French Azerbaijani University
Arash Habibi
Second year
2017-2018

Advanced C Programming

Exercice session 2 : Recursion

1 Scope

Consider the following program. As is, this program compiles and runs well. Its output is :

```
6x7 = 42
```

1. Would it still work if we defined `times3` before `times2` ?
2. Would it still work if we defined `times6` before `times2` ?
3. Would it still work if we defined `main` before `times6` ?
4. Is it possible to make the program work whatever the order in which the functions were defined ?

```
#include <stdio.h>

int times2(int n)
{         return(2*n);   }

int times3(int n)
{         return(3*n);   }

int times6(int n)
{         return(times2(times3(n)));   }

int main()
{
         int n=7;
         printf("6x%d=%d\n",n,times6(n));
         return 0;
}
```

2 Stopping conditions

For the following recursive functions, determine after how many functions calls the recursion terminates.

1. Let function `f` be defined in the following way. If we call `f(3)`, after how many calls (including the first one) does the recursion terminate ? Same question for `f(7)` and `f(-1)`.

```
int f(int n)
{
         if(n==0) return(1);
         else     return(n*f(n-1));
}
```

2. Let function `g` be defined in the following way. If we call `g(4)`, after how many calls (including the first one) does the recursion terminate ? Same question for `g(3)`.

```
int g(int n)
{
         if(n==0) return(1);
         else     return(g(n-2));
}
```

3. Let function `h` be defined in the following way. If we call `h(53)`, after how many calls (including the first one) does the recursion terminate ? Same question for `h(1534)` and `h(-50)`.

```
int h(int n)
{
        if(n<10) return(1);
        else     return(10*h(n/10));
}
```

4. In the previous question, when the recursions do terminate, what is the output of the function ?

3 Recursive algorithms

In this section, our aim is to write function which perform the same operations as in the previous exercice session but with recursive algorithms. For example, in order to calculate the factorial sequence, the *base case* is $n = 0$ (because $0! = 1$) and the *inductive step* is : $n! = n \times (n-1)!$. Here is a function that implements this recursive algorithm :

```
int factorial_r(int n)
{
  if(n=0) then return 1;
  else return(n * factorial_r(n-1));
}
```

You may need two functions written in the previous exercice session : `units` and `allButUnits`.

— Write in C a function with prototype `int units(int n);` which returns the units digit of `n`. For example `units(27453)` returns 3 ;

— Write in C a function with prototype `int allButUnits(int n);` which returns `n` without the units digit. For example `units(27453)` returns 7453.

3.1 The *base case* and the *inductive step* are supplied

1. Write in C a recursive function with prototype : `int square_r(int n);` and which returns n^2 by using only additions, substractions and multiplications by 2.
   The *base cases* are $u_0 = 0$ and $u_1 = 1$ the *inductive step* is : $u_n = 2u_{n-1} - u_{n-2} + 2$
2. Write in C a recursive function with prototype `int magnitude_r(int n)` and which does the same thing as function `magnitude_i`, but with a recursive algorithm. *Hint* : the *base case* is when n is between 0 and 9. And for the *inductive step*, don't try to see the relation between `magnitude_r(n)` and `mangitude_r(n-1)` but between `magnitude_r(n)` and `magnitude_r(allButUnits(n))` (in other words `magnitude_r(n/10)`).
3. Before you begin working on this question, make sure you have finished the previous question. Write in C a non-recursive function with prototype `int mostSignificantBit(int n);` which returns the most significant digit of integer n. For example `mostSignificantDigit(374527)` should return 3.
4. For this question as well, you will need the `magnitude_r` function. Write in C a non-recursive function with prototype `int middle(int n);` which returns a number composed of the middle digits of `n`. For example `middle(43216)` returns `321`.
5. For this question you will need the two previous functions. A palindrome number is a number which is equal to its reverse. For example 17971 is a palindrome, so is 121 and 987656789 and 5. Write in C a recursive function with prototype `int palindrome(int n);` which returns 1 if n is a palindrome and 0 if it is not. the base case is when `n<10` (one digit numbers are always palindromes). *inductive hypothesis :* let us suppose that we

not if `middle(n)` is a palindrome or not. If it is not, then `n` is not a palindrome either. If it is, you still have to check whether the most significant digit equals the units digit.

