0x08. C - Recursion

**README.md**

**Project**

**0x08. C - Recursion**

**main.h**

**#ifndef MAIN\_H**

**#define MAIN\_H**

**void \_puts\_recursion(char \*s);**

**int \_putchar(char c);**

**void \_print\_rev\_recursion(char \*s);**

**int \_strlen\_recursion(char \*s);**

**int factorial(int n);**

**int \_pow\_recursion(int x, int y);**

**int \_sqrt\_recursion(int n);**

**int is\_prime\_number(int n);**

**int is\_palindrome(char \*s);**

**#endif**

**\_putchar.c**

**#include "main.h"**

**#include <unistd.h>**

**/\*\***

**\* \_putchar - writes the character c to stdout**

**\* @c: The character to print**

**\***

**\* Return: On success 1.**

**\* On error, -1 is returned, and errno is set appropriately.**

**\*/**

**int \_putchar(char c)**

**{**

**return (write(1, &c, 1));**

**}**

**0-puts\_recursion.c**

**#include "main.h"**

**/\*\***

**\* \_puts\_recursion - function like puts();**

**\* @s: input**

**\* Return: Always 0 (Success)**

**\*/**

**void \_puts\_recursion(char \*s)**

**{**

**if (\*s)**

**{**

**\_putchar(\*s);**

**\_puts\_recursion(s + 1);**

**}**

**else**

**\_putchar('\n');**

**}**

**1-print\_rev\_recursion.c**

**#include "main.h"**

**/\*\***

**\* \_print\_rev\_recursion - Prints a string in reverse.**

**\* @s: The string to be printed.**

**\*/**

**void \_print\_rev\_recursion(char \*s)**

**{**

**if (\*s)**

**{**

**\_print\_rev\_recursion(s + 1);**

**\_putchar(\*s);**

**}**

**}**

**2-strlen\_recursion.c**

**#include "main.h"**

**/\*\***

**\* \_strlen\_recursion - Returns the length of a string.**

**\* @s: The string to be measured.**

**\***

**\* Return: The length of the string.**

**\*/**

**int \_strlen\_recursion(char \*s)**

**{**

**int longit = 0;**

**if (\*s)**

**{**

**longit++;**

**longit += \_strlen\_recursion(s + 1);**

**}**

**return (longit);**

**}**

**3-factorial.c**

**#include "main.h"**

**/\*\***

**\* factorial - returns the factorial of a number**

**\* @n: number to return the factorial from**

**\***

**\* Return: factorial of n**

**\*/**

**int factorial(int n)**

**{**

**if (n < 0)**

**return (-1);**

**if (n == 0)**

**return (1);**

**return (n \* factorial(n - 1));**

**}**

**4-pow\_recursion.c**

**#include "main.h"**

**/\*\***

**\* \_pow\_recursion - returns the value of x raised to the power of y**

**\* @x: value to raise**

**\* @y: power**

**\***

**\* Return: result of the power**

**\*/**

**int \_pow\_recursion(int x, int y)**

**{**

**if (y < 0)**

**return (-1);**

**if (y == 0)**

**return (1);**

**return (x \* \_pow\_recursion(x, y - 1));**

**}**

**5-sqrt\_recursion.c**

**#include "main.h"**

**int actual\_sqrt\_recursion(int n, int i);**

**/\*\***

**\* \_sqrt\_recursion - returns the natural square root of a number**

**\* @n: number to calculate the square root of**

**\***

**\* Return: the resulting square root**

**\*/**

**int \_sqrt\_recursion(int n)**

**{**

**if (n < 0)**

**return (-1);**

**return (actual\_sqrt\_recursion(n, 0));**

**}**

**/\*\***

**\* actual\_sqrt\_recursion - recurses to find the natural**

**\* square root of a number**

**\* @n: number to calculate the sqaure root of**

**\* @i: iterator**

**\***

**\* Return: the resulting square root**

**\*/**

**int actual\_sqrt\_recursion(int n, int i)**

**{**

**if (i \* i > n)**

**return (-1);**

**if (i \* i == n)**

**return (i);**

**return (actual\_sqrt\_recursion(n, i + 1));**

**}**

**6-is\_prime\_number.c**

**#include "main.h"**

**int actual\_prime(int n, int i);**

**/\*\***

**\* is\_prime\_number - says if an integer is a prime number or not**

**\* @n: number to evaluate**

**\***

**\* Return: 1 if n is a prime number, 0 if not**

**\*/**

**int is\_prime\_number(int n)**

**{**

**if (n <= 1)**

**return (0);**

**return (actual\_prime(n, n - 1));**

**}**

**/\*\***

**\* actual\_prime - calculates if a number is prime recursively**

**\* @n: number to evaluate**

**\* @i: iterator**

**\***

**\* Return: 1 if n is prime, 0 if not**

**\*/**

**int actual\_prime(int n, int i)**

**{**

**if (i == 1)**

**return (1);**

**if (n % i == 0 && i > 0)**

**return (0);**

**return (actual\_prime(n, i - 1));**

**}**

**100-is\_palindrome.c**

**#include "main.h"**

**int check\_pal(char \*s, int i, int len);**

**int \_strlen\_recursion(char \*s);**

**/\*\***

**\* is\_palindrome - checks if a string is a palindrome**

**\* @s: string to reverse**

**\***

**\* Return: 1 if it is, 0 it's not**

**\*/**

**int is\_palindrome(char \*s)**

**{**

**if (\*s == 0)**

**return (1);**

**return (check\_pal(s, 0, \_strlen\_recursion(s)));**

**}**

**/\*\***

**\* \_strlen\_recursion - returns the length of a string**

**\* @s: string to calculate the length of**

**\***

**\* Return: length of the string**

**\*/**

**int \_strlen\_recursion(char \*s)**

**{**

**if (\*s == '\0')**

**return (0);**

**return (1 + \_strlen\_recursion(s + 1));**

**}**

**/\*\***

**\* check\_pal - checks the characters recursively for palindrome**

**\* @s: string to check**

**\* @i: iterator**

**\* @len: length of the string**

**\***

**\* Return: 1 if palindrome, 0 if not**

**\*/**

**int check\_pal(char \*s, int i, int len)**

**{**

**if (\*(s + i) != \*(s + len - 1))**

**return (0);**

**if (i >= len)**

**return (1);**

**return (check\_pal(s, i + 1, len - 1));**

**}**

**101-wildcmp.c**

**#include "main.h"**

**/\*\***

**\* wildcmp - Compare strings**

**\* @s1: pointer to string params**

**\* @s2: pointer to string params**

**\* Return: 0**

**\*/**

**int wildcmp(char \*s1, char \*s2)**

**{**

**if (\*s1 == '\0')**

**{**

**if (\*s2 != '\0' && \*s2 == '\*')**

**{**

**return (wildcmp(s1, s2 + 1));**

**}**

**return (\*s2 == '\0');**

**}**

**if (\*s2 == '\*')**

**{**

**return (wildcmp(s1 + 1, s2) || wildcmp(s1, s2 + 1));**

**}**

**else if (\*s1 == \*s2)**

**{**

**return (wildcmp(s1 + 1, s2 + 1));**

**}**

**return (0);**

**}**