Department of Information Systems and Technologies

CTIS152 – Algorithms and Data Structures

Spring 2024 - 2025

Lab Guide #13 – Week 8 – 1

OBJECTIVE	: Recursive Functions
Instructor **Assistants**	: Serpil TIN : Berk ÖNDER & Hatice Zehra YILMAZ

Q1. Write a C program that reads positive numbers from the user into an integer array with a maximum size of **30**, and checks if the array is sorted in ascending order.

Write the following recursive function;

- **`isSorted:`** checks if the array is sorted in ascending order and returns 1 if it is sorted otherwise returns 0.

Project Name: LG13_Q1
File Name: Q1.cpp

Example Run#1:
```
Enter a positive number: 10
Enter a positive number: 20
Enter a positive number: 30
Enter a positive number: 40
Enter a positive number: 50
Enter a positive number: -1
The array is sorted in ascending order.
```

Example Run#2:
```
Enter a positive number: 12
Enter a positive number: 25
Enter a positive number: 34
Enter a positive number: 17
Enter a positive number: 49
Enter a positive number: 50
Enter a positive number: -1
The array is NOT sorted in ascending order.
```

Q2. A perfect number is a number in which the sum of its divisors equals the number itself. For example, 6 is a perfect number since the sum of its divisors 3, 2, and 1 adds up to 6.

Write a C program that reads a non-negative number from the user, decides whether the number is perfect or not, and displays a message as in the example run.

Write the following recursive function;

- **`sumDivisors:`** finds and returns the sum of the divisors of an integer number.

HINT: The maximum divisor of a number may be half of it.

Project Name: LG13_Q2
File Name: Q2.cpp

Example Run#1:
```
Enter a non-negative number: -5
Enter a non-negative number: 0
Enter a non-negative number: 6

6 is a perfect number!
```

Example Run#2:
```
Enter a non-negative number: -9
Enter a non-negative number: -1
Enter a non-negative number: -3
Enter a non-negative number: 0
Enter a non-negative number: 18

18 is NOT a perfect number!
```

Q3. Write a C program that gets a string from the user and checks whether it is a palindrome or not. (A **palindrome** is a word, phrase, number, or sequence of words that reads the same backward as forward)

Write the following recursive function;

- **`isPalindrome:`** finds whether a string is a palindrome or not; the function returns 1 if the string is a palindrome and 0 if otherwise. The function should have no other parameters other than the string.

Project Name: LG13_Q3
File Name: Q3.cpp

Example Run #1:

```
Enter a word: redivider

The word <redivider> is a palindrome.
```

Example Run #2:

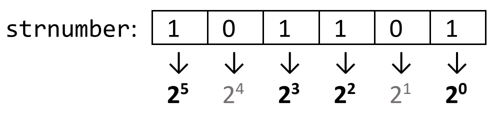
```
Enter a word: algorithm

The word <algorithm> is NOT a palindrome.
```

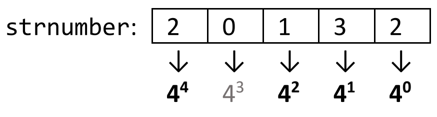
Q4. Write a C program that reads a number into a string and the base of the number. First, it will validate the base number, then it will find the number's decimal equivalent by using recursive functions.

Write two recursive functions;

- **`validateBase`** that takes a number as a string and its base as parameters. Then, it checks if the number is on the given base recursively. If the number is on that base then it returns 1, otherwise returns -1;

- **`convertDecimal`** that takes a number as a string and its base as parameters. Then, it finds and returns the decimal equivalent of the number recursively as shown in the example right below:





For base 2 the result will be 32+8+4+1=45

For base 4 the result will be 512+16+12+2 = 542

HINT: You can use the built-in function **`powf()`** by including the math header file.

Project_name: LG13_Q4
File_name: Q4.cpp

Example Run #1:

```
Enter a number: 56845
Enter the base of the number: 5
Enter the base of the number: 7
Enter the base of the number: 8
Enter the base of the number: 9
The decimal is: 37868
```

Example Run #2:

```
Enter a number: 1345
Enter the base of the number: 2
Enter the base of the number: 7
The decimal is: 523
```

Additional Questions

AQ1.

Some algorithms require nested recursion where the result of one function call is a parameter to another function call. For Example; finding the Catalan Number on that nth position. The formula is given as below;

$$C(0) = 1 \quad and \quad C(n) = \sum_{i=0}^{n-1} C(i)C(n-i-1) \qquad for \quad n \geq 1$$

The few Catalan numbers for every n = 0, 1, 2, 3, 4 … are 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, …

Write a C program that gets one integer number from the user and computes the result of C(n). The value of n has got to be non-negative values {n} >= 0.

Write a recursive function;

- **catalan:** takes an integer number as a parameter, finds and returns the ***nth*** Catalan number.

Project_name: LG13_AQ1
File_name: AQ1.cpp

Example Run #1:
```
Enter a number: 6
Catalan equivalent is: 132
```

Example Run #2:
```
Enter a number: 18
Catalan equivalent is: 477638700
```

Example Run #3:
```
Enter a number: 0
Catalan equivalent is: 1
```

AQ2.

In a supermarket there is a campaign for chocolates, you can get extra chocolates by returning the wrapper of the chocolates.

Write a C program that gets the **money** amount, the **unit price** of chocolate, the **number of wrappers** to be returned for getting one extra chocolate, and finds the total number of maximum chocolates you can eat. (It may be assumed that all the given values are positive integers and greater than 1.)

Write the following two functions;

- **countRec:** gets the number of chocolates and the number of wrappers required to get one extra chocolate as input parameters, **recursively** finds and returns the number of chocolates we can have. (hint: count the number of chocolates by returning wrappers until the wrappers left didn't become less than required to get a chocolate)
- **countMaxChoco:** gets the money amount, the unit price of chocolate, and the number of wrappers required to get a chocolate. The function finds and returns the maximum number of chocolates we can eat using the recursive function **countRec.**

Project Name: LG13_AQ2
File Name: AQ2.cpp

Example Run #1:
```
How much money do you have: 15
Enter the unit Price for chocolate: 1
Enter # of wrapper(s) for getting one extra chocolate: 3

You can buy 15 chocolates

You can return 15 wrappers back and get 5 more chocolates
You can return 5 wrappers back and get 1 more chocolates
You can return 3 wrappers back and get 1 more chocolates

Finally the total # of chocolates is 22
```

Example Run #2:
```
How much money do you have: 16
Enter the unit Price for chocolate: 2
Enter # of wrapper(s) for getting one extra chocolate: 2

You can buy 8 chocolates

You can return 8 wrappers back and get 4 more chocolates
You can return 4 wrappers back and get 2 more chocolates
You can return 2 wrappers back and get 1 more chocolates

Finally the total # of chocolates is 15
```