

Estructura y contenido

1. Definición y especificación de requerimientos:

a) Definición general del proyecto de software:

Idea general:

Desarrollar en el lenguaje de programación C un programa mediante el cual evaluar expresiones aritméticas en notación prefija.

Objetivos:

Se brinda un mecanismo con el cual se logran resolver expresiones aritméticas en notación prefija y devolver su correspondiente resultado con el fin de facilitar la resolución de éstas expresiones.

Usuarios:

Este programa está dirigido hacia cualquier usuario que tenga la necesidad de resolver una expresión aritmética, pero para poder utilizar el programa dicho usuario debe tener conocimiento de expresiones aritméticas formuladas en notación preorden, ya que la forma en la cual el sistema acepta estas expresiones es únicamente en esta notación. También se brinda la opción de ejecutar este sistema desde la línea de comandos para aquellos usuarios que tengan experiencia en esta sección.

El presente informe está dirigido hacia los usuarios finales comentados anteriormente y hacia desarrolladores de software que busquen información respecto a cómo se desarrolló este sistema.

El módulo de evaluar de éste sistema es una de las entidades que interactúa con el propio sistema. Esto se basa en que dicho módulo utiliza diversas funciones para calcular la expresión ingresada, y para que estas funciones se lleven a cabo deben utilizar otras partes del sistema, es decir, necesitan utilizar los módulos de lista y pila para poder funcionar correctamente.

b) Especificación de requerimientos del proyecto:

• Requisitos generales:

El proyecto consta de las siguientes pautas:

- 1) El TDA lista está implementada con memoria dinámica mediante las funciones de librería void free (*void ptr) y void *malloc (size_t size). Este TDA debe ser capaz de almacenar variables enteras.
- 2) El TDA lista debe ser capaz de almacenar variables string.

- 3) La operación aritmética ingresa por la entrada estándar del sistema (lectura por teclado).
- 4) El programa conforma la especificación \$ evaluar [-h] al ser invocado desde la línea de comandos, donde el parámetro -h es opcional y brinda un texto de ayuda.

- **Requisitos funcionales:**

Para que la expresión aritmética sea correctamente evaluada, ésta debe cumplir con ciertas pautas:

- 1) Las expresiones aritméticas utilizan la siguiente sintaxis:
(< operador > <operando₁> <operando₂> ... <operando_n>)
- 2) Los operadores, operandos y paréntesis pueden ser separados por cualquier cantidad de espacios en blanco.
- 3) Los operandos son números enteros sin signo, de uno o más dígitos.
- 4) Los operadores + y * pueden recibir más de dos operandos. Los operadores – y / sólo pueden recibir dos operandos.
- 5) Las expresiones deben ser ingresadas en preorden.
- 6) Si se ingresa una única variable, no se debe ingresar operador, y esta variable debe ser ingresada sin estar acompañada de paréntesis.

- **Información de autoría y Legacy del proyecto:**

Este proyecto forma parte de un desarrollo original. Es compatible bajo el sistema operativo Debian, a partir de la versión gcc (Debian 4.9.2-10)4.9.2.

- **Alcances del sistema:** Este programa solo realiza las operaciones matemáticas básicas (multiplicación, resta, división y suma) sobre únicamente números enteros. Además, las operaciones de resta y división sólo serán efectuadas exclusivamente sobre una cantidad de dos operandos.
Sólo serán aceptadas expresiones aritméticas de una longitud menor o igual a 250 caracteres.

c) **Procedimientos de instalación y prueba:**

Procedimientos de desarrollo:

- **Herramientas utilizadas:** Este sistema fue desarrollado en el lenguaje de programación C, en el sistema operativo Debian, en la versión gcc (Debian 4.9.2-10)4.9.2. La escritura del código del programa se llevó a cabo en el entorno de desarrollo Code::Blocks. Se utilizó además LXTerminal para la verificación del sistema desde una consola de comandos.

