Laboratory 08 – Recursive Functions

30.11.2021

1. [2points] A recursive function that will print an array.

a) Define a recursive function that will print an array from 0 to the last element.

b) Define a recursive function that will print an array from the last element to 0.

In main function:

c) Declare an array of `10` integers. Use the preprocessor `#define` directive.
d) Fill the array with pseudo-random numbers from `0` to `100`.
e) Call recursive functions that will print the array.

Test data:

```
arr[0] = 178
arr[1] = 78
arr[2] = 14
arr[3] = 21
arr[4] = 116
arr[5] = 166
arr[6] = 113
arr[7] = 119
arr[8] = 143
arr[9] = 149

arr[9] = 149
arr[8] = 143
arr[7] = 119
arr[6] = 113
arr[5] = 166
arr[4] = 116
arr[3] = 21
arr[2] = 14
arr[1] = 78
arr[0] = 178
```

2. [3points] A recursive function that returns the sum of the elements stored in the array.

a) Define a recursive function that computes the sum of all array elements.

The function must sum the elements of the array without using a loop.

We use recursion instead of loops.

A recursive function must return the sum of the elements.

The function must determine the start value of the summation.

In main function:

b) Declare an array of `10` integers. Use the preprocessor `#define` directive.
c) Fill the array with pseudo-random numbers from `0` to `100`.

d) Call the recursive function and print the summation result.

Test data:

```
arr[0]=113
arr[1]=131
arr[2]=127
arr[3]=91
arr[4]=30
arr[5]=103
arr[6]=140
arr[7]=126
arr[8]=140
arr[9]=51
```

```
sum = 1052
```

3. [2points] Write two functions calculating the `n`th term of the Fibonacci sequence, one recursive and the other iterative.

In mathematics, the Fibonacci numbers (`Fn`), form a sequence, called the Fibonacci sequence, such that each number is the sum of the two preceding ones, starting from `0` and `1`. That is,

$$F_n := \begin{cases} 0 & \text{dla } n = 0, \\ 1 & \text{dla } n = 1, \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

The beginning of the sequence is thus:

`0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987,…`

a) Use the mathematical definition of the Fibonacci sequence to write a recursive function that computes the `n-th` term of the Fibonacci sequence. The function only uses one variable `n`.

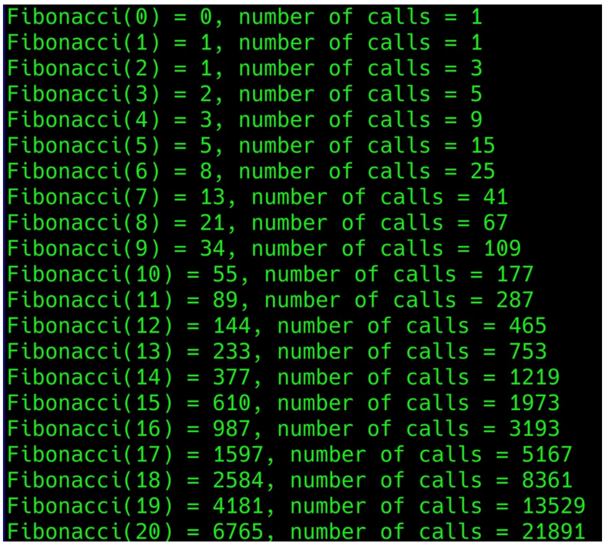
b) Write a non-recursive function that computes the `n-th` term of the Fibonacci sequence. Use a `for` loop. Don't use array.

Use `long int` instead of `int`.

Test data:

```
Fibonacci (28) = 317811
Fibonacci (28) = 317811
```

`4.[6points]` Let's count how many times the recursive function that computes the `n-th` term of the Fibonacci sequence calls itself.



```
Fibonacci(0) = 0, number of calls = 1
Fibonacci(1) = 1, number of calls = 1
Fibonacci(2) = 1, number of calls = 3
Fibonacci(3) = 2, number of calls = 5
Fibonacci(4) = 3, number of calls = 9
Fibonacci(5) = 5, number of calls = 15
Fibonacci(6) = 8, number of calls = 25
Fibonacci(7) = 13, number of calls = 41
Fibonacci(8) = 21, number of calls = 67
Fibonacci(9) = 34, number of calls = 109
Fibonacci(10) = 55, number of calls = 177
Fibonacci(11) = 89, number of calls = 287
Fibonacci(12) = 144, number of calls = 465
Fibonacci(13) = 233, number of calls = 753
Fibonacci(14) = 377, number of calls = 1219
Fibonacci(15) = 610, number of calls = 1973
Fibonacci(16) = 987, number of calls = 3193
Fibonacci(17) = 1597, number of calls = 5167
Fibonacci(18) = 2584, number of calls = 8361
Fibonacci(19) = 4181, number of calls = 13529
Fibonacci(20) = 6765, number of calls = 21891
```

a) In the `main` function, declare a variable that will count recursive Fibonacci function calls. Pass the address of this variable to the recursive Fibonacci function. The recursive Fibonacci function now has two parameters, the first is `long int` and the second is `long int *`.

b) In the recursive Fibonacci function, increase the value (`++`) of the variable passed to the function using a pointer.

c) In the `main` function, call the recursive Fibonacci function.

d) Make a copy of the recursive Fibonacci function. In the function declare an array of size `21`, its first two elements are `0` and `1`, the rest of the elements are zero. Use the `static` keyword when declaring the array.

e) Modify the recursive Fibonacci function.

