Week 9 Programming Assignment

Steffen Petersen — au722120

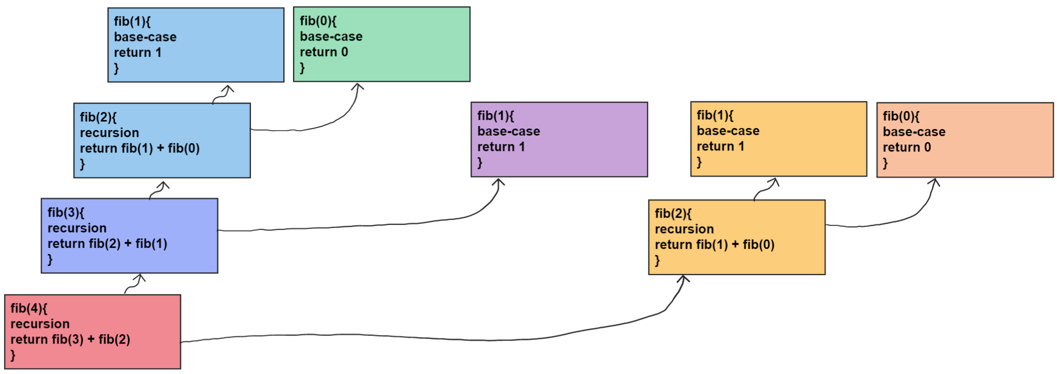
November 7th 2022

Here is the link for my repository, in which you will find all the edited code files and such.
https://github.com/Aarhus-University-ECE/assignment-9-SirQuacc

1

> (Text) We talked about the run-time stack (see e.g. slides from lecture 7). In the lecture, we looked at the Fibonacci numbers and a program to calculate them (`fib.c`). Draw the stach as it evolves when calculating `fib(4)`

In the image below I've drawn a schematic of how the runtime stack evolves as the function calls upon itself. It will start returning values to the previous functions once it reaches any base case, and a recursion branch ends. Each box represents a new function call, i.e. a new stack in the runtime. The timeline runs left to right.



2

(Code) Summing an array kan recursively be described as follows (a is the array, n is the length of the array):

$$sum(a, n) = \begin{cases} a[n-1] + sum(a, n-1), & \text{if } n > 0 \\ 0, & \text{if } n = 0 \end{cases}$$

Implement a recursive function with the signature `int sum(int a[], int n)` that sums the integer array a

Below is the recursive function, it can also be found in sum.c

```
int sum(int a[], int n)
{
    assert(!(n < 0)); // Can't search an array of lower than 0 length
    if(n == 0){
        return 0; //Base case, we're at the end of he array, return 0 as the "sum" of nothing
    }
    else return a[n-1] + sum(a, n-1); //Recursively ask for the sum of the next number, and
        add it to the current
}
```

3

(Code) In the lecture, we looked at an recursive binary search. To use binary search, the elements must be sorted. A recursive search function NOT requiring the elements to be sorted could look like (a is the array, n is the length of the array, x is the element to be found):

$$search(a, n, x) = \begin{cases} true, & \text{if } n > 0 \text{ } and \text{ } a[n-1] == x \\ search(a, n-1, x), & \text{if } n > 0 \text{ } and \text{ } a[n-1]! = x \\ false, & \text{if } n = 0 \end{cases}$$

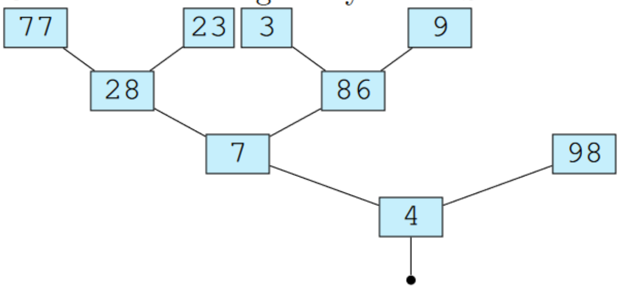
Implement a recursive function with the signature:
`bool search(int a[], int n, int x)` that searches the integer array a for the element x.

The code is seen below and can be found in search.c

```
bool search(int a[], int n, int x)
{
    assert(!(n < 0)); // Can't search an array of lower than 0 length
    if(n == 0){ //Base case, if we're beyond the array, element x wasn't in there,
        return false;
    }
    if(x == a[n-1]){ //Recusively check with linear search if x is equal to any element in
        the array of length n.
        return true;
    } else rethurn search(a, n-1, x);
}
```

4

(Code) Implement depth-first search using a stack in a fashion similar to as presented in the lectures. Your stack should be implemented as a linked list, and your tree as *tree nodes* that each have an integer as the data item and a left and right child. Given the following tree your DFS code should print the sequence of nodes visited.



The correct output should be: 4, 7, 28, 77, 23, 86, 3, 9, 98

Below is my code for this function, it can also be found in dfs.c

```c
void DFT (node * root)
{
  printf("The given tree:\n");
  print_tree(root, 0); //Print the given tree first

  stack* mainStack = malloc(sizeof(stack)); //Allocate a stack node
  initStack(mainStack); //Initialize the stack
  mainStack = push(mainStack, root); //Push the root on to the stack first
  stack* popped; //Pointer to the saved node after popping

  printf("Order of visiting tree: ");
  while(!isEmpty(mainStack)){ //!isEmpty(mainStack)
    popped = pop(&mainStack); //Pop the top node, popped variable saves pointer to the popped node
    print_node(popped->node); //Print the visited (popped) node's value.
    if(popped->node->rchild != NULL) mainStack = push(mainStack, popped->node->rchild); //If there is a right child, add this to the stack
    if(popped->node->lchild != NULL) mainStack = push(mainStack, popped->node->lchild); //If there is a left child, add this to the stack
    free(popped); // Free stack-node, clean-up.
  }
  printf("\n");
}
```