- **Planificación:**

Se comenzó la resolución del problema con la lectura de los requisitos establecidos en los TDA lista y pila, se elaboraron los algoritmos correspondientes a cada operación que debían contener cada uno de estos TDA y se procedió con la implementación de estos algoritmos. Luego de esto se testearon con diversos casos cada uno de los servicios que estos TDA proveen.

Al finalizar la construcción de los módulos lista y pila que contienen a los TDA lista y pila respectivamente, se continuó con la comprensión del módulo núcleo evaluar a ser creado, donde debía desarrollarse la evaluación de la expresión solicitada por el usuario. Para este módulo se elaboraron diversos algoritmos y se evaluaron cada uno de estos algoritmos hasta encontrar el más óptimo para resolver el cometido de este proyecto. Luego de implementar éste módulo con el algoritmo elegido, se procedió a testearlo con diversos casos de expresiones aritméticas donde dicha expresión estaba correctamente ingresada. Al finalizar los test con expresiones correctas y corregir errores en la implementación, se relevaron los requisitos que el proyecto debía cumplir, con lo cual se agregaron los casos particulares donde se ingresase una expresión incorrecta y se volvió a testear el sistema con diversas expresiones, tanto bien ingresadas como mal formadas. Finalmente, se volvieron a leer los requisitos de este proyecto y se controló que el sistema cumpliera con cada uno de ellos.

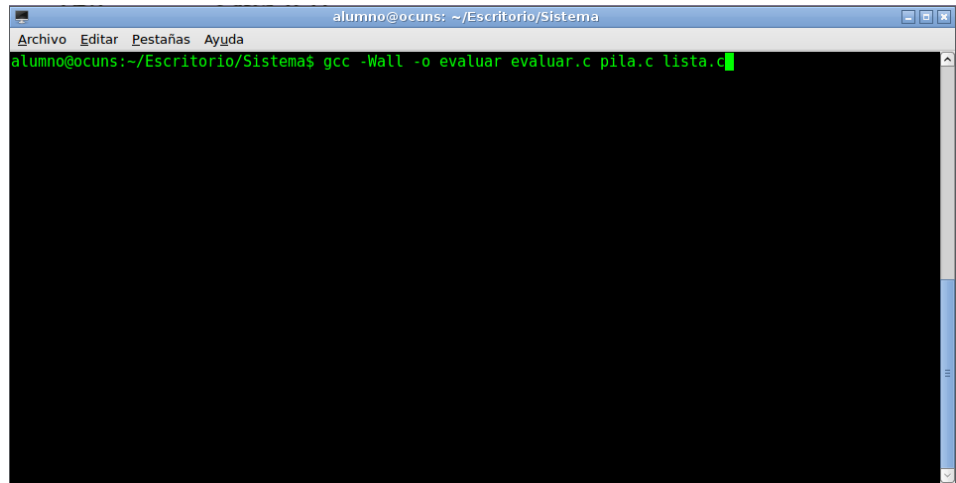
Procedimientos de instalación y prueba:

- **Requisitos no funcionales:** El correcto funcionamiento del sistema se verá afectado si no se tiene en cuenta el máximo tamaño permitido de la expresión a introducir especificado en la sección “Alcances del sistema”.
- **Obtención e instalación:** Este programa centra su función principal en un archivo ejecutable por lo cual no consta de instalación alguna, sino que para su utilización se debe contar únicamente con la carpeta contenedora de los archivos pertenecientes al programa. No obstante para poder ejecutar dicho ejecutable, se debe disponer de la consola de comandos, cuyo uso se explica en “Especificación de prueba y ejecución”.
- **Especificaciones de prueba y ejecución:**
Este sistema fue desarrollado en el sistema operativo Debian, versión gcc (Debian 4.9.2-10)4.9.2. El sistema funciona correctamente bajo esta plataforma. No obstante, esto no implica que bajo otros sistemas

operativos este programa no se ejecute correctamente, sino que no ha sido testeado bajo otras plataformas.

Este sistema se ejecuta desde LXTerminal.

