Department of Software Engineering

University of Sialkot

Data Structures

COMPUTER LABORATORY MANUAL

Instructor: Azka Mir

Table of Contents

PREFACE

This lab manual has been prepared to facilitate the students of software engineering in studying and analyzing data structures and algorithms. The aim of this course is to provide an introduction to computer algorithms and data structures, with an emphasis on practical implementations. Data Structures and Algorithms introduces fundamental techniques for problem solving that are relevant to most areas of computer science, both theoretical and applied. Algorithms and data structures for sorting, searching, graph problems, and geometric problems are covered. The lab manual is designed to help students understand the basic concepts of data structures and implement basic computer science algorithms in object oriented approach.

GENERAL INSTRUCTIONS

a. Students are required to maintain the lab manual with them till the end of the semester.
b. All readings, answers to questions and illustrations (code and output) must be solved on the place provided. If more space is required then additional sheets may be attached.
c. It is the responsibility of the student to have the manual graded before deadlines as given by the instructor
d. Loss of manual will result in re submission of the complete manual.
e. Lab session details will be given in training schedule.
f. Keep the manual neat clean and presentable.
g. Plagiarism is strictly forbidden. No credit will be given if a lab session is plagiarized and no re submission will be entertained.
h. Marks will be deducted for late submission.

EXPERIMENT 3 – RECURSION

1. Objectives:

a) Understanding the concepts of recursion and its applications.
b) Implement recursion in two contexts: simple function operations and using linked lists.
c) Demonstrate recursive techniques for traversing and processing data.

2. Time Required: 3 hrs

3. Software Required:

(a). Windows OS (b). Bloodshed Dev C++ (c). Microsoft Visual Studio Code

4. Recursion:

Recursion is a programming technique where a function calls itself to solve smaller instances of a problem. It is characterized by two key elements:

1. **Base Case**: The condition under which the recursion stops.
2. **Recursive Case**: The part where the function calls itself with modified arguments to approach the base case.

