Tutorial 7- Solution

Objectives: To practice with
- Recursive functions
- User Defined Data Types

1. **Consider the following recursive function that calculates x^y:**

where `x = base;` `y = exponent`

```
int power(int base, int  exponent)
{
      if (exponent == 0)
            return 1;
      else
      return base * power(base, exponent-1);
}
```

a. Where is the Base Case?

b. Where is the General Case?

c. What is the returned value of this function call: `result = power( 2, 3);`

d. Draw a diagram explaining all stages that follow this function call.

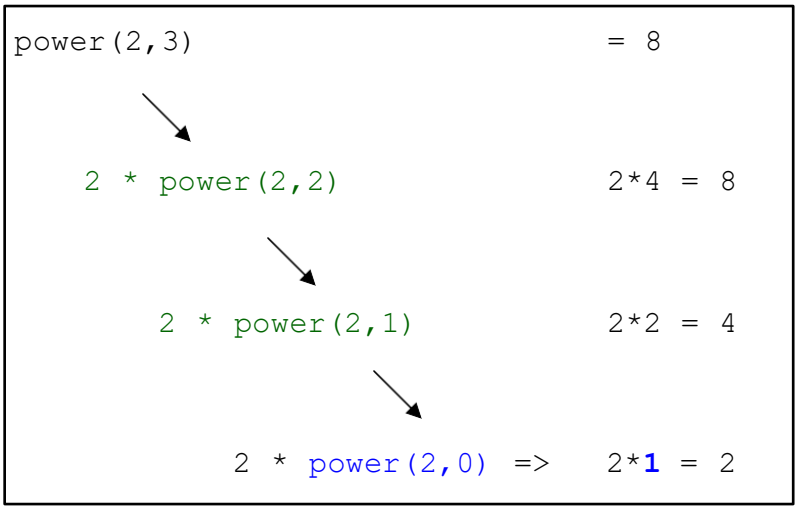
Answer:

a. Base Case: `if (exponent == 0)`

b. General Case: `base * power(base, exponent-1);`

c. result = 8.

d. Diagram is below.

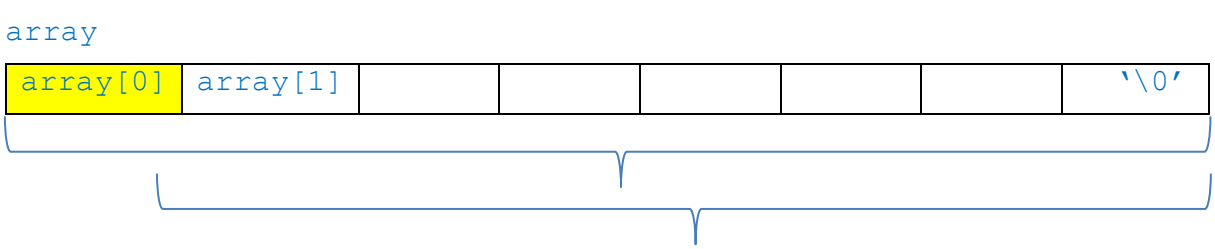


2. Write a recursive function count_digits that counts all the digits in a string.

Answer:

A C-string ends with '\0'

In the answer below, we use the built-in function isdigit() from the ctype.h related library, see Lecture5b_CStrings.pptx. Otherwise, you could check that the character is between '0' and '9'



Recursive expression:

count_digits(array[])=count_digits(&array[1])+(int) isdigit(array[0])

At each recursive call, we reduce the size of the inspected array, until its length becomes 1, then its content is '\0', this corresponds to our stopping condition.

```
 /*  Counts the number of digits in the string str. */
#include<ctype.h>
int count_digits(const char str[])
{
      int ans;

      if (str[0] == '\0')                  /*  base case */
        ans = 0;
      else                  /*redefine problem using recursion */
        if (isdigit(str[0]))/*first character must be counted*/
          ans = 1 + count_digits(&str[1]);
        else              /*  first character is not counted  */
          ans = count_digits(&str[1]);

      return (ans);
}
Function call: number = count_digits(input);
     Assuming: #define SIZE 20
               char input[SIZE];
```

3. What is the output of the following program? What does function `strange` compute when called with a positive integer?

```
#include <stdio.h>
int strange(int n);

int main(void)
{
      printf("%d\n", strange(7));
}
int strange(int n)
{
      int ans;
      if (n == 1)
            ans = 0;
      else
            ans = 1 + strange(n / 2);
return (ans);
}
```

Answer:

strange(7) = 2.
strange(n) computes the integer portion of the $\log_2(n)$, e.g. $\log_2(7) = 2.807$

in fact, looking at the function we can deduce that strange(2^x)= x;

4. Write a recursive function `find_sum` that calculates the sum of successive integers starting at 1 and ending at `n`
(i.e., `find_sum(n) = (1 + 2 + . .+ ( n – 1) + n`).

Answer:

Sum(n)= **1 + 2 + . . . + (n – 1) + n**

For n> 0, the recursive implementation of the function is :
sum(1)=1
sum(n)=n+sum(n-1), when n>1

```
/*
 *  Computes (1 + 2 + 3 + ... (n-1) + n) using a recursive
    definition.
 *  Pre:  n > 0
 */
    int find_sum(int n)
    {
        int ans;
```

```
            if (n == 1)
                ans = 1;
            else
                ans = n + find_sum(n - 1);

            return (ans);
        }
```

The following gives the iterative and recursive solution to this problem

```
#include<stdio.h>
int iter_sum(int n);
int recu_sum(int n);
int main(){
    int n=3;
    printf("%d\n", iter_sum(n));
     printf("%d\n", recu_sum(n));

}

int recu_sum(int n){
    //design S.C n==1; retur 1
    // g. recu_sum(n)=recu_sum(n-1)+n
    if (n<1) return -1;
    if(n==1) return 1;
    else
    return(recu_sum(n-1)+n);
}

int iter_sum(int n){
    if (n<1) return -1;
    int acc=0;
    for (int i=1; i<=n;i++)
      acc+=i;

    return acc;
}
```