6. For this question, make sure you have answered the previous question. Write in C a recursive function with prototype `int reverse_r(int n)` and which is the recursive equivalent of `reverse_i`.*Hint* : the *base case* is when n is between 0 and 9. And for the *inductive step*, don't try to see the relation between `reverse_r(n)` and `reverse_r(n-1)` but between `reverse_r(n)` and `reverse_r(allButUnits(n))`.
7. In all these functions, what would happen for $n < 0$?
8. Modify one of the previous functions so that it prints "Hello world" only when the function is called for the first time. (Think about static variables).

3.2 Find the *base case* and the *inductive step* before implementing

For the following questions, you may write the *base case* and the *inductive step* in a comment before the function itself.

1. Write in C a recursive function with prototype : `float power_r(int n, float x);` and which returns x^n by using only multiplications.
2. Write in C a recursive function with prototype : `int sumOfOdd_r(int n);` and which returns the sum of the first n odd numbers. For example `sumOfOdd_r(5)` should be 1 + 3 + 5 + 7 + 9 = 25.
3. Write in C a recursive function with prototype : `float sumOfInverse_r(int n);` and which returns the sum of the inverses of the n first integers. For example `sumOfInverse_r(5)` should be 1 + 1/2 + 1/3 + 1/4 + 1/5 = 2.283333.
4. Difficult question. Write in C a function with prototype `int howManyTimesSubstract_r(int a, int b);` which substracts `b` from `a` (operates `a-b`), then subtracts `b` from the result of this substraction and so on. It returns the number of substractions performed before the result is smaller than `b`. For example for `howManyTimesSubstract_i` is already smaller than `b`. *Inductive hypothesis :* Let us suppose that we know how many times we can substract `b` from `a-b`. If we know that, how many times can we subsract `b` from `a`.
5. Write in C a recursive function with prototype : `int divideBy7(int n);` and which returns the quotient of the division of n by 7. You cannot use the division operator. For example `divideBy7(50)` returns 7. *Hint* : the *base case* is when $n < 7$ and the *inductive step* is not between n and $n - 1$.

4 Dichotomy search of $f(x) = 0$

We suppose that function f is continuous and decreasing over $[a, b]$ with $f(a) > 0$ and $f(b) < 0$. This implies that

$$(\exists x_0 \in [a, b]) \quad f(x_0) = 0$$

We wish to find an approximation of x_0 with a precision of ϵ. In other words, we wish to find x such that $|x - x_0| < \epsilon$.

Here is the algorithm, also described on figure 1. We know that $f(a) > 0$ and $f(b) < 0$. Let us calculate the value of f at the middle m of the $[a, b]$ interval (m is calculated by $m = (a + b)/2$).

— if $f(m) > 0$ then it means that x_0 is not in $[a, m]$ but in $[m, b]$;

— if $f(m) < 0$ then it means that x_0 is not in $[m, b]$ but in $[a, m]$;

Now, we can restart the whole operation, but instead of $[a, b]$ we use this new smaller interval. The algorithm stops when the size of the interval is smaller than ϵ.

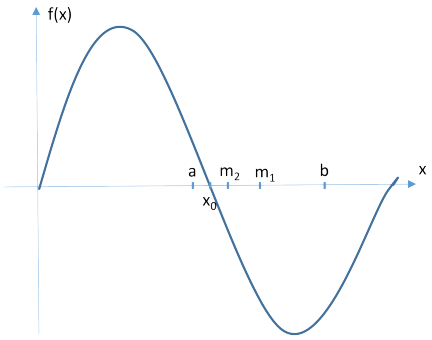

FIGURE 1 – We search for the solution x_0 of equation $f(x) = 0$ in the $[a, b]$ interval. Let $m_1 = (a + b)/2$. We see that $f(m_1) < 0$. This means that x_0 is not in $[m_1, b]$, but in $[a, m_1]$. So we start everything again, but instead of the $[a, b]$ interval, we take $[a, m_1]$.

Express the *base case*, the *inductive step* and write a recursive function with prototype

```
float dicho(float a, float b);
```

We suppose that $f(x)$ is $sin(x)$ and $\epsilon = 0.000001$. Initially, take $[a, b] = [3, 4]$. The result should be π=3.141592....