5. Recursive function:

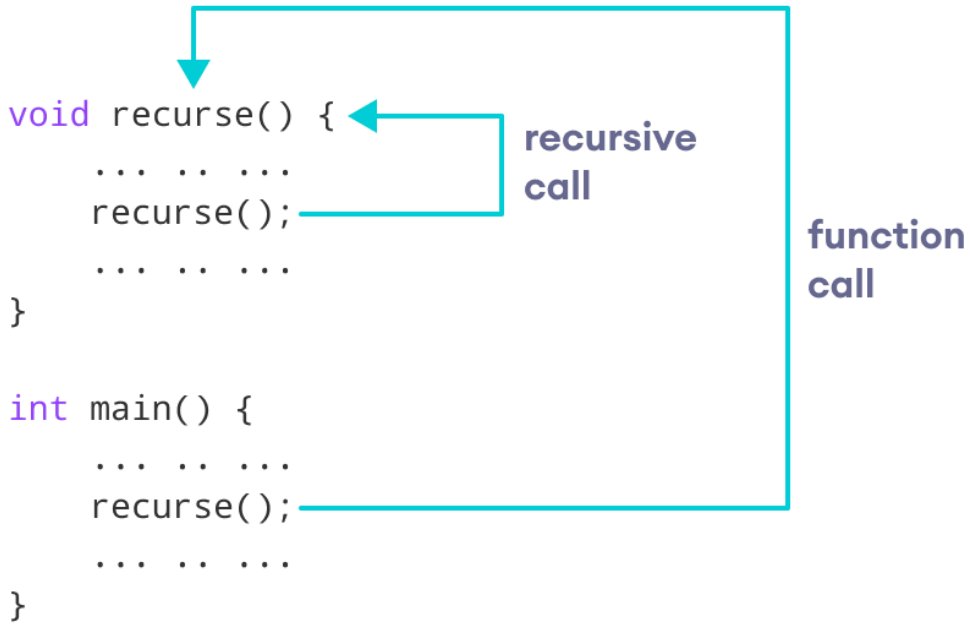


Figure 1: Working of a recursive function

6. Recursive function for factorial:

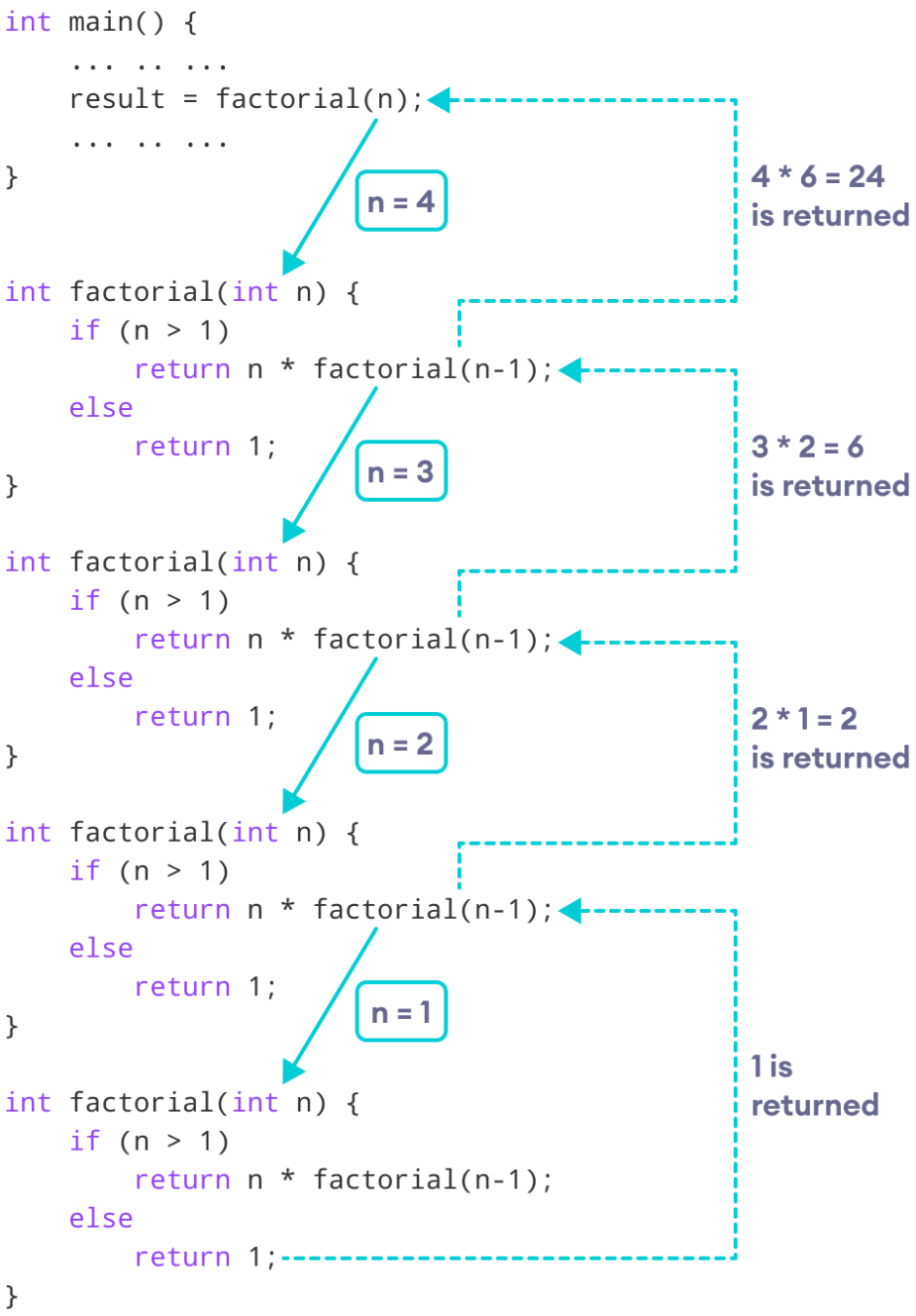


Figure 2: Calculating Factorial using recursion

Key Idea for reverse

Assume you're investigating a house of mirrors and come across a door labeled ***"Exit"***. In recursion, this door represents the ***base case***. It is the point at which the recursive function decides to ***pause*** and begin unwinding the layers of calls it has made.

The ***base case*** is like a safety net, ensuring that the recursion doesn't spiral into an ***infinite loop***. Without it, the function would keep calling itself indefinitely, leading to a ***stack overflow*** error or an ***infinite loop***. Therefore, the base case is essential in any recursive algorithm.

Basic Principle:

Each ***function call*** in a recursive chain creates a new stack frame in memory. This frame stores ***local variables, parameters***, and the ***return address*** as the recursive calls pile up, and a stack-like structure forms, known as the ***call stack***. When a call returns, its stack frame is removed, freeing up memory. The call stack ensures that the program can keep track of where to resume execution after each recursive call and prevents memory overflow. It's a crucial mechanism for managing the recursion process efficiently.

The reverse happens because recursion builds up calls in a stack-like manner. The first elements are processed last because the function unwinds in reverse order of the calls. This naturally creates a reverse traversal.

Exercise 3.1

Create a calculator with the following functions:

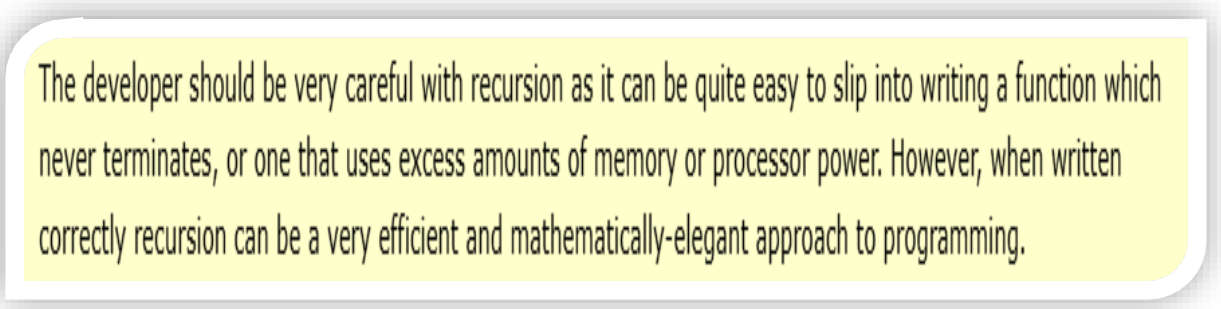
1. **Factorial()**: Calculate factorial of 30 and display it.
2. **Sum()**: Calculate power of a number and display it.
3. **FibonacciSeries()**: Calculate power of a number and display it.
4. **Power()**: Calculate power of a number and display it.

Note: A menu should be created and input should be taken by the user.

Exercise 3.2

(a) Implement a singly linked list of Songs Playlist with dynamic memory allocation.
(b) Write a function to recursively traverse the linked list in forward order.
(c) Write another function to recursively traverse the linked list in reverse order.

Tid-bits:
In object-oriented programming, recursion can be used to traverse complex object structures like **trees** and **graphs**. Recursive methods can navigate through object relationships and perform operations at each level.



Summary:
This lab session introduces the concepts of recursion. It also helps students in implementing recursive functions.