10 de Octubre del 2016

Programación en Lenguaje C

Organización de Computadoras

Universidad Nacional del Sur
Departamento de Ciencias e Ingeniería de la Computación

Diaz Figueroa, Rodrigo
Figliuolo, Nestor Orlando

Definiciones y especificación de requerimientos

La finalidad del sistema es evaluar expresiones aritméticas que contengan sumas, restas, multiplicaciones y divisiones. Dichas expresiones deberán tener la siguiente forma: *(< operador > < operando 1 > < operando 2 > . . . < operando n >).*
Este sistema esta apuntado a personas que necesitan una solución a la hora de evaluar expresiones del tipo mencionado anteriormente, estos usuarios para poder usar el sistema deben tener conocimientos de como ejecutar un programa por consola y como ingresar los parámetros que sean adecuados.

El sistema cumple con los siguientes requerimientos:

- *Los operadores, paréntesis y operandos están separados por cualquier número de espacios en blanco.*
- *Los operandos son números enteros sin signo, de uno o más dígitos.*
- *Los operadores + y * pueden recibir dos o más operandos.*
- *Las expresiones se interpretan en preorden. Algunos ejemplos de expresiones válidas son:*
- *La expresión (+ (/ 51 37) 6) se interpreta como 51 / 37 + 6.*
- *La expresión (∗ (+ 9 12 3) 4 5 (− 3 2)) se interpreta como (9 + 12 + 3) ∗ 4 ∗ 5 ∗ (3 − 2)*
- *Para evaluar la expresión aritmética, se debe diseñar e implementar un algoritmo no recursivo, que utilice el TDA pila_t implementado anteriormente. Este algoritmo, además de resolver el problema, debe respetar las siguientes consideraciones:*
- *Si alguno de los operadores no corresponde a un operador válido, debe mostrar un error y finalizar la ejecución con exit status OPRD_INV.*
- *Si la cantidad de operandos es insuficiente para aplicar el operador, debe mostrar un error y abortar con exit status OPND_INSUF.*
- *Si el operador es / o -, y la cantidad de operadores es > 2, debe mostrar un error, y abortar con exit status OPND_DEMAS.*
- *Si alguno de los operandos no corresponde a un número entero válido, debe mostrar un error, y terminar con exit status OPND_INV.*
- *Si al evaluar la expresión no se encuentra un símbolo (, debe mostrar un error y terminar con exit status EXP_MALF.*
- *Si al evaluar la expresión, se encuentra que un (o) no cuenta con su correspondiente) o (, debe mostrar un error y terminar con exit status EXP_MALF.*
- *Si hay un único elemento en la expresión, y es un entero, debe mostrar el resultado y terminar con exit status EXITO. Si el elemento no es un entero, debe terminar con exit status OPND_INV.*
- *La operación aritmética ingresa por la entrada estándar del sistema.*
- *Cada operación es evaluada por una función, que toma cómo parámetro una lista de operandos, contenidos en una estructura de tipo lista_t. Por ejemplo, la operación suma (+), es evaluada por la función int suma(lista_t operandos).*

El sistema es un desarrollo original, es decir no deriva ni esta originado en base a otro sistema.

Para obtener el codigo fuente del sistema se puede clonar el repositorio ubicado en el siguiente enlace: https://github.com/Makaan/ProyectoC/tree/final.

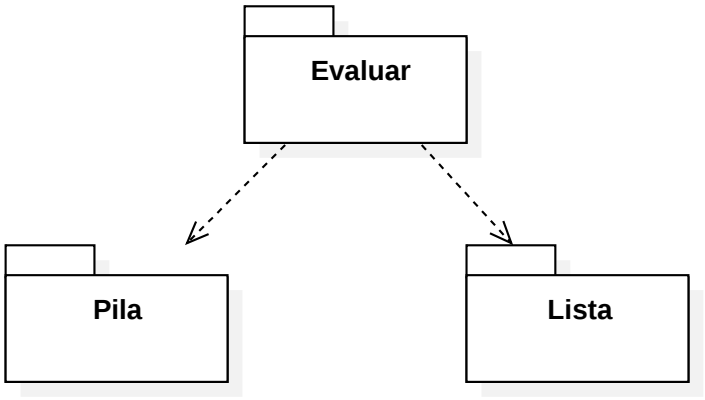
El programa debe ser compilado sobre un sistema operativo GNU/Linux con arquitectura x86 o x86_64 para garantizar su funcionamiento. Dicha compilación debe ser realizada con el

compilador GCC (*https://gcc.gnu.org/*). Una vez finalizada la compilación el programa esta listo para funcionar.