Para poder llevar a cabo la ejecución, se debe posicionar en la carpeta contenedora del sistema. Una vez que se haya ingresado a la carpeta se debe ejecutar la sentencia “gcc -Wall -o evaluar evaluar.c pila.c lista.c”

A screenshot of an LXTerminal window. The title bar at the top reads 'alumno@ocuns: ~/Escritorio/Sistema'. Below the title bar is a menu bar with the options 'Archivo', 'Editar', 'Pestañas', and 'Ayuda'. The terminal area has a black background with green text. The prompt 'alumno@ocuns:~/Escritorio/Sistema\$' is followed by the command 'gcc -Wall -o evaluar evaluar.c pila.c lista.c'. A green cursor is positioned at the end of the command line.

```
alumno@ocuns:~/Escritorio/Sistema$ gcc -Wall -o evaluar evaluar.c pila.c lista.c
```

En este punto se brinda al usuario un servicio de ayuda donde se explica el funcionamiento del sistema, el cual se ejecuta mediante el comando `./evaluar -h`. Si se ingresa el comando `./evaluar -h x k` se ejecuta un mensaje de error debido al incorrecto ingreso de parámetros.

En cambio si lo que se busca es llevar a cabo la ejecución del sistema evaluador de expresiones, debe de ingresarse el comando “`./evaluar`”, lo que llevara a ejecutar el sistema mostrando el mensaje “Ingresa la notación en expresión prefija”.

2. Arquitectura del sistema

- **Descripción jerárquica:**

Este sistema fue desarrollado con una estructura monolítica, donde se tiene dos módulos pertenecientes a estructuras de almacenamiento y un tercer módulo núcleo (evaluar) donde se lleva a cabo la función principal del sistema mediante la utilización de recursos de librerías y de servicios provistos por los dos módulos de estructuras de datos.

- **Diagrama de módulos:**



- **Descripción individual de los módulos:**

Módulo de lista:

- **Descripción general y propósito:**

Es una estructura de datos que consiste en una secuencia de celdas donde cada celda conoce la celda posterior a esta si es que existe y un conjunto de hasta 4 enteros como máximo.

Este módulo debe ser capaz de crear nuevas listas, almacenar valores enteros en la lista y posición que se indica, eliminar un elemento de la lista y posición que se indica, indicar cuantos elementos posee la estructura y permitir agregar un nuevo entero a la última posición de la lista.

- **Responsabilidad y restricciones:**

Su función específica dentro del sistema consta de almacenar valores de variables enteras para futuros usos.

- **Dependencias:**

_Para que las operaciones de este módulo funcionen la lista además de ser declarada tiene que anteriormente haber sido inicializada.

_Para que la lista sea implementada con memoria dinámica, lo cual es una pauta definida inicialmente, esta debe de contar con los métodos de librería *void free (*void ptr)* y *void malloc (size_t size)*.

_Debe poder acceder a la librería <lista.h> donde se encuentra la definición del tipo de dato lista.

- **Implementación:** La implementación de esta estructura se encuentra dentro del archivo "lista.c", y la definición del tipo de dato se encuentra dentro del archivo "lista.h".

Módulo de pila:

- **Descripción general y propósito:**

Se trata de una estructura de datos que almacena en este caso string, y

que además conoce el elemento string que le sigue a ella, si es que lo hay. Esta estructura otorga los servicios de crear nuevas pilas, insertar nuevos string en la estructura, saber cuál es el elemento que se encuentra en el tope de la pila, poder eliminar/desapilar un elemento de la pila y conocer si una pila está o no vacía.

- **Responsabilidad y restricciones:**

Su función específica dentro del sistema es la de almacenamiento de strings (cadena de caracteres).

- **Dependencias:**

A excepción de la operación que crea una nueva pila, los restantes servicios de este módulo requieren que la pila haya sido inicializada; y además ciertos servicios requieren no solo una pila inicializada, sino que esta tenga contenida un elemento (string) en su tope.

_Debe poder acceder a la librería <pila.h> donde se encuentra la definición del tipo de dato pila.

- **Implementación:** La implementación de esta estructura se encuentra dentro del archivo “pila.c”, y la definición del tipo de dato se encuentra dentro del archivo “pila.h”.

