Universidad Nacional de San Agustín
Facultad de Ingeniería de Producción y Servicios
Escuela Profesional de Ingeniería de Sistemas

Tecnología de Objetos

Alumno: Mestas Escarcena, Carlos Alberto
Grupo: B

Laboratorio 2

El desarrollo del proyecto se puede ubicar en el siguiente repositorio de GitHub.

En el desarrollo se agregaron dos clases más para trabajar con el árbol binario de expresión, un *BETNode* que implementará los nodos del árbol con el cual vamos a trabajar y *BinaryExpressionTree* que se encargará de trabajar con un nodo raíz y también de los distintos ordenamiento de cada operador y número.

También se agregaron algunos método en las clases ya presentadas:

```cpp
// TextProcesor.cpp Metodos de la clase procesador de texto
// Curso: Tecnologia de Objetos
// Laboratorio: B
// Autor: Carlos Alberto Mestas Escarcena
#include "TextProcesor.h"
.
.
.

std::vector<uint16_t> TextProcesor::obtainNumbers(std::string _text){
    std::vector<uint16_t> vectorOfNumbers;
    uint8_t i = 0; uint8_t size = _text.size();
    while(i < size){
        std::string tmpNumber = "";
        while(i < size && (_text.at(i) != '+') && ( _text.at(i) != '*')){

            tmpNumber += _text.at(i);
            i++;
        }
        vectorOfNumbers.push_back(std::stoi(tmpNumber));
        i++;
    }
    return vectorOfNumbers;
}

std::vector<char> TextProcesor::obtainOperations(std::string _text){
    std::vector<char> vectorOfOperations;
    uint8_t i = 0; uint8_t size = _text.size();
    while(i < size){
        if(_text.at(i) == '+')
            vectorOfOperations.push_back(_text.at(i));
        else if(_text.at(i) == '*')
            vectorOfOperations.push_back(_text.at(i));
        i++;
    }
    return vectorOfOperations;
}
```

Se modificaron los métodos para obtener los números y las operaciones, ya que ahora tenemos una operación extra.

```
// Operation.cpp Metodos de la clase operacion
// Curso: Tecnologia de Objetos
// Laboratorio: B
// Autor: Carlos Alberto Mestas Escarcena
.
.
.

std::uint16_t Operation::multiplication(std::uint16_t _number1, std::uint16_t _number2){
    std::uint16_t answer = _number1 * _number2;
    return answer;
}
```

Solamente se agregó el método para realizar una multiplicación.

```
// Calculator.h Cabecera de la clase Binary Tree Expresion Node
// Curso: Tecnologia de Objetos
// Laboratorio: B
// Autor: Carlos Alberto Mestas Escarcena

#ifndef BETNODE_H
#define BETNODE_H

#include <iostream>
#include <stddef.h>

class BETNode{
private:
    std::string data;          // Informacion que tendra el nodo (simbolo o numero)

public:
    BETNode *leftNode;  // Nodo que se ubicara a su izquierda
    BETNode *rightNode; // Nodo que se ubicara a su derecha

    BETNode(std::string _data);    // Contructor principal
    BETNode getLeftNode();  // Getter para obtener el nodo de la izquierda
    BETNode getRightNode(); // Getter para obtener el nodo de la derecha
    void setLeftNode(BETNode *_betNode);     // Setter para agregar un nodo a la izquierda
    void setRightNode(BETNode *_betNode);    // Setter para agregar un nodo a la derecha
    std::string getData();
};

#endif // BETNODE_H
```

```
// Calculator.cpp Metodos de la clase Binary Tree Expresion Node
// Curso: Tecnologia de Objetos
// Laboratorio: B
// Autor: Carlos Alberto Mestas Escarcena

#include "BETNode.h"

// Contructor por defecto con nodos vacios a los costados
BETNode::BETNode(std::string _data){
    this->data = _data;
```

```
11      this->leftNode = NULL;
12      this->rightNode = NULL;
13  }
14
15  BETNode BETNode::getLeftNode(){
16      return *leftNode;
17  }
18
19  BETNode BETNode::getRightNode(){
20      return *rightNode;
21  }
22
23  void BETNode::setLeftNode(BETNode *_betNode){
24      this->leftNode = _betNode;
25  }
26  void BETNode::setRightNode(BETNode *_betNode){
27      this->rightNode = _betNode;
28  }
29
30  std::string BETNode::getData(){
31      return this->data;
32  }
```