Arquitectura del sistema

El sistema esta conformado por tres módulos separados en base a su funcionalidad:

- Evaluar
- Pila
- Lista



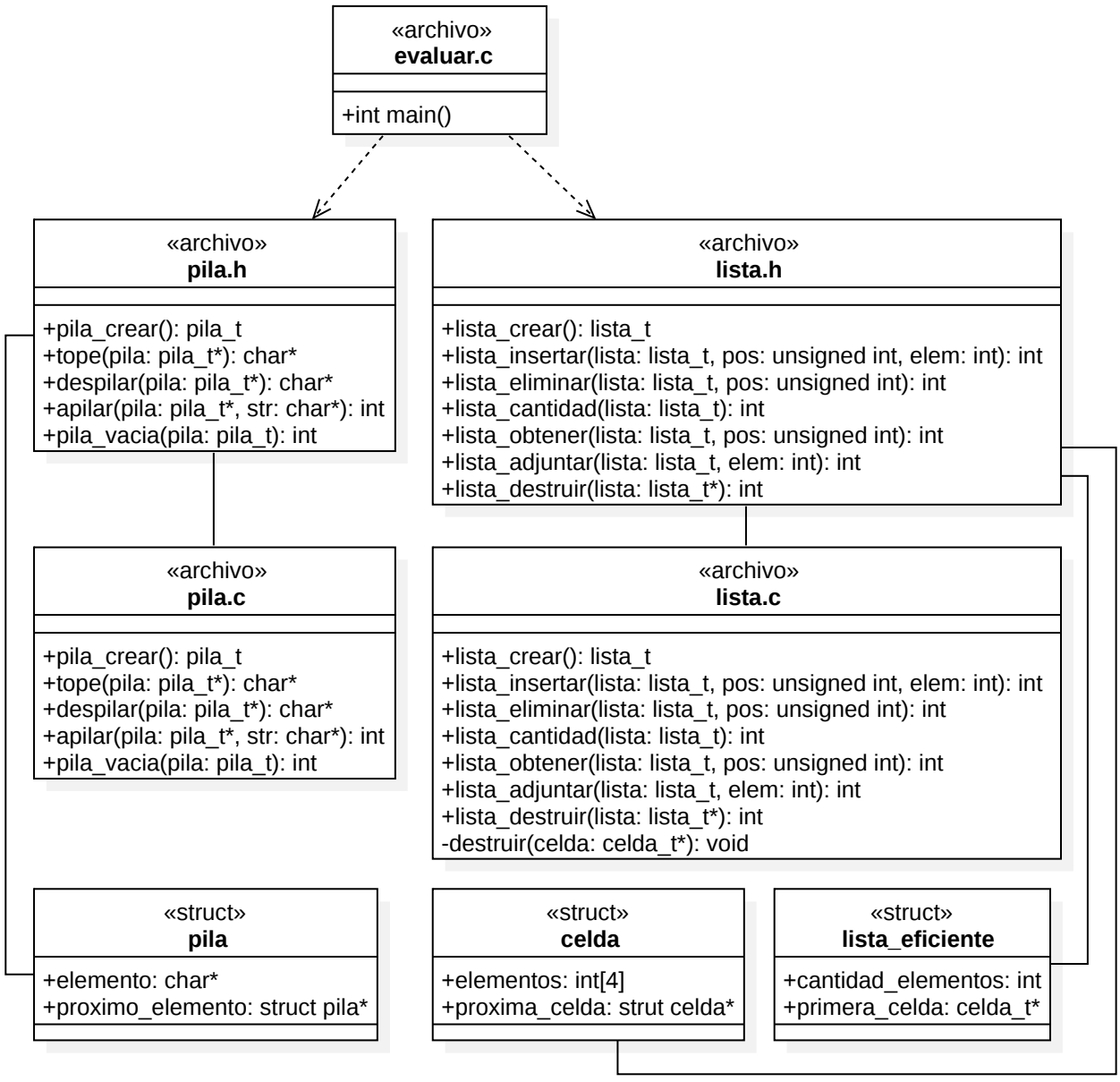
Cada modulo tiene la siguiente descripción:

- Evaluar: aquí esta empaquetado el programa principal, y por ende en el se encuentra toda la implementación concerniente al manejo de los datos de entrada/salida y a la resolución propiamente dicha del problema definido anteriormente. Este modulo depende de los otros dos para funcionar, así también como de la librería de entrada/salida estándar *(stdio)*, la librería estándar *(stdlib)* y la librería para manipulación de cadenas de caracteres *(string),* todas provistas por el lenguaje de programación.
- Pila: empaqueta la estructura de datos pila, esta se encarga de modelar una pila con todas sus funcionalidades para que pueda ser utilizada a la hora de resolver el problema almacenando los operadores y paréntesis de la expresión ingresada al sistema. Depende de la librería de entrada/salida estándar *(stdio)* y la librería estándar *(stdlib).*
- Lista: El paquete “Lista” empaqueta todas las funciones necesarias para poder crear y usar estructuras de datos tipo lista y con ellas poder almacenar los números con los cuales se va a evaluar la expresión. Este paquete depende de a librería de entrada/salida estándar *(stdio)* y la librería estándar *(stdlib).*

Todo el sistema esta programado usando el lenguaje de programación C y compilado bajo el sistema operativo GNU/Linux.

Diseño del modelo de datos

El sistema esta diagramado de la siguiente forma:



El sistema usa como dato de entrada la expresión algebraica ingresada por el usuario, internamente se usaran pilas y listas para manejar tanto los operadores como los operandos, y como dato de salida se imprimirá el resultado por pantalla.

Al invocar el programa se le pedirá al usuario que ingrese la expresión algebraica que quiere resolver, si se invoca con el parámetro -h se mostrara una ayuda de como usar el sistema. A continuación cada elemento de la expresión sera apilado en una pila, exceptuando los espacios en blanco. Aquí el programa puede reportar las siguientes situaciones de error:

1. Si hay exceso o falta de paréntesis.
2. Si alguno de los operandos/operadores es desconocido.

Luego se aplica un algoritmo que es una variación del Shunting Yard (*https://es.wikipedia.org/wiki/Algoritmo_shunting_yard*) y que funciona de la siguiente manera:

- Mientras haya elementos en la expresión
  - Si es un numero lo apilo en la pila auxiliar.
  - Si es un paréntesis que cierra lo apilo en la pila auxiliar.
  - Si es un operador:
    - Si es un paréntesis que abre, y en el tope de la pila auxiliar hay un numero:
      - Desapilo de la pila auxiliar.
      - Si en la pila auxiliar hay un paréntesis que cierra:
        - Desapilo el paréntesis que cierra.
        - Apilo el numero.
      - Si no es un paréntesis que cierra:
        - Guardo el operador.
        - Mientras no desapilé un paréntesis que cierra de la pila auxiliar:
          - Guardo el numero en una lista.
  - Llamo al método que corresponda dependiendo del operador que encontré con la lista de números.
  - Apilo el resultado en la pila auxiliar.
  - Desapilo de la pila.
- El resultado de la expresión esta en el tope de la pila auxiliar.

Especificación técnica:

Evaluar:

`int es_digito(char carácter);`
Evaluá si el carácter pasado como parámetro es un dígito o no.
Toma como parámetro el carácter a evaluar.
Retorna 1 si es un dígito y 0 en caso contrario.

`int carácter_valido(char carácter);`
Evaluá si el carácter es un carácter valido, es decir si es “+”, “-”, “*”, “/”, “(” o “)”.
Toma como parámetro el carácter a evaluar.
Retorna 1 si es un carácter valido y 0 en caso contrario.

`char* obtener_cadena(char carácter);`
Transforma el carácter pasado como parámetro en su representación como cadena de caracteres.
Toma como parámetro el carácter a transformar.
Retorna un string con la representación del parámetro.

`int suma(lista_t lista);`

Suma todos los elementos en la lista pasada como parámetro.
Toma como parametro la lista con los operando.
Retorna la suma de todos los operandos.

`int producto(lista_t lista);`
Multiplica todos los elementos en la lista pasada como parámetro.
Toma como parámetro la lista con los operando.
Retorna el producto de todos los operandos.

`int resta(lista_t lista);`
Resta todos los elementos en la lista pasada como parámetro.
Toma como parámetro la lista con los operando.
Retorna la resta de todos los operandos.

`int division(lista_t lista);`
Divide todos los elementos en la lista pasada como parámetro.
Toma como parámetro la lista con los operando.
Retorna la división de todos los operandos.

`void desapilar_y_evaluar(pila_t pila);`
Toma una pila con todos los elementos de la expresión algebraica y calcula el resultado de dicha expresión.
Toma como parámetro la pila con la expresión aritmética.

`void apilar_cadena(char* cadena);`
Crea una pila con todos los elementos de la expresión algebraica.
Toma como parámetro la cadena de caracteres con la expresión.

`void mostrar_ayuda();`
Imprime por pantalla la ayuda del programa.

`int main(int argc, char** argv);`
Método del programa principal.