Módulo de evaluar:

- **Descripción general y propósito:**

Es un módulo encargado de procesar expresiones aritméticas en notación preorden. Este módulo debería leer dicha expresión mediante entrada estándar (lectura por teclado), evaluarla y retornar el resultado obtenido de dicha expresión.

- **Responsabilidad y restricciones:**

Este módulo brinda el mecanismo por el cual la expresión es obtenida mediante la lectura de teclado y además es el encargado de evaluar dicha expresión si es que esta fue correctamente ingresada o retornar el error que se produjo. Contiene además aquellos servicios necesarios para llevar a cabo la ejecución del sistema desde una consola de comandos.

- **Dependencias:**

Para que este módulo pueda ejecutarse correctamente, requiere de los servicios que brindan los módulos “lista” y “pila”, y además de ciertas operaciones ubicadas en las librerías <stdio.h>, <stdlib.h> y <string.h>.

- **Implementación:** La implementación de los métodos de este módulo se encuentran dentro del archivo “evaluar.c”.

- **Dependencias externas:**

Librerías utilizadas:

<string.h>:

strlen (const char *_s): Calcula la longitud de una cadena de caracteres.

strcmp (const char *s1, const char *s2): Compara dos cadenas lexicográficamente. Retorna 0 en caso de que sean iguales.

<stdio.h>:

printf(...): Imprime por pantalla un mensaje con un tipo de formato establecido y una cantidad acorde de variables para cada formato dentro del mensaje.

<stdlib.h>:

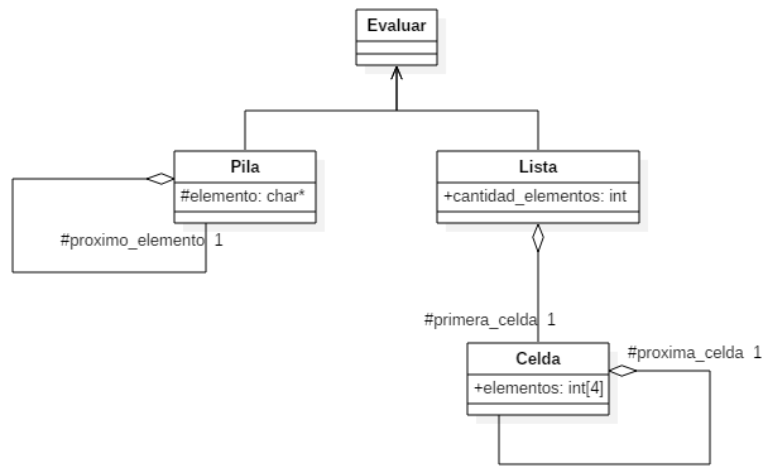
exit(...): Termina la ejecución del sistema con un valor entero pasado como parámetro.

Se ha utilizado además un método para la lectura de la expresión por teclado para que la cadena resultante almacenada a ser evaluada contuviese dicha expresión en su totalidad y no la cadena introducida hasta el primer espacio en blanco leído, que es la manera por defecto que utiliza C para la lectura de string por teclado. Dicho método fue obtenido del siguiente enlace:

<http://stackoverflow.com/questions/4023895/how-to-read-string-entered-by-user-in-c>

Este sistema fue desarrollado bajo el lenguaje de programación imperativo C

3. Diseño del modelo de datos



4. Descripción de procesos y servicios ofrecidos por el sistema

Este sistema ofrece un único servicio, cuya funcionalidad radica en evaluar expresiones aritméticas expresadas en preorden, también conocida como notación prefija. Esta notación tiene como principal característica que coloca los operadores a la izquierda sus operandos. Por ejemplo, dada la expresión en notación infija convencional “51 / 37 + 6” se expresa en preorden como “(+ (/ 51 37) 6)”

El programa recibe por entrada estándar (esto es, por la consola de comandos) una expresión aritmética expresada en notación prefija siguiendo la siguiente sintaxis:

(<operador> <operando₁> <operando₂> <operando₃> ... <operando_n>)

Dónde:

- Los operadores, paréntesis y operandos están separados por cualquier número de espacios en blanco
- Los operandos deben sí o sí ser números enteros (sin signo)
- Sólo se pueden realizar operaciones de suma, resta, multiplicación o división, donde cada operador de resta y de división deben tener sólo dos operandos, mientras que los operadores de suma y multiplicación pueden tener 2 o más.