Tenemos la nueva clase BETNode, donde podemos observar que se cada nodo tiene dos nodos hijos, uno a la izquierda y otro a la derecha, un constructor por defecto con ambos hijos nulos y los respectivos setters y getters.

```
1   // Calculator.h Cabecera de la clase Binary Tree Expresion
2   // Curso: Tecnologia de Objetos
3   // Laboratorio: B
4   // Autor: Carlos Alberto Mestas Escarcena
5
6   #ifndef BINARYEXPRESSIONTREE_H
7   #define BINARYEXPRESSIONTREE_H
8
9   #include "BETNode.h"
10  #include "TextProcesor.h"
11  #include "Operation.h"
12
13  #include <stddef.h>
14  #include <cinttypes>
15
16  class BinaryExpressionTree{
17  private:
18      BETNode *rootNode; // Nodo raiz
19  public:
20      BinaryExpressionTree();
21      void putNode(BETNode *_node); // Metodo para ingresar un nodo
22      void putTwoNode(BETNode *_nodeOperation, BETNode *_nodeNumber); // Metodo para ingresar
        ↪  tanto una operacion y un nodo
23      void putLF(BETNode *_nodeFather, std::string _num1, std::string _num2); // Metodo para
        ↪  ingresar dos nodos con una operacion
24      void enterPlaneText(std::string _text); // Metodo para ingresar el texto plano
25      void operate(); // Metodo para realizar la suma (Aun sin terminar)
26      void operateAux(uint16_t _ans, BETNode *_node); // Metodo que ayuda a realizar la suma
        ↪  (Aun sin terminar)
27      void printTree(); // Metodo para imprimir el arbol
28      void printTreeAux(BETNode *_node); // Metodo que ayuda a imprimir el arbol
```

```
};

#endif // BINARYEXPRESSIONTREE_H
```

```
// Calculator.cpp Metodos de la clase Binary Tree Expresion
// Curso: Tecnologia de Objetos
// Laboratorio: B
// Autor: Carlos Alberto Mestas Escarcena

#include "BinaryExpressionTree.h"

/*
 * Constructor con la raiz nula
 */
BinaryExpressionTree::BinaryExpressionTree(){
    this->rootNode = NULL;
}
/*
 * Metodo para colocar un nodo ya sea raiz o no
 */
void BinaryExpressionTree::putNode(BETNode *_node){
    // En el caso de que el nodo sea nulo lo ingresamos en ese lugar
    if(rootNode == NULL){
        rootNode = new BETNode(_node->getData());
    }
    // De otro caso lo ingresamos tanto en la derecha o la izquierda
    else{
        // En el caso de que el nodo a la izquierda este vacio
        if(this->rootNode->leftNode == NULL){
            std::string aux = rootNode->getData();
            std::cout << "aux \t" << aux << std::endl;
            std::cout << "aux2\t" << _node->getData() << std::endl;
            rootNode = new BETNode(_node->getData());
            rootNode->setLeftNode(new BETNode(aux));

        }
        // En el caso de que el nodo de la derecha este vacio
        else if (this->rootNode->rightNode == NULL) {
            std::cout << "test derecha" << std::endl;
            std::string aux = _node->getData();
            std::cout << "aux3\t" << aux << std::endl;
            rootNode->setRightNode(new BETNode(aux));
        }
    }
}

/*
 * Metodo para colocar dos nodos
 * En este caso un nodo que contenga a la operacion
 * Y otro nodo con el correspondiente numero
 */
void BinaryExpressionTree::putTwoNode(BETNode *_nodeOperation, BETNode *_nodeNumber){
    // En el caso de que tengamos una suma antes y ahora una multiplicacion
    // En este caso tenemos que colocar una rotacion para que la multiplicacion
    // Sea la que tenga prioridad
    if(rootNode->getData() == "+" && _nodeOperation->getData() == "*"){
