COL100 Lab 9

I semester 2016-17

Week 9, 2016

Objective

To be able to write C programs involving functions and recursion.

Instructions

1. There are multiple ways to achieve a task in C. Please follow the method that the questions asks you to do.

Examples

Functions

Here is an example to add 2 numbers using a function, $add2Num()$:

```
#include <stdio.h>

int add2Num(int a, int b);      // function prototype (it is exactly
    like the function definition, with a SEMICOLON at the end)

int main()
{
    int x, y, sum = 0;

    printf("Enters two numbers: ");
    scanf("%d %d",&x,&y);

    sum = add2Num(x, y);     // function call

    printf("sum = %d\n",sum);
    return 0;
}

int add2Num(int a,int b)      // function definition (note that the
    name of the function is preceded by the return type (type of
    the result that will be generated), while "int a" & "int b"
    denotes that the INPUT to this function is an integer)
{
    int result;
    result = a+b;
    return result;                // returns sum
}
```

Recursion

The process of a function calling itself repeatedly is known as recursion. Let us see how we can calculate the sum of the first n natural numbers using recursion.

```
#include <stdio.h>

int add(int a); //function prototype

int main()
{
    int n, sum = 0;
    {
      printf("Enter the value of n");
      scanf("%d", &n);
      sum = add(n); // Call the function
        printf("\nSum is [%d] \n", sum);
    }

    return 0;
}

int add(int a) // function definition
{
    if(n!=0)
        return n+add(n-1);
    else
        return n;
    return;
}
```

Here, every call to the function $add()$ is broken down to further calls to $add()$, until we reach zero. The condition for termination is provided by the *base case*. A base case is the bottom point of a recursive program where the operation is so trivial as to be able to return an answer directly. All recursive programs must have at least one base case and must guarantee that they will hit one eventually; otherwise the program would run forever or until the program runs out of memory or stack space.

Let us assume the input is 3. So here is what $add()$ will return in each step:

1. $sum = add(3)$
2. $sum = 3 + add(2)$
3. $sum = 3 + 2 + add(1)$
4. $sum = 3 + 2 + 1 + add(0)$
5. $sum = 3 + 2 + 1 + 0$

So, what will happen if we do not include the condition $if(n! = 0)$ in the function $add()$?

Programs

1. Write a C program to find the sum of digits of an integer **using recursion**.

   Input format: an integer

   Output format: sum of digits, followed by a newline character.

2. (a) Write a C function, $checkPrime(int\ a)$ to check if the given number is prime or not.

   (b) Take an integer as input from the user, and use this function ($checkPrime(int\ a)$) to express the integer as a sum of two prime numbers. Choose the numbers in such a way that the first number is the smallest possible prime number. If it is not possible to do so, the program should print 0.
   For example, 22 = 3 + 19 or 5 + 17 ..., but the program should

print "3 19". For 27, it should print "0".

Input format: the integer that has to be expressed as sum of primes.

Output format: If the number can be expressed as sum of two prime numbers, then, print the smallest such prime number first, followed by a single space and then the larger number, followed by '\$n$'. If the number cannot be expressed as sum of two prime numbers, print "0\$n$".

3. Write a program to sort n numbers in ascending order using Merge sort recursively.
   The algorithm for merge sort is as follows:

   (a) Divide the array into two halves

   (b) Sort the two sub-arrays

   (c) Merge the two sorted sub-arrays into a single sorted array

   (d) (sorting the sub-arrays) is done recursively (divide in two, sort, merge) until the array has a single element (base condition of recursion)

   Hint: Write a function *merge*() to perform the merge operation. Then write another function *mergesort*() to perform steps (a), (b) and (c). Call these two functions in the main function.

   Input format: The integer n followed by n numbers on separate lines.

   Output format: The sorted numbers, each followed by a newline character.

4. (a) Write a function `void encrypt(char* str, int k)` to encrypt a string using *Caesar's cipher*, and print it.
   If p is some plaintext (i.e., an unencrypted message), p_i is the i^{th} character in p, and k is a secret key (i.e., a non-negative integer), then each letter, c_i, in the ciphertext (i.e the encrypted message), c, is computed as:

   $$c_i = (p_i + k)\%26 \tag{1}$$

   The user has to enter the string to be encrypted, and the key . Then each letter in the plaintext is 'shifted' by a certain number of places. For example, if the string is "Hello!", and the key is 3, the encrypted text becomes "Khoor!"

(b) Write another function `void decrypt(char* str, int k)` to decrypt a string using *Caesar's cipher*, and print it.

(c) (*Optional*) Write a main function, which takes a string and a key as input, and prints the encrypted version using Caesar's cipher. Then it decrypts the string and prints the decrypted version.

Optional Programs

1. Write a program to implement a variation of the classic wordgame Hangman. For details, you can visit here: `https://en.wikipedia.org/wiki/Hangman_(game)`.