Una vez ingresada una expresión aritmética correcta, el programa se encarga de evaluarla haciendo uso de los TDA Pila y Lista para llevar a cabo la ejecución del algoritmo.

En primer lugar, se testea si los paréntesis de la expresión están bien balanceados. Luego de esto se recorre cada uno de los caracteres de la expresión insertándolos en una pila hasta llegar al final de la expresión. Si en la lectura de caracteres se encuentra un “)” (paréntesis que cierra), se obtiene el resultado de evaluar la expresión entre este paréntesis que cierra encontrado y el paréntesis que abre más cercano en la pila a través de un método que utiliza el TDA Lista para ingresar los operandos y llamar a un tercer método encargado de realizar la operación correspondiente con los operandos de dicha lista. Luego se incrementa la posición actual en la expresión (con el fin de avanzar de posición saltando el paréntesis que cierra encontrado anteriormente) y se apila el resultado obtenido en forma de string. Al terminar de leer la expresión, se tendrá, contenido en la pila, el resultado de haber evaluado la expresión completamente y luego de desapilarlo, se mostrará el mismo en la salida estándar del sistema (consola de comandos).

5. Documentación técnica – Especificación API

Métodos de evaluar:

void ayuda (): Este es el método que se encarga de mostrar por pantalla una ayuda para que el usuario entienda el funcionamiento del simulador.

void error_parametros (): Este es el método que se encarga de mostrar por pantalla un mensaje de error en el caso de que el usuario ingrese mal los parámetros del programa.

void main (): Este método se encarga de controlar los parámetros de entrada al llamar al programa y llamar al método correspondiente según corresponda, si los parámetros se ingresaron mal llamará a error_parametros (), si se ingresó el parámetro de ayuda se llamará a ayuda (), si no se sucede ninguna de estas dos situaciones se llama a iniciar () y comienza la ejecución de la parte principal del programa.

int getLine (char* mensaje, char* buff, size_t size): Este método se encarga de mostrar por consola el mensaje “mensaje” recibido por parámetro y luego leer de consola la entrada del usuario almacenándola en el parámetro “buff” controlando que no se produzca un overflow del buff en caso de superar el tamaño “size” recibido por parámetro. El método retorna 0 en caso de éxito, 12 si la entrada fue demasiado larga y 13 si no hubo ninguna entrada.

int str_cut (char *str, int begin, int len): Este método recibe por parametro una cadena de caracteres “str” que se desea cortar desde algún punto marcado por el parámetro “begin” (posición en la cadena). El parámetro “len” indica la cantidad de caracteres que se desean cortar a partir de begin. El método hace uso de la función provista por la librería string.h void *memmove (void *str1, const void *str2, size_t n) que copia “n” caracteres de “str2” a “str1”. Luego el método retorna la cantidad de caracteres cortados de “str”.

int iniciar (): Esta función hace uso del método getLine para pedirle al usuario que ingrese la expresión en notación prefija y una vez que la obtiene evalúa si solo se ingresó un único elemento. En caso de ser así, si el elemento es un numero lo retorna como resultado, si no es un numero retorna el error OPND_INV. En caso de que no se haya ingresado un único elemento, la función debe comenzar con “(” y si esto es así, se llama al método evaluar para resolver la expresión aritmética.

int comprobar_parentesis_balanceados (char expresion [], int size): Esta función recibe una cadena de caracteres “expresión” y su respectivo tamaño a través del parámetro “size” y evalúa si tiene paréntesis correctamente balanceados. Para hacer esto, crea una pila y recorre la expresión buscando paréntesis, cada vez que encuentra un paréntesis que abre “(” lo apila y cada vez que encuentra un paréntesis que cierra “)”, intenta desapilar

un paréntesis que abre, en caso de que la pila esté vacía, la expresión está mal balanceada y se retorna "0" y de la misma forma, si se termina de recorrer toda la expresión y al final la pila no está vacía (esto es, quedaron paréntesis que abren apilados) también significa que la expresión está mal balanceada y se retorna "0".

