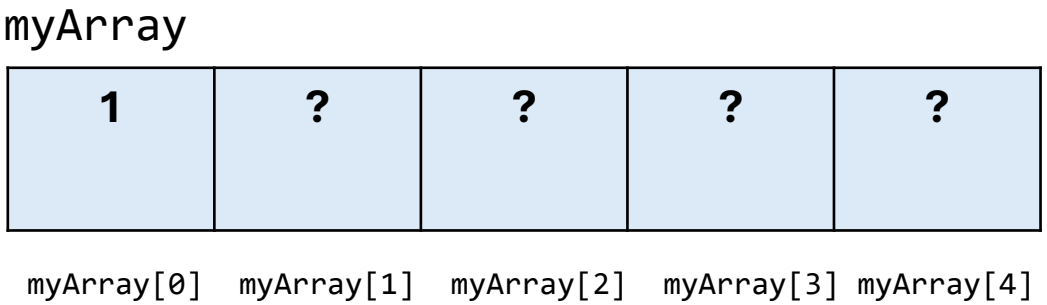
As p and q do not go out of scope when swap returns, swap operates on the original p and q variables.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointers and Arrays

- An array is a collection of identically typed items

```
int myArray[5];
myArray[0] = 1;
printf("%d", myArray[0]);
```



Creates an array that **can only** hold 5 ints.

Assign a value to the first element in the array.

Output that value in the first array element.

Beware! C doesn't check array boundaries – so if you try to access, say, myArray[5], you will get an unexpected value

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointers and Arrays

- Pointers and arrays in C are closely linked.
- `myArray` is like a pointer to element 0 in the array.

```
int myArray[5];
myArray[0] = 1;
int *p = myArray;
printf("%d", *p);
```

Output:
1

Here, we have created an integer pointer, p, and assigned `myArray` to it.

By dereferencing it to print the value held by that pointer, we can see it is element 0 of `myArray`.

Shows that the array identifier ('myArray') is a pointer to the first element in its collection.

We could get all elements after this by incrementing the pointer.

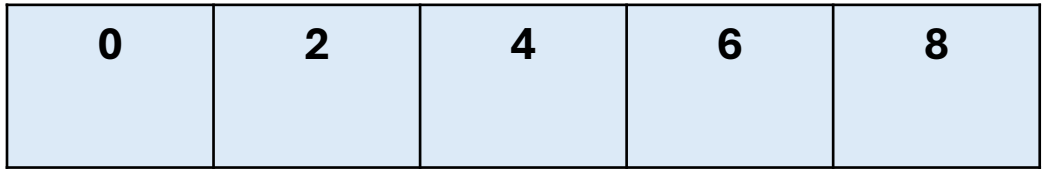
Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointers and Arrays

```
int myArray[5];
for(int i=0;i<5;i++){
    myArray[i] = i*2;
}
int *p = myArray;
printf("%d", *(p+2));
```

Output:
4

myArray



Element 0	Element 1	Element 2	Element 3	Element 4
*p	*(p+1)	*(p+2)	*(p+3)	*(p+4)

We have incremented the pointer by 2.

i.e. The pointer has moved 2 x the sizeof(int) to element 2

Be careful here:

*p+2 means ‘add two to the value at *p.
*(p+2) means move the pointer two units.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Have a look here:
https://www.codeproject.com/Articles/7042/How-to-interpret-complex-C-Cplusplus-declarations

Right – Left Rule

- When deciphering variable declarations in C, it is useful to read right to left. For example:

```
int x;
```

Start at the identifier: '**x is**'
Read leftwards: '**an int**'
'**x is an int**'

```
float *y;
```

'y is'
'a pointer to'
'a float'
'**y is a pointer to a float**'

```
char *c[];
```

'c is' **(move right until you run out of symbols, or hit right parenthesis)**
'an array of'
'pointers to'
'char'
'c is an array of pointers to char'

```
char (*c)[];
```

'c is'
'a pointer'
'to array'
'char'
'c is a pointer to an array of chars'

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Right – Left Rule

- So, for:

```
float *(*ptr)[];
```

What is ‘ptr’ in this example…?

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Right – Left Rule

- So, for:

```
float *(*ptr)[];
```

‘ptr is’
‘a pointer’
‘to an array’
‘of pointers’
‘to floats’

ptr is a pointer to an array of float pointers

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Pointer Arithmetic

- When you add an integer value to a pointer – you are changing its address – not the value it points to

```
char charArray[5];
char *c = charArray;
```

Say, `charArray` is at memory location 100.

*(c+2) moves the pointer 2 x the size of a char (location: 102)

```
int intArray[5];
int *i = intArray;
```

Say, `intArray` is at memory location 100.

*(i+2) moves the pointer 2 x the size of an int (location: 108)

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Arrays of *char* - strings

```
char str[10];
printf("Enter 'hello'\n");
fgets(str, 10, stdin);
for(int i=0;i<10;i++){
    printf("%d ", str[i]);
}
```

This will get user input and put it into ‘str’.

We output integers though (‘%d’) for each element of str.

What does it output?

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Arrays of *char* - strings

```
char str[5];
printf("Enter 'hello'\n");
fgets(str, 5, stdin);
for(int i=0;i<5;i++){
    printf("%d ", str[i]);
}
```

This will get user input and put it into ‘str’.

We output integers though (‘%d’) for each element of str.

5 spaces in the array for 5 characters…What does it output?

104 101 108 108 0

These are the ascii values for the letters ‘hell’ (See https://www.asciitable.com/)

What is the ’0’ though?

The ‘0’s are the NULL character. The char representation for NULL is ‘\0’

fgets expects 5 characters – the last one being the NULL character.

Increase the number of chars it is expecting to read:

```
fgets(str, 10, stdin);
```


Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

Example: A simple array search

```c
#include <stdio.h>
#include <stdlib.h>
#define SIZE 200

int linear(int value, int* arrayToSearch, int numberOfItems){
    //return '1' if value is in array, or '0' if it isn't
    int *pArray;
    int *pArrayEnd = arrayToSearch + numberOfItems;

    for(pArray = arrayToSearch; pArray < pArrayEnd; pArray++){
        if(*pArray == value) return 1;
    }

    return 0;
}

int main(int argc, char **argv)
{
    int valueToSearch = 201;
    int arr[SIZE];
    for(int i=0; i<SIZE;i++){
        arr[i] = i+1;
    }
    if(linear(valueToSearch, arr, SIZE)==1){
        printf("Success!");
    }else{
        printf("Not in collection");
    }
    return 0;
}
```

Need to use a #define for array size – unless setting it directly (needs to be a constant integer)

Increment the pointer by the array size to determine the end of the array

Set the array pointer to the start of the array. Iterate through it until it reaches the end.

Dr Mark Doughty, School of Engineering and Physical Sciences, University of Lincoln.

In conclusion …

- In this session, we have covered:
  - Pointers
    - Concept
    - Scope
    - Arithmetic
    - Right-Left deciphering rule
  - Arrays
    - Concept
    - Link to pointers
    - Array of `char`
    - Using pointers to move through arrays