Pointers and Coursework

1 Function Pointers

- We've seen pointers to variables. We can also have pointers to functions!

```c
#include<stdio.h>
void hello_function(int times);

int main(){
  void (*func_ptr)(int);
  func_ptr=hello_function;
  func_ptr(3);
  return 0;
}

void hello_function(int times){
  for(int i=0;i<times;i++) {
    printf("Hello, World!\n");
  }
}
```

2 Using `qsort()`

- `stdlib.h` contains an implementation of the quicksort algorithm. The function declaration is:

```c
void qsort(void *base, size_t nmemb, size_t size,
    int (*compar)(const void *, const void *))
```

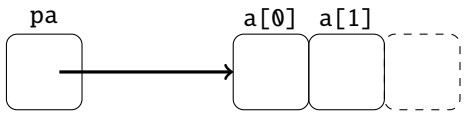
- `void *base` is a pointer to the array
- `size_t nmemb` is the number of elements in the array
- `size_t size` is the size of each element
- `int (*compar)(const void *, const void *)` is a function pointer composed of two arguments and returns `0` when the arguments have the same value, `<0` when `arg1` comes before `arg2`, and `>0` when `arg1` comes after `arg2`.

```c
#include <stdio.h>
#include <stdlib.h>
int compare (const void *, const void *);
int main() {
  int arr[] = {52, 14, 50, 48, 13};
  int num, width, i;
  num = sizeof(arr)/sizeof(arr[0]);
  width = sizeof(arr[0]);
  qsort(arr, num, width, compare);
  for (i = 0; i < 5; i++)
    printf("%d ", arr[i]);
  printf("\n");
  return 0;
}

int compare (const void *arg1, const void *arg2) {
  return *(int *)arg1 - *(int *)arg2;
}
```

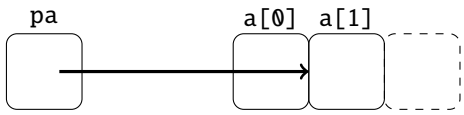
3 Recap — Pointer arithmetic

- Pointer arithmetic accounts for the base type of the items:

```
int a[10];
int *pa;

pa = &a[0];
pa = a;
```




```
pa = &a[1];
pa = (a+1);
```



- The two pairs of statements above are equivalent using array or pointer notation: `+1` translates to `+4` bytes (1 `int`)

4 Recap — Strange but true

- In C if I write `a[x]` this works by adding `x` to `a` to find the pointer
- Hence `a[x]` is the same as `*(a+x)`
- This seems fine if I write `a[2]`
- But what if I write `2[a]`?
- It compiles and works!
- We can also have multi-dimensional arrays in C e.g.

```
int matrix[2][3] = {{1,2,3},{4,5,6}};
```

- Now `matrix[0][1]==2`
- Can have more than 2-dimensional arrays:

```
int arr3d[3][2][4] = {
    {{1, 2, 3, 4}, {5, 6, 7, 8}},
    {{9, 10, 11, 12}, {13, 14, 15, 16}},
    {{17, 18, 19, 20}, {21, 22, 23, 24}}
};
```

- The elements of `arr3d` will be allocated in memory in the order `arr3d[0][0][0]`, `arr3d[0][0][1]`, `arr3d[0][0][2]`, `arr3d[0][0][3]`, `arr3d[0][1][0]`, `arr3d[0][1][1]` etc.

```
int arr3d[3][2][4] = {
    {{1, 2, 3, 4}, {5, 6, 7, 8}},
    {{9, 10, 11, 12}, {13, 14, 15, 16}},
    {{17, 18, 19, 20}, {21, 22, 23, 24}}
};
```

- Now `&arr3d[i][j][k]` is the same as `&arr3d[0][0][0]+(i*2*4)+j*4+k`
- What is the type of `arr3d[0][0][0]`?

- What is the type of `arr3d[0][0]`?
- What is the type of `arr3d[0]`?
- What is the type of `arr3d`?
- What does `int (*p)[2][4]=arr3d;` do?
- For further fun with pointers and arrays, read https://www.oreilly.com/library/view/understanding-and-using/9781449344535/ch04.html