int suma (lista_t operandos): Esta función recibe una lista "operandos" de elementos enteros y se encarga de sumar todos los elementos que estén contenidos en ésta para luego retornar el resultado de dicha suma.

int resta (lista_t operandos): Esta función recibe una lista "operandos" que contiene sólo dos elementos enteros y se encarga de restar el primer elemento del segundo para luego retornar el resultado de dicha resta.

int division(lista_t operandos): Esta función recibe una lista "operandos" que al igual que "resta" contiene sólo dos elementos enteros y se encarga de dividir el segundo elemento por el primero para luego retornar el resultado de dicha división.

int multiplicación (lista_t operandos): Esta función recibe una lista "operandos" de elementos enteros y se encarga de multiplicar todos los elementos que estén contenidos en ésta para luego retornar el resultado de dicho producto.

int transformar_a_int (char* var, int size): Esta función recibe una cadena de caracteres "var" y su respectivo tamaño "size" por parámetro. La cadena de caracteres está compuesta sólo por caracteres que representan dígitos decimales y la función se encarga de devolver el valor entero que representa dicha cadena pero como una variable de tipo int.

int operar (lista_t lista, char* operador): Esta función está encargada de decidir qué operación aritmética entre las cuatro disponibles se debe realizar. Recibe una lista de elementos enteros y un operador. Si el operador es correcto (esto es, si es un operador de suma, de resta, de multiplicación o de división) se llama al método correspondiente al mismo para que realice la operación sobre la lista de elementos enteros. Luego retorna el resultado.

char* concatenar_pila (pila_t* pila): Esta función recibe como parámetro una pila de tipo pila_t* y se encarga de concatenar los caracteres contenidos en dicha pila en una sola cadena de caracteres. (Esta función podría simplificarse utilizando la función strcat provista por la librería string.h de C pero no logramos que funcione correctamente).

char* evaluar_aux (pila_t* pila): Esta función recibe como parámetro una pila de tipo pila_t* que contiene una operación aritmética en notación prefija en su forma más simple, es decir, es una expresión aritmética de sólo un operador y no una expresión compuesta de otras subexpresiones simples. Ejemplos de expresiones en notación prefija de forma

simple pueden ser: (+ 7 25 3 8), (- 51 26), etc.

char* evaluar (char exp[], int size): Esta es la función principal del programa, recibe como parámetro la cadena de caracteres “exp” y su respectivo tamaño “size”. La cadena exp es la cadena inicial ingresada por consola por el usuario y por lo tanto, la primera labor de esta función es verificar que contiene los paréntesis bien balanceados, en caso de que los paréntesis de la expresión no estén bien balanceados, se termina la ejecución del programa con código de error 2 (EXP_MALF), de lo contrario, se comienza a recorrer la totalidad de la expresión “exp” analizando carácter a carácter y apilando en una pila cada carácter analizado hasta encontrar un paréntesis que cierra (“) ”). Una vez encontrado un paréntesis que cierra éste no se apila y se llama al método “evaluar_aux” para que analice la subexpresión delimitada entre éste paréntesis “)” encontrado y el paréntesis “(” más próximo ya apilado en la pila. Una vez que evaluar_aux termina de ejecutarse, si la subexpresión cumplía las condiciones de ser una expresión aritmética simple expresada en notación prefija, se obtiene el resultado, se apila en la pila (nótese que el resultado queda apilado en el lugar que le corresponde) y se continua con el recorrido de la expresión “exp” repitiendo éste proceso hasta haber llegado al final de la misma. Una vez alcanzado el final de “exp” queda el valor final apilado en la pila en forma de cadena de caracteres y se retorna éste valor al método que haya invocado a esta función “evaluar”.