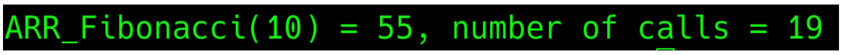
If `n > 1`.

> If the value in the array is not zero then return the value. Otherwise, calculate the of value the `n-th` term of the Fibonacci sequence by calling the recursive Fibonacci function. As soon as the value is calculated, assign it to the appropriate cell in the array.

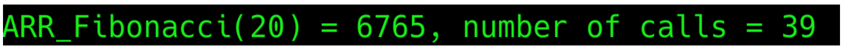
If `n < 2`, then return `n`.

f) The number of recursive calls should be reduced to those necessary for the correct calculation of the function value. In the `main` function, call the modified recursive Fibonacci function.

Test data:



```
ARR_Fibonacci(10) = 55, number of calls = 19
```



```
ARR_Fibonacci(20) = 6765, number of calls = 39
```

5.[4points] Write two functions calculating the greatest common divisor (`gcd`) of two integers, one recursive and the other iterative.

In mathematics, the greatest common divisor (gcd) of two integers, which are not all zero, is the largest positive integer that divides each of the integers. For example, the gcd of 8 and 12 is 4

Euclid's algorithm, is an efficient method for computing the greatest common divisor (gcd) of two integers, the largest number that divides them both without a remainder.

Formally, the algorithm can be described as:

```
gcd(a,b) = a, if b=0

gcd(a,b) = gcd(b, a % b) if b>0
```

a) Use the mathematical definition of gcd(a,b) to write a recursive function that computes gcd(a,b). Add return where appropriate. We do not declare any new variables in the function, we only use a and b.

b) Write a non-recursive function that computes gcd(a,b). Use a while loop. Don't use array. Notice how the values in the table change. The following lines in the table correspond to the iterations of the loop.

gcd(646, 360) = 2

a	b	c = a%b
646	360	286
360	286	74
286	74	64
74	64	10
64	10	4
10	4	2
4	2	0

c) In the main function, print the greatest common divisor (gcd) of (646,360).

Test data:

```
rgcd(646, 360) = 2
igcd(646, 360) = 2
```

6.[3points] Given a sorted array of n elements, write a recursive function to search a given element xyz in the array.

Binary Search:

a) Search a sorted array by repeatedly dividing the search interval in half.

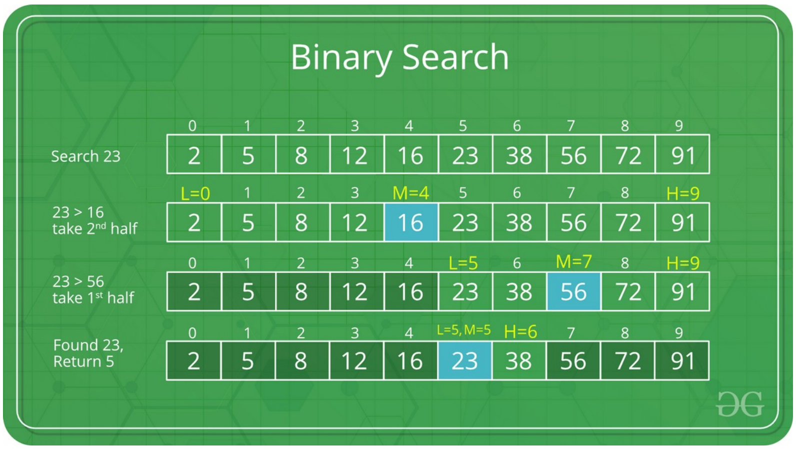
b) Begin with an interval covering the whole array.

c) If the value of the search key (`xyz`)is less than the item in the middle of the interval, narrow the interval to the lower half.

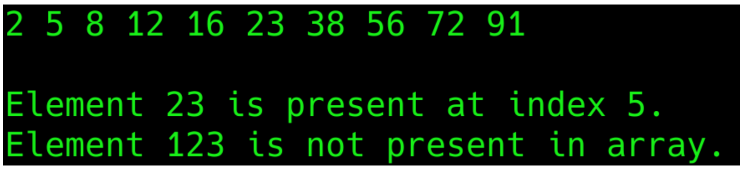
d) Otherwise, narrow it to the upper half.

e) Repeatedly check until the value is found or the interval is empty.

f) If the value is not found or the interval is empty `return -1`.



Test data:



```
2 5 8 12 16 23 38 56 72 91

Element 23 is present at index 5.
Element 123 is not present in array.
```

Next lab 9 – Pointers.