Segundo Laboratorio Calificado

Alumno: Moreno Vera, Felipe Adrian.

Codigo: 20120354I.

Curso: Estructura de Datos, CM-098.

Tema: Lista Enlazada Simple, Pilas.

Trabajo: Operaciones basicas con Numeros grandes.

1.Concepto preliminar:

A nivel de computadoras, se sabe que sea cual sea la capacidad de memoria de un computador tiene limite.

Por ejemplo, para realizar una suma de 2 numeros, la computadora trabaja a nivel de bytes donde cada numero tendra cierta longuitud de bytes (depende de la arquitectura del sistema) por lo que a veces hacer una operación de una simple suma puede ser factible hasta cierto tope.

Podemos sumar 12 + 10 = 22 , pero la maquina lo trabaja en binario (12, 10 y 22 en binario).

Un lenguaje de programacion nos permite comunicarnos con el computador, ya sea un lenguaje interpretado como Python o compilado como C.

Cada lenguaje de programacion tiene sus tipos de variables que tienen una cierta longuitud de bytes. Por ejemplo, un dato tipo entero (int) en C tiene una longuitud de 4 bytes. Un char tiene 1 byte, es decir su valor varia desde 0 a 255 (1 byte son 8 bits donde cada bit es un espacio representado por 0's o 1's si tengo 00000010 binario sera igual a 4 decimal y 11111111 = 255).
Pero según lo anterior, cada tipo de dato tendra un limite, por ejemplo los tipo int en C tendran un limite de 2^32 -1, considerando al tipo long (8 bytes) el tipo de dato con mas capacidad (2^64 -1)

Ahora que pasaria si quisieramos realizar una suma de numeros de 100 cifras o mas cifras, la memoria no podria soportar y ocurriria un overflow (rebasa la memoria), nuestro problema seria como lograr esa suma.

Punteros: se define asi al tipo de variable definida por ejemplo int *a;
se dice que a apunta a una direccion de memoria.

2.Objetivo:

Implementar un programa que logre la suma de numeros grandes (100 cifras o 200 cifras o incluso mas, donde el limite del numero sea la memoria misma del computador y no del tipo de dato).

3.Programa:

Una vez que se implemento el programa, pasaremos a especificar como se logro la suma de numeros.

El programa se distribuye en 3 archivos y estan escritos en el lenguaje de programacion C, creado por Dennis Ritchie ayudado por Brian Kernighan.

El primer archivo se llama nm_main.c, es en donde esta toda la logica de la suma y resta.
El segundo archivo se llama nm_fun.c, es en donde estan escritas todas las funciones usadas.
Y por ultimo el tercer archivo es un header (cabecera) llamado nm_fun.h.

HEADER: nm_fun.h

Empecemos por el header, en este archivo se definen las funciones y el tipo de dato llamado Pila, que es un tipo estructura con 2 elementos, una variable tipo char que contiene la informacion o dato de la Pila y otra variable tipo struct * que es un apuntador a otra variable Pila.

Las funciones vienen escritas por su tipo de retorno su nombre (parametros) y precedidos por un tipo extern

ejemplo: extern void Crear_pila(Pila *l); // lo monto como funcion externa con retorno tipo void (no retorna nada) de nombre Crear_pila y de parametro una variable l tipo Pila * que es un tipo struct.

FUNCIONES: nm_fun.c

Aqui se definen la estructura de las funciones descritas en el header con el mismo nombre “ nm_fun.h”

se puede apreciar que hay 3 llamadas a biblioteca mediante #include <stdio.h> , #include <stdlib.h> y #include “nm_fun.h” que es el que llama al Header definido.

MAIN: nm_main.c

Finalmente la funcion main que se encarga del manejo de todas las funciones descritas anteriormente.

Hagamos un listado de las funciones implementadas:

1. int strrcbff(char *str,int n);
Esta funcion tiene el proposito de eliminar el buffer de entrada, es decir, las teclas presionadas. Para que no haya el clasico problema de C que despues de ingresar una cadena o carácter despues de otro, el siguiente tome el salto de linea.

2. void Crear_pila(Pila *l);
Esta funcion inicializa una variable tipo Pila* apuntandolo a NULL (nil).

3. int esPila_vacia(Pila l);
Retorna el valor de verdad evaluando si una Pila esta vacia o no.

4. void Destruir_pila(Pila *l);
Devuelve un apuntador tipo Pila a NULL (nil).

5. Pila Crear_pila_sgte();
Esta funcion nos ahorra el tiempo de estar escribiendo (struct pila*)malloc(struct pila) para reservar memoria.

6. int Generar_pila(Pila *l,int n);
Con esta funcion agregamos el numero ingresado por teclado a una pila.

7. void Pila_insert_left(Pila *l,char dato);
insertamos por izquierda de la pila.
La pila se maneja insertando por derecha y eliminando por derecha (LI-FO)
pero por convencion al momento de la suma/resta necesitamos invertir la pila y quitar ya no por derecha sino por izquierda.

8. void Pila_insert_right(Pila *l,char dato);
insertamos por derecha de la pila

9. char Pila_delet_right(Pila *l);
Borra por derecha de la pila.

10. int Buscar_elem_Pila(Pila l,char info);
Retorna si encuentra un elemento en una pila, en este programa lo usamos para buscar un signo '-'.

11. Pila Invertir_pila(Pila *l);
Si una pila es 3->2->1->NULL , devolvera 1->2->3->NULL .

12. void Pila_print(Pila l);
Imprime la pila en pantalla.

13. int esNumero(char i);
Es para evaluar que los datos ingresados solo sean los caracteres '0' hasta '9' y el signo negativo '-'.

14. void Completar_zeros(Pila *l,int n,int m);
Esta funcion completa zeros, 123 + 5 = 123 + 005.

15. Pila Suma(Pila l,Pila q);
Esta funcion retorna una Pila que es la suma de la pila l y pila q

16. Pila Resta(Pila l,Pila q);
Esta funcion retorna una lista que es la resta de la pila l y pila q.

Notas adicionales

Explicacion sobre los punteros

&: es un operador unario que nos devuelve la direccion de memoria de una variable.
Int c;
int *p=&c; // esto podria ser 0x12ffffff. Que es una direccion de memoria
p tiene almacenada la direccion de memoria de la variable c que es tipo int.

*: es un operador unario que nos devuelve el contenido de una direccion de memoria.
Int c=10; // c es una variable tipo int con valor 10.
Int *p; // p es una variable cuyo contenido es una direccion de memoria de una variable tipo int.
p=&c; // a p le estoy asignando la direccion de memoria de la variable c.
// entonces el valor de *p = 10.
//porque recordemos: & devuelve una direccion de memoria y * no da el contenido de una direccion
//de memoria.
//Entonces *p = *(&c) =10