   Requirements

   Here are the requirements for your game:

   The computer must select a word at random from the list of available words. A small list of words has been written in to the program, and a random word has to be selected from this list.

   The game must be interactive; the flow of the game should go as follows:

   At the start of the game, let the user know how many letters the computer's word contains.

   Ask the user to supply one guess (i.e. letter) per round.

   The user should receive feedback immediately after each guess about whether their guess appears in the computer's word.

   After each round, you should also display to the user the partially guessed word so far, as well as letters that the user has not yet guessed.

   Some additional rules of the game: A user is allowed 8 guesses. Make sure to remind the user of how many guesses s/he has left after each round. Assume that players will only ever submit one character at a time (A-Z).

   A user loses a guess only when s/he guesses incorrectly.

   If the user guesses the same letter twice, do not take away a guess - instead, print a message letting them know they've already guessed that letter and ask them to try again.

   The game should end when the user constructs the full word or runs out of guesses. If the player runs out of guesses (s/he "loses"), reveal the word to the user when the game ends.

```
// Hangman game
//

// -----------------------------------
// Helper code
// You do not need to understand this helper code,
// but you will have to know how to use the functions

#include <stdio.h> // input and output
#include <string.h> // strlen()
#include <ctype.h> // toupper()
#include <stdlib.h> // exit()
#include <time.h>  // srand(), rand()

#define NUM_OF_WORDS 21         // The number of words in the words
    list.
#define NUM_OF_CHANCES 8        // The number of tries allowed
    before hanging the hangman


const char* words[NUM_OF_WORDS] = {"REGISTRATION", "ENTERTAINMENT",
    "SECURITY","PASSWORD", "SERVICE", "SUPPORT", "QUICK",
    "ELEPHANT", "BANANA", "YELLOW", "BROWN", "WHITE", "BLACK",
    "GREEN","WRITER","PURPLE", "CHOCOLATE" , "ICECREAM","MAGENTA",
    "CAMPUS"};


char* chooseWord(){
    /*
    Returns a word from wordlist at random
    */
        srand(time(NULL));
        int g = (rand() % (NUM_OF_WORDS));
        return words[g];
}

// end of helper code
// -----------------------------------


int isWordGuessed(char* secretWord, char* lettersGuessed){
    /*
    secretWord: string, the word the user is guessing
    lettersGuessed: array, what letters have been guessed so far
    returns: 1 if all the letters of secretWord are in
        lettersGuessed;
      0 otherwise
    */
    // FILL IN YOUR CODE HERE...

char* getGuessedWord(char* secretWord,char* lettersGuessed){
    /*
    secretWord: string, the word the user is guessing
    lettersGuessed: string, what letters have been guessed so far
    returns: string, comprised of letters and underscores that
        represents
      what letters in secretWord have been guessed so far.
    */
    // FILL IN YOUR CODE HERE...
}


char* getAvailableLetters(char* lettersGuessed){
    /*
    lettersGuessed: string, what letters have been guessed so far
    returns: string, comprised of letters that represents what
        letters have not
      yet been guessed.
    */
    // FILL IN YOUR CODE HERE...
}

int main()
{
    /*
    secretWord: string, the secret word to guess.

    Starts up an interactive game of Hangman.

    * At the start of the game, let the user know how many
      letters the secretWord contains.

    * Ask the user to supply one guess (i.e. letter) per round.

    * The user should receive feedback immediately after each guess
      about whether their guess appears in the computers word.

    * After each round, you should also display to the user the
      partially guessed word so far, as well as letters that the
      user has not yet guessed.

    Follows the other limitations detailed in the problem write-up.
    */

        // FILL IN YOUR CODE HERE...
}
```

Useful Commands in Linux

1. Open terminal: Ctrl + Alt + T
2. Terminate current Linux command: Ctrl + C
3. Make a new directory: mkdir *dirname*
4. Copy: cp *src dest*
5. Rename: mv *originalname newname*
6. Delete: rm *filename*
7. Change working directory: cd *path*
8. List contents of a folder: ls
9. List contents of a folder including hidden files: ls -a
10. Print current directory: pwd

Points to Remember

1. To set proxy: Open an internet browser and set the Automatic proxy configuration url to http://www.cc.iitd.ernet.in/cgi-bin/proxy.btech (or proxy.dual if you are a Dual Degree student).
   (For Firefox, open Options > Advanced > Network Tab > (Connection) Settings > Choose "Automatic proxy configuration" and set the URL)

Optional : Use vim editor

1. Open a file: vim filename.txt
2. Insert in a file: i (insert mode) (Use Esc to come out of the insert mode)
3. Navigation: arrow keys
4. Undo u
5. Redo Ctrl+R
6. Saving a file :w
7. Closing a file without saving :q!
8. Saving and closing a file :wq
9. Deleting a line dd
10. Copying a line yy
11. Pasting a line p