COMP2521

Function Pointers in C

Function Pointers

- C can pass functions by passing a pointer to them.
- Function pointers ...
  - are references to memory addresses of functions
  - are pointer values and can be assigned/passed

- Function pointer variables/parameters are declared as:

```
typeOfReturnValue (*fp)(typeOfArguments)
```

- In the following example, **fp** points to a function that returns **int** and have one argument of type **int**.

```
int (*fp)(int)
```

Function Pointers

```
int square(int x) { return x*x;}

int timesTwo(int x) {return x*2;}

int (*fp)(int);

fp = &square;        //fp points to the square function

int n = (*fp)(10); //call the square function with input 10

fp = timesTwo;       //works without the &
                    //fp points to the timesTwo function

n = (*fp)(2);        //call the timesTwo function with input 2

n = fp(2);           //can also use normal function call
                    //notation
```

Higher-order Functions

- Functions that get other functions as arguments, or return functions as a result
- **Example:** the function **traverse** takes a list and a function pointer (**fp**) as argument and applies the function to all nodes in the list

```
void traverse (list ls, void (*fp) (list) ){
    list curr = ls;
    while(curr != NULL){
        // call function for the node
        fp(curr);
        curr = curr->next;
    }
}
```

Second argument is **fp**,
Pointer to a function like,
void functionName(list n)

Higher-order Functions: Example

```
void printNode(list n){
    if(n != NULL){
        printf("%d->",n->data);
    }
}
```

```
void printGrade(list n){
    if(n != NULL){
        if(n->data >= 50){
            printf("Pass");
        }
        else {
            printf("Fail");
        }
    }
}
```

```
void traverse (list ls, void (*fp) (list));


//The second argument must have matching prototype

traverse(myList,    printNode);
traverse(myList,    printGrade);
```