Pila:

`pila_t  pila_crear();`
Retorna una pila nueva vacía.

`char* tope(pila_t pila);`
Retorna el string que se encuentra en el tope de la pila. Si la
pila se encuentra vaciá, aborta su ejecución con exit status PIL_VACIA.
Toma como parámetro una pila a la cual pedirle el tope.
Retorna un puntero a carácter representando el tope de la pila.

`char* desapilar(pila_t* pila);`
Elimina el string que se encuentra en el tope de la pila y lo retorna. Si la
pila se encuentra vacia, aborta su ejecución con exit status PIL_VACIA
Toma como parámetro una pila de la cual se quiere desapilar.
Retorna un puntero a carácter representando lo desapilado.

`int apilar(pila_t* pila, char* str);`
Inserta el string str en el tope de la pila. Retorna verdadero si la inserción fue exitosa, falso en caso contrario. Si la pila no se encuentra inicializada, finaliza la ejecución con exit status PIL_NO_INI.
Tiene como parámetro un puntero a pila y un string que es lo que se va a apilar.
Retorna 1 si la operación fue completada con éxito.

`int pila_vacia(pila_t pila);`
Retorna verdadero si la pila esta vacia, falso en caso contrario. Si la pila no se encuentra inicializada, finaliza la ejecución con exit status PIL_NO_INI.
Toma como parametro la pila.
Retorna 1 si la pila esta vacia 0 caso contrario.

Lista:

`lista_t lista_crear();`
Retorna una nueva lista vaciá.

`int lista_insertar(lista_t lista, unsigned int pos, int elem);`
Inserto un elemento en una posición pasada como parámetro.
Si la posición es mayor a la cantidad de elementos, finaliza la ejecución con error LST_POS_INV
Si la posición es igual a la cantidad de elementos, inserto al final.
Tiene como parámetro un puntero a una struct lista donde se quiera insertar, un entero sin signo que es la posición donde se quiere insertar y un entero que es el elemento a insertar.
Retorna 1 si la operación se completo con éxito o 0 caso contrario.

`int lista_eliminar(lista_t lista, unsigned int pos);`
Elimina un elemento de la lista según la posición pasada como parámetro
Si la posición pasada es mayor que la cantidad de elementos, finaliza la ejecución con error LST_POS_INV
Tiene como parámetro un puntero a struct lista donde se quiera eliminar un elemento y la posición de la lista que se quiere eliminar.
Retorna 1 si la operación se completo exitosamente.

`int lista_cantidad(lista_t lista);`
Retorna la cantidad de elementos de la lista.
Si la lista no esta inicializada finaliza la ejecución con error LST_NO_INI.
Tiene como parámetro un puntero a un struct lista.
Retorna la cantidad de elementos de la lista.

`int lista_obtener(lista_t lista, unsigned int pos);`
Retorna el elemento en la posición pasada como parámetro.
Si la posición es mayor a la cantidad de elementos de la lista finaliza la ejecucion con error LST_POS_INV .
Tiene como parametro un puntero a un struct lista y un entero que representa la posicion del elemento que se quiere obtener.
Retorna el elemento en la posicion pos.

`int lista_adjuntar(lista_t lista, int elem);`
Agrego un elemento al final de la lista
Si la lista no esta inicializada finaliza la ejecucion con error LST_NO_INI.
Tiene como parametro el puntero a la lista y un entero elem que es el elemento a adjuntar.
Retorna 1 si la operación se completo satisfactoriamente.

`int lista_destruir(lista_t* lista);`
Libero todo el espacio ocupado por la lista, incluyendo las celdas.
Tiene como parametro un puntero a un puntero a lista.
Retorna 1 si la operación se completo satisfactoriamente.

Codigos de error:

Exit Status	***Valor Numerico***	***Significado***
EXITO	EXIT_SUCCESS	Terminación exitosa.
EXP_MALF	2	Expresion malformada,falta o exceso de “( )”.
LST_NO_INI	3	Lista sin inicializar.
LST_POS_INV	4	Intento de acceso a posición invalida en la lista.
OPND_DEMAS	5	Demasiados operandos para el operador.
OPND_INSUF	6	Insuficientes operandos para el operador.
OPND_INV	7	Operando invalido.
OPRD_INV	8	Operador desconocido o invalido.
PIL_NO_INI	9	Pila no inicializada.
PIL_VACIA	10	Pila vacia.

Conclusiones

Extendimos la funcionalidad del programa original al agregar como correcto las expresiones de tipo (operando), ej. (123) o ((((123)))) o (+ (123) 123).

El largo máximo de la expresión es de 256 caracteres.