Algoritmo del método evaluar:

Este es el método principal del archivo “evaluar”. En primer lugar, testea si los paréntesis de la expresión están bien balanceados mediante una operación auxiliar. Luego de esto se encarga de recorrer cada uno de los caracteres de la expresión e ir insertándolos en una pila hasta llegar al final de la expresión. Si en la lectura de caracteres encuentra un “)” (paréntesis que cierra), invoca un método auxiliar enviando la pila actual en la cual se estaban agregando los caracteres en forma de string de la expresión que retorna el resultado de evaluar la expresión entre este paréntesis que cierra encontrado y el paréntesis que abre más cercano en la pila, y retorna el resultado de evaluar la operación ubicada entre ambos paréntesis como una variable de tipo string. Además, se incrementa el la posición actual en la expresión (con el fin de avanzar de posición salteando el paréntesis que cierra encontrado anteriormente) y se apila el resultado obtenido en forma de string de la operación evaluar_aux en la pila que se tiene como variable denominada “pila”.
Al terminar de leer la expresión, se tendrá contenido en la pila el resultado de haber evaluado la expresión completamente y se retorna este string luego de desapilarlo de la pila.

```
char* evaluar (char[] exp, int size){
        Inicializo una pila nueva denominada “pila”
        Si los paréntesis no están balanceados ──→ // Verificado mediante un método auxiliar llamado comprobar_parentesis_balanceados (char [] exp, int size)
                Termino la ejecución debido al ingreso de una expresión incorrectamente formada
                en cuanto a paréntesis faltantes o incorrectamente distribuidos.
        sino
                int i← 0                //Creo e inicializo una variable entera “i” en 0
                char*c res              //Creo una cadena de caracteres (String) llamada “res”
                Mientras i sea menor a la cantidad de caracteres de la expresión (size)
                        Si no leo un paréntesis que cierra en la posición “i” de la expresión
                                apilo el carácter de la posición “i” enviado como String en “pila”
                        sino
                                Aumento i en 1 ──→ //Salteo la posición de la expresión donde se encuentra el paréntesis que cierra
                                res ← evaluar_aux (pila) ←── //Se explica el método evaluar_aux (pila) a continuación de este algoritmo
                                Apilo el resultado en la pila
                        Si la “i” es menor a la cantidad de caracteres de la expresión
                                Aumento la variable entera “i” en 1
        Retorno la variable de tipo string ubicada en el tope de la pila al desapilar de ella, que hace
        referencia al resultado de evaluar la expresión en su totalidad.
}
```

Algoritmo del método evaluar_aux

Este método recibe una pila se string y se encarga de evaluar la expresión contenida entre el tope de la pila y el primer paréntesis que abre a encontrarse dentro de la pila enviada como parámetro.

```
char* evaluar_aux (pila_t pila){
        Declaro un string denominado aux;
        Declaro una nueva pila denominada pila_aux y la inicializo
        Declaro una variable entera cant_operadores y la inicializo con 0
        Declaro una lista llamada "lista" y la inicializo
        Declaro un string denominado operador
        Declaro una variable entera denominada "variable"
        Mientras no leo un "(" en "pila"
                Desapilo de pila y asigno el string retornado a aux
                Si aux es un operando
                        apilo en "pila_aux"
                sino
                        Si aux es un operador
                                Si la cant_operadores es igual a 0
                                        Asigno el valor de aux a la variable operador
                                        Le asigno 1 a cant_operadores
                                sino
                                        Termino la ejecución por exceso de operadores
                        sino
                                Si aux es un espacio
                                        Si la pila no está vacía
                                                Convierto los string de la pila en entero y asigno
                                                el entero obtenido a "variable"
                                                Agrego "variable" al final de la lista
                                                Le asigno nulo a "variable"
                                sino
                                        Termino la ejecución por tratarse de un carácter inválido
        Desapilo de "pila"      //Elimino el paréntesis que abre, "("
        Opero con la lista de variables enteras "lista" y el string del operador llamado "operador" y
        retorna el resultado como un string
}
```

//Mediante un método auxiliar explicado más adelante denominado convertir_string_a_int (pila_t pila)

//A través de un método auxiliar llamado operar (lista_t lista, char* operador)

Algoritmo de comprobar_parentesis_balanceados:

```
int comprobar_parentesis_balanceados (char [] expresion, int size){
        Declaro una variable booleana “estan_balanceados” y la inicializo con verdadero
        Declaro e inicializo una nueva pila denominada “pila”
        Mientras no llegue al final de expresion y estan_balanceados avanzo en la posición de la
        expresión
                Si leo un “(“ en expresion
                        apilo el paréntesis en “pila
                sino
                        si leo un “)” en expresion
                                si “pila” está vacía
                                        asigno falso a “estan_balanceados”
                                sino
                                        Desapilo de pila
        Si la pila no está vacía
                Asigno falso a “estan_balanceados”
        Retorno “estan_balanceados”
}
```

Algoritmo de convertir_string_a_int:

Este método recibe una pila como parámetro, donde el tope inicial es el primer digito (expresado en string) del entero a devolver. Por esto, es necesario tener una variable que multiplique el digito que se está leyendo por un múltiplo de 10 que indique la posición en la cual se debe insertar el dígito para formar el número a retornar.

```
int convertir_string_a_int (pila_t pila){
        Declaro una variable entera denominada “operando” e inicializo con 0
        Declaro una variable string llamada “tope”
        Declaro una variable entera “multiplo” y la inicializo con 1
        Mientras la pila no está vacía
                Desapilo de pila y asigno el string obtenido a “tope”
                Comparo “tope” con cada entero en forma de string y dependiendo el entero que
                sea lo multiplico por “multiplo” y al resultado lo sumo a “operando”
                Multiplico “multiplo” por 10
        Retorno “operando”
}
```

Algoritmo de operar:

Esta método recibe una lista de enteros y un operador, y dependiendo del operador que sea realiza determina operación.

```
char* operar (lista_t lista, char* operador){
        Declaro un entero “resultado” y la inicializo con 0
        Si la cantidad de elementos de la lista es menor que 2
                Termino la ejecución debido a que no hay suficientes operadores
        sino
                si el operador es un “+”
                        A resultado le asigno suma (lista)
                sino
                        si el operando es un “*”
                                A resultado le asigno multiplicacion (lista)
                        sino
                                Si la cantidad de elementos de la lista es igual a dos
                                        Si el operador es un”/”
                                                A resultado le asigno división (lista)
                                        sino
                                                        Si el operador es un “-”
                                                                A resultado le asigno resta (lista)
                                sino
                                        Termino la ejecución por exceso de operandos
        Retorno “resultado” como un string
}
```

Algoritmos de las operaciones aritméticas:

int suma (lista_t operandos) {
    Declaro un entero “resultado” y lo inicializo con 0
    Para cada posición de “operandos”
        Sumo a resultado el entero de la posición actual de la operandos”
    Retorno “resultado
}

int resta (lista_t operandos) {
    Retorno el resultado de efectuar la resta entre el operando de la segunda posición de “operandos” y el operando de la primera posición de “operandos”
}

int multiplicacion (lista_t operandos) {
    Declaro un entero “resultado” y lo inicializo con 1
    Para cada posición de “operandos”
        Multiplico a resultado el entero de la posición actual de “operandos”
    Retorno “resultado”
}

int división (lista_t operandos) {
    Declaro un entero “x” y lo inicializo con el entero de la segunda posición de la “operandos”
    Si “x” vale 0 entonces retorno x
    sino
        retorno el resultado de efectuar la división del primer elemento de “operandos” por “x”
}