Higher order functions

- Functions that take other functions as parameters
- Easily supported in languages that are dynamically typed and where functions treated as a type of data
- More complex in statically typed languages where functions are not inherently a data type (like C/C++, need to resort to function pointers)
- How does type checking work for passed function?

Lisp

- we’ve looked at lisp, where we can easily pass functions and where callee can check passed item at run time to see if it really is a function
- Variety of built-in mechanisms to support calling the passed function on other data
- Type checking is done at run time, as with rest of lisp, so potential type issues deferred until point where relevant data is used on invoked function

C / C++

- Function identifiers are actually pointers to the location in memory where the relevent executable instructions are stored
- Can pass name of one function to another, as long as the profile (number/type/order of parameters) of the actual parameter of the passed function matches the profile of the corresponding parameter
- sample syntax

```
void higherOrder(char (*func)(int, double)) {
    Char c = (*func)(10, 3.14);
}
```

func is the function pointer, int, double are parameter types it is expecting, and char is the return type, sample call to higherOrder might be

```
C = higherOrder(somefunction)
```

C++ and templates

- If using C++, we can make the profile for the formal parameter more flexible by giving some or all of its parameters/return type templated types
- Here we make one of the parameters a templated type, can pass any function that returns a char and takes two parameters, the second of which is a double

```
template <class T>
void higherOrder(char (*func)(T, double));
```