A Level Computer Science

Topic 4: Functions and Recursion
Aims

• Recursion the idea
  • Problem solving by divide and conquer

• Revision of functions

• Programming with recursion
Recursion

Recursion is a fundamental idea
Example: Factorial

- $5! = 5 \times 4 \times 3 \times 2 \times 1$
- Factorial can be defined using factorial:
  \[
  \text{factorial}(N) = N \times \text{factorial}(N-1)
  \]
- But is this useful?

1. **Base Case:** \(\text{factorial}(0) = 1\)
2. **Recursive case:**
   \[
   \text{factorial}(N) = N \times \text{factorial}(N-1), \text{ provided } N > 0
   \]
Factorial Example

1. **Base Case:** \( \text{factorial}(0) = 1 \)

2. **Recursive case:**
   \[
   \text{factorial}(N) = N \times \text{factorial}(N-1), \text{ provided } N > 0
   \]

\[
\text{factorial}(4) = 4 \times \text{factorial}(3) \\
= 4 \times 3 \times \text{factorial}(2) \\
= 4 \times 3 \times 2 \times \text{factorial}(1) \\
= 4 \times 3 \times 2 \times 1 \times 1
\]
Recursion

• Define something in terms of itself
  • Recursive case – smaller instance of the problem
  • Base case – smallest instance

• Some algorithms are elegantly stated recursively
• Recursive implementation
  • factR calls factR – recursive call

• Divide and conquer
**Binary Search (Again)**

- Search a sorted array – is E in the array?

- **Base case:**
  - empty array or E found

- **Recursive case:**
  - Compare E with the middle element
  - If E smaller, search the left half
  - If E larger, search the right half
Recursive Sorting

• Concept
  • Split array in two halves, sort each half
  • Combine two sorted arrays
  • Single item sorted (base case)

• Two algorithms
  • Merge sort
    • Halve array – merge two sorted lists
  • Quicksort
    • Partition array – combine easy
Merge Sort – Insight

• How could we share out sorting between two people?
  • Half it and sort each half
  • Merge the two sorted lists

• When there is only a single entry – it is sorted
Merge Sort

17 31 52 19 41 34 76 11
Decompose
17 31 52 19 41 34 76 11
Recursion
17 19 31 52 11 34 41 76
Merge
11 17 19 31 34 41 52 76
Merging

• Work done in the merge

• Repeatedly select smallest

\[
\begin{array}{cccc}
11 & 34 & 41 & 76 \\
17 & 19 & 31 & 52 \\
11 & 17 & 19 & 31 & 34 & 41 & 52 & 76
\end{array}
\]
Quicksort – Insight

• How could we share out sorting between two people?
  • Choose a value V
  • Give first person all values < V
  • Give second person all values > V

• When there is only a single entry – it is sorted
Quicksort Example

17  31   52   19  41   34  76   11  28

28   17  19   11  31   34  76   52  41

all < 31

all >= 31

11  17  19   28

41  34  52  76

19   28

34   41
Quicksort Description

• Choose a pivot value
• Partition the array
  • Values less than the pivot to the left
  • The pivot
  • Values greater than the pivot to the right
• Repeat process on each partition
• … until partition has no more than one value

• Work done in partition
Exercise 1.1-1.2

• Demonstrate merge sort using playing cards. Can you make the use of recursion clear?
• Also try Quicksort
Properties

• Merge sort
  • $O(N \log N)$ – same as quick sort but extra space
  • Stable

http://www.sorting-algorithms.com/merge-sort

• Quicksort
  • More efficient: $O(N \log N)$
  • Not stable

http://www.sorting-algorithms.com/quick-sort
To write recursive programs we need a good understanding of functions
Why Functions?

- Code organisation
  - Functions allow code to be organised in parts
  - Top-down development

- Code reuse
  - The library
  - Functions with parameters: existing code, your data

- Recursion
Elements of Functions

• Name and definition
  
  Issue: function must be called

• Parameters and return
  
  Issue: return versus print

• Local variables and scope
  
  Issue: where are the variables
Simple Functions

- No parameters
- No return

**Issue**
- A function must print or return
- Return and print are confused

**Suggestions**
- Always use return
- Avoid print

```python
def greet():
    print("Hello")

def greetByName(name):
    print("Hello", name)

def greet():
    print("Hello")
    return

def greetByName(name):
    return "Hello" + name
```
def fun1(p):
    p = p + 1
    return p

a = 1
b = fun1(a)
print("argument a =", a)
print("return value b =", b)

