Chapter 16

Recursive Functions

16.1 Recursive Functions
   16.1.1 Iterative versus Recursive
   16.1.2 Comparing Iterative and Recursive Processes

16.2 Further Examples with Recursion
   16.2.1 String Reversion
   16.2.2 Recursion over Arrays

16.3 The Towers of Hanoi
   16.3.1 Problem Definition
   16.3.2 Problem Definition
   16.3.3 Ideas for a Recursive Solution
   16.3.4 A Recursive Tower-of-Hanoi Algorithm
16.4 References
16.1 Recursive Functions

16.1.1 Iterative versus Recursive

Definitions of the Factorial Function

\[ \text{factorial}(n) := n! \]
\[ := n \times (n-1) \times \ldots \times 2 \times 1 \]
\[ := \prod_{i=1}^{n} i \]
Iterative Implementation

\[
\text{factorial}(n) := \prod_{i=1}^{n} i
\]

Here is a typical iterative C++ implementation of the factorial function:

```cpp
int factorial_iter (int n)
{
    int i, f=1;
    for(i=1; i <= n; i++) f *= i;
    return f;
}
```
Towards a Recursive Definition

\[
\text{factorial}(n) := \prod_{i=1}^{n} i = n \times (n-1) \times \ldots \times 2 \times 1
\]

\[
\text{factorial}(n-1) := \prod_{i=1}^{n-1} i = (n-1) \times (n-2) \times \ldots \times 2 \times 1
\]

\[
\text{factorial}(n-2) := \prod_{i=1}^{n-2} i = (n-2) \times \ldots \times 2 \times 1
\]

\[
\ldots
\]

\[
\text{factorial}(2) := \prod_{i=1}^{2} i = 2 \times 1
\]
A Recursive Definition

\[
\text{factorial}(n) := \begin{cases} 
1 & \text{if } n = 1 \\
 n \cdot \text{factorial}(n - 1) & \text{if } n > 1 
\end{cases}
\]

Recursive Implementation of the Factorial Function (version 1)

```c
int factorial_rec (int n)
{
    if(n == 1) return 1;
    return n * factorial_rec(n-1);
}
```
16.1.2 Comparing Iterative and Recursive Processes

Iterative Version

```c
int factorial_iter (int n)
{
    int i, f=1;
    for(i=1; i <= n; i++) f *= i;
    return f;
}
```

Executing this program generates a linear iterative process:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>i</th>
<th>f</th>
<th>n</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>24</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Recursive Version

```c
int factorial_rec(int n)
{
    (n == 1) ? 1 : (n * factorial_rec(n-1));
}
```

This generates a linear recursive process, as the following example shows:

```
fac_rec(5)
(5 * fac_rec(4))
(5 * (4 * fac_rec(3)))
(5 * (4 * (3 * fac_rec(2))))
(5 * (4 * (3 * (2 * fac_rec(1)))))
(5 * (4 * (3 * (2 * 1))))
(5 * (4 * (3 * 2)))
(5 * (4 * 6))
(5 * 24)
(120)
```
16.2 Further Examples with Recursion

16.2.1 String Reversion

The following example uses a recursive function to print a string backwards.

```c
void reverse(char *s)
{
    if(*s)
        reverse(s+1)
    else
        return;
}

void main()
{
    cout << reverse("Left to right");
}
```
16.2.2 Recursion over Arrays

A function for adding elements \( m \) through \( n \) of an array, can be defined as follows:

- If there is only one element, the sum is the value of this element.
- Otherwise, the sum is calculated by adding the first element and the sum of the rest.

Here is the C++ implementation:

```cpp
int sum(int first, int last, int array[]) {
    if(first == last)
        return array[first];
    /* else */
    return
        (array[first] +
        sum(first+1,last,array));
}
```
Here is another example of the recursive process generated by this function:

\[ \text{Sum}(1 \ 8 \ 3 \ 2) = \]

\[ (1 + \text{Sum}(8 \ 3 \ 2)) = \]

\[ (1 + (8 + \text{Sum}(3 \ 2))) = \]

\[ (1 + (8 + (3 + \text{Sum}(2)))) = \]

\[ (1 + (8 + (3 + 2))) = \]

\[ (1 + (8 + 6)) = \]

\[ (1 + 14) = \]

\[ (15) \]
16.3 The Towers of Hanoi

16.3.1 Problem Definition
16.3 The Towers of Hanoi

16.3.1 Problem Definition
Step 0 — Initial

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 1 — Recursion 1

A

B

C

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 2 — Recursion 1

Initial Setup:
Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:
Move all $n$ disks from tower A to tower B.

Moves:
Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 3 — Recursion 1

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 4 — Recursion 1

Initial Setup:

Tower A contains $n$ disks of different sizes.
Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower.
No disk may be put on top of a smaller disk.
Step 5 — Recursion 1

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 6 — Recursion 1

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 7 — Recursion 1

A BC

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 8 — Move

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 9 — Recursion 2

Initial Setup:

Tower A contains \( n \) disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all \( n \) disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 10 — Recursion 2

Initial Setup:

Tower A contains \( n \) disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all \( n \) disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 11 — Recursion 2

Initial Setup:
Tower A contains \( n \) disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:
Move all \( n \) disks from tower A to tower B.

Moves:
Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 12 — Recursion 2

**Initial Setup:**
Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

**Objective:**
Move all $n$ disks from tower A to tower B.

**Moves:**
Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 13 — Recursion 2

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 14 — Recursion 2

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
Step 15 — Recursion 2

Initial Setup:

Tower A contains $n$ disks of different sizes. Disks can only go on top of smaller disks (or directly on the board).

Objective:

Move all $n$ disks from tower A to tower B.

Moves:

Take the top disk of a tower and move the disk to another tower. No disk may be put on top of a smaller disk.
16.3.2 Ideas for a Recursive Solution

... is equivalent to ...

move tower
with \( n \) disks

move tower
with \( n-1 \) disks

move
disk

move tower
with \( n-1 \) disks

move tower
with \( n \) disks
16.3.3 A Recursive Tower-of-Hanoi Algorithm

The Main Algorithm:

```cpp
int main()
{
    int n;

    cout << "How many disks? ";
cin >> n;

    if( n > 0)
        moveTower(n, 'A', 'B', 'C');

    return 0;
}
```

Exponential problem: \( n \) disks \( \rightarrow 2^n - 1 \) moves
Algorithm output:

How many disks? 4

Move top disk from tower A to tower C.
Move top disk from tower A to tower B.
Move top disk from tower C to tower B.
Move top disk from tower A to tower C.
Move top disk from tower B to tower A.
Move top disk from tower B to tower C.
Move top disk from tower A to tower C.
Move top disk from tower A to tower B.
Move top disk from tower C to tower B.
Move top disk from tower C to tower A.
Move top disk from tower C to tower A.
Move top disk from tower B to tower A.
Move top disk from tower C to tower B.
Move top disk from tower A to tower C.
Move top disk from tower A to tower B.
Move top disk from tower C to tower B.
void moveTower(int n, char from, char to, char temp)
{
    if (n==1)
        moveDisk(from, to);
    else {
        moveTower(n-1, from, temp, to);
        moveDisk(from, to);
        moveTower(n-1, temp, to, from);
    }
}

void moveDisk(char from, char to)
{
    cout << "Move top disk from tower " << from << endl;
    cout << " to tower " << to << "." << endl;
}
16.4 References
