

# **C102 – Day 30**

**Recursion**  
**Abstraction**  
**Translation**

# Repetition

- **In some situations, we don't know exactly how many times a block of instructions should be repeated.**
- **All we know is that repetition is needed**
  - **For example, in a board game like chess or checkers, we don't know exactly how many moves it will take for a player to win or lose the game – all we know is that several moves will be needed**

# Indefinite Repetition

- In programs where a count of repetitions is not known (indefinite), we can use one of two repetition control mechanisms:
  - Recursion
  - *While* statement

# Recursion

- **It is not a program statement with a special word that identifies it as part of the programming language**
  - **Recursion means that a method (or a function) calls itself**
- **It is used in situations where the programmer does not know how many times the loop should be repeated**

# Recursion

 **Recursion is your friend!**

 **Recursion is powerful!**

- **Powerful friends can get you what you want simply and efficiently, but they should be treated with care lest they turn on you and get you in a heap of trouble instead.**

# Recursion

- **Recursion is more than just a programming technique. It has two other uses in computer science and software engineering, namely:**
  - **as a way of describing, defining, or specifying things.**
  - **as a way of designing solutions to problems (divide and conquer).**

# Recursion

- **As a descriptive method, and as a problem-solving method, recursion is extremely widely used and it is often the method of choice, unparalleled in its power and clarity.**

# Recursion

- **It is an algorithmic technique where a function, in order to accomplish a task, calls itself with some part of the task**
- **A recursive process is one in which objects are defined in terms of other objects of the same type**
- **Using some sort of recurrence relation, the entire class of objects can then be built up from a few initial values and a small number of rules**

# Everyday Examples

- **Royalty**
  - A person is royal if they are a monarch or are descended from a royal person.
- **Citizenship**
  - A person is a citizen if they were born in the nation, or has acquired citizenship after emigrating to the nation, or has a parent who is a natural citizen

# Mathematical Example

- **The factorial function:**
  - **factorial(0) = 1**
  - **factorial(n) = n \* factorial(n-1) [for n>0]**

# Mathematical Example

- **Let's compute factorial(3):**
  - **factorial(3) = 3 \* factorial(2)**
    - = 3 \* ( 2 \* factorial(1) )**
    - = 3 \* ( 2 \* ( 1 \* factorial(0) ) )**
    - = 3 \* ( 2 \* ( 1 \* 1 ) )**
    - = 6**

# Recursive Definition

- **Base Case(s) + Recursive Case**
- **In general, a recursive definition is made up of two parts:**
  - **There is a base case that tells us directly what the answer is, and**
  - **There is a recursive case that defines the answer in terms of the answer to some other, related problem**

# Recursive Definition

- In factorial, the base case is  $\text{factorial}(0)=1$ , this directly tells us the answer; the recursive case is  $\text{factorial}(n) = n * \text{factorial}(n-1)$

# Recursive Definition

- **This does not directly tell us the answer, but tells how to construct the answer for  $\text{factorial}(n)$  on the basis of the answer to the related problem,  $\text{factorial}(n-1)$**

# Recursive Definition

- **The Royalty and Citizenship definitions are just the same – there is a base case, which directly determines royalty or citizenship, and a recursive case, which defines the royalty/citizenship of one person in terms of the royalty/citizenship of some other person**

# Using Recursion To Solve Problems

- **Recursion is a common form of the general-purpose problem-solving technique called “divide and conquer”**

# Using Recursion To Solve Problems

- **The principle of divide-and-conquer, is that you solve a given problem  $P$  in 3 steps:**
  - **Divide  $P$  into several subproblems,  $P_1, P_2, \dots, P_n$**
  - **Solve, however you can, each of the subproblems to get solutions  $S_1 \dots S_n$**
  - **Use  $S_1 \dots S_n$  to construct a solution to the original problem,  $P$**

# Using Recursion To Solve Problems

- This is often recursive, because in order to solve one or more of the subproblems, you might use this very same strategy
- For instance, you might solve P1 by subdividing it into P1a, P1b..., solving each one of them, and putting together their solutions to get the solution S1 to problem P1

# An Everyday Example

- **To give a simple example, suppose you want to solve this problem:**
  - **P = carry a pile of 50 books from your office to your car**
- **Well, you can't even lift such a big pile, never mind carry it. So you split the original pile into 3 piles:**
  - **P1 contains 10 books**
  - **P2 contains 15 books**
  - **P3 contains 25 books**

# An Everyday Example

- Your plan, of course, is to carry each pile to the car separately, and then put them back together into one big pile once they are all there
- This is a recursive divide-and-conquer strategy; you have divided the original problem  $P$  into 3 problems *of the same kind*

# An Everyday Example

- **Now suppose you can directly solve P1 and P2 - they are small enough that you can carry them to the car**
- **So, P1 and P2 are now sitting out there by the car. Once you get P3 there, you will pile them up again**

# An Everyday Example

- **But P3 is too big to be carried. You apply the same strategy recursively:**
  - **Divide P3 into smaller problems, solve them, and combine the results to make the solution for P3**
  - **You divide P3 into two piles, P3a has 11 books, P3b has 14, and carry these separately to the car**
  - **Then you put P3a back onto P3b to form the pile P3 there at the car**

# An Everyday Example

- **Now you have solved all the subproblems: P1, P2, P3 are sitting at the car. You stack them into one big pile, and presto, you have solved the original problem.**
- **This may seem like a rather hokey example, but it illustrates very well the principles of divide-and-conquer. Let us see how we can use the very same ideas to solve mathematical and computer science problems.**

# Abstraction

- **Abstraction is the process generalization by reducing the information content of a concept or an observable phenomenon, typically in order to retain only information which is relevant for a particular purpose**
- **For example, abstracting a leather soccer ball to a ball retains only the information on general ball attributes and behaviour**
- **Similarly, abstracting happiness to an emotional state reduces the amount of information conveyed about the emotional state.**

# Abstraction & CS

- **In computer science, abstraction is a mechanism and practice to reduce and factor out details so that one can focus on a few concepts at a time.**

# Abstraction & CS

- **The concept is by analogy with abstraction in mathematics**
- **The mathematical technique of abstraction begins with mathematical definitions; this has the fortunate effect of finessing some of the vexing philosophical issues of abstraction**
- **For example, in both computing and in mathematics, numbers are concepts in the programming languages, as founded in mathematics. Implementation details depend on the hardware and software, but this is not a restriction because the computing concept of number is still based on the mathematical concept**

# Control & Data Abstraction

- **Roughly speaking, abstraction can be either that of control or data**

# Control Abstraction

- **Control abstraction is the abstraction of actions while data abstraction is that of data structures**
  - **For example, control abstraction in structured programming is the use of subprograms and formatted control flows**

# Data Abstraction

- **Data abstraction is to allow for handling data bits in meaningful manners**
  - For example, it is the basic motivation behind datatype
- **Object-oriented programming can be seen as an attempt to abstract both data and code**

# Translation

- **Linguistics**
- **Using machines to translate natural languages**