• What is the result?
• Is variable a changed?
Exercise: 2.1

- Discuss with another teacher
  - Explaining definition versus call
  - Explaining print versus return
Function Call Tree

• One function calls another

```python
def doub(p):
    return p*2
def quad(p):
    d = doub(p)
    return doub(d)
n = int(input("Number: "))
print("4 x", n, ", quad(n)")
```

```
main
    |
    v
quad
call
    |   return
    v
doub
    |   return
```
Function Scope

- Scope: dictionary of variables

```python
def doub(p):
    return p * 2

def quad(p):
    d = doub(p)
    return doub(d)

n = int(input("Number: "))
print("4 x", n, "=", quad(n))
```

- $p \rightarrow \text{integer (6)}$
- $d \rightarrow \text{integer (6)}$
- $n \rightarrow \text{integer (3)}$
- $\text{doub} \rightarrow \ldots \text{code}$
- $\text{quad} \rightarrow \ldots \text{code}$
Global Variables

- A variable defined in a function is **local**
- Global variables can be read
- Update a global using 'global'

```python
def f():
    global g1
    g1 = g1 + g2

    g1 = 1
    g2 = 3
    f()

    print("g1 = ", g1)
    print("g2 = ", g2)
```

**Update g1**

**Read g2**

**Globals instead of parameters**
**Better to use parameters**
**Worse to mix**
def fun1(thelist):
    thelist.append(41)

myl = [2,3,4]
fun1(myl)
print("myl =", myl)

• What is the result?
• Is the list variable a changed?

Just like assignment a list: … variable refers to the list … parameter refers to the list
No global: reference is read
Exercise: 2.2-2.3

• Implement ‘shopping list’ functions.
def BSearch(A, target):
    left = 0
    right = len(A)
    while right > left:
        mid = (left + right) // 2
        if A[mid] == target:
            return mid
        elif A[mid] < target:
            left = mid+1
        else:
            right = mid
    return -1
Programming with Recursion
Recursive Algorithms & Programs

• Algorithms described recursively
  • Each stage like the whole

• Can be implemented iteratively
  • Uses a loop

• Can be implemented recursively
  • No loop
  • Recursive function call
Factorial: Iterative and Recursive

```python
def factR(n):
    if n <= 0:
        return 1
    else:
        return n * factR(n-1)
```

```python
def factI(n):
    f = 1
    for i in range(n, 0, -1):
        f = f * i
    return f
```
Exercise: 3.1

- Implement factorial using both recursion and a loop.
- Which do you prefer?
Binary Search (Again)

- Base case:
  - empty array or element found

- Recursive case: search array
  - search either left half or right half of array

```python
def findBRec(A, target, left, right):
    ...
    mid = (left + right) // 2
    ...
    return findBRec(A, target, mid+1, right)
    ...
    return findBRec(A, target, left, mid)
```
Merge Sort – Algorithm

mergeSort(Array)
    L = length of Array
    if L <= 1 then it is sorted, return it
    else
        mid = L // 2
        A1 = mergeSort(Array from 0 up to mid)
        A2 = mergeSort(Array from mid up to L)
        return merge of A1 and A2

• Note: merge sort does not work ‘in place’ – need to copy data items
def mergeSort(A):
    L = len(A)
    if L < 2:
        return A
    m = L // 2
    A1 = mergeSort(A[0:m])
    A2 = mergeSort(A[m:L])
    return merge(A1, A2)

def merge(A1, A2):
    if len(A1) == 0: return A2
    elif len(A2) == 0: return A1
    else:
        if A1[0] < A2[0]:
            M = [A1[0]]
            M.extend(merge2(A1[1:], A2))
        else:
            M = [(A2[0])]
            M.extend(merge2(A1, A2[1:]))
    return M
Merge Sort Example

• Print sorted lists, base case and after merge

```python
global mergeSort
mergeSort([23, 21, 54, 17, 62, 25, 19, 11])
```

Base case
Merge
Merge
Merge
Exercise 3.3

• Complete the recursion implementation of binary search
Summary
Summary

• Recursion
  • Way of thinking
  • Equivalent to loops

• Many algorithms elegantly expressed recursively

• Recursive functions
  • … must understand functions