        std::cout << "Test dato " << std::endl;
        std::string aux = rootNode->getRightNode().getData();
```

```
55          std::cout << "info root " << aux <<std::endl;
56          rootNode->setRightNode(new BETNode(_nodeOperation->getData()));
57          putLF(rootNode->rightNode, aux, _nodeNumber->getData());
58          std::cout << "info root derecha " << rootNode->getRightNode().getData()
            ↪  <<std::endl;
59          std::cout << "info a colocar " << aux << "\t "<< _nodeNumber->getData()
            ↪  <<std::endl;
60          std::cout << "info root derecha izq" <<
            ↪  rootNode->getRightNode().getLeftNode().getData() << std::endl;
61          std::cout << "info root derecha der" <<
            ↪  rootNode->getRightNode().getRightNode().getData() <<std::endl;
62
63      }
64      // En caso de ser solo dos sumas entonces se agrega sin problema
65      else if(rootNode->getData() == "+" && _nodeOperation->getData() == "+"){
66           BETNode *tmp = rootNode;
67           rootNode = new BETNode(_nodeOperation->getData());
68           rootNode->setLeftNode(tmp);
69           rootNode->setRightNode(new BETNode(_nodeNumber->getData()));
70      }
71      // Lo tratamos de la misma manera que suma suma ya que la suma tiene menos prioridad
72      else if(rootNode->getData() == "*" && _nodeOperation->getData() == "+"){
73           BETNode *tmp = rootNode;
74           rootNode = new BETNode(_nodeOperation->getData());
75           rootNode->setLeftNode(tmp);
76           rootNode->setRightNode(new BETNode(_nodeNumber->getData()));
77      }
78      // Lo tratamos como suma suma ya que dos multiplicaciones tienen la misma prioridad
79      else if(rootNode->getData() == "*" && _nodeOperation->getData() == "*"){
80           BETNode *tmp = rootNode;
81           rootNode = new BETNode(_nodeOperation->getData());
82           rootNode->setLeftNode(tmp);
83           rootNode->setRightNode(new BETNode(_nodeNumber->getData()));
84      }
85
86  }
87
88  /*
89   * Metodo para agregar a ambos lados de un nodo de operacion los numeros
90   */
91  void BinaryExpressionTree::putLF(BETNode *_nodeFather, std::string _num1, std::string
    ↪  _num2){
92      _nodeFather->setLeftNode(new BETNode(_num1));
93      _nodeFather->setRightNode(new BETNode(_num2));
94
95  }
96  /*
97   * Metodo para imprimir el arbol que estamos generando
98   */
99  void BinaryExpressionTree::printTree(){
100     printTreeAux(rootNode);
101 }
102
103 /*
104  * Metodo que ayuda al metodo de imprimir
105  * En este caso podemos observar que se realizar una impresion de padre, hijo izquierda e
    ↪  hijo derecha
106  */
107 void BinaryExpressionTree::printTreeAux(BETNode *_node){
108     if(_node != NULL){
```

```
109         std::cout << _node->getData() << " Test" << std::endl;
110         printTreeAux(_node->leftNode);
111         printTreeAux(_node->rightNode);
112 
113     }
114 }
115 
116 /*
117  * Metodo para realizar la suma
118  * AUN NO IMPLEMENTADO TOTALMENTE
119  */
120 void BinaryExpressionTree::operate(){
121     uint16_t ans = 0;
122     operateAux(ans, rootNode);
123 }
124 
125 /*
126  * Metodo que ayuda a realizar la suma
127  * AUN NO IMPLEMENTADO TOTALMENTE
128  */
129 void BinaryExpressionTree::operateAux(uint16_t _ans,BETNode *_node){
130     if(_node->leftNode != NULL && _node->rightNode != NULL){
131         if(_node->getData() == "+"){
132             _ans =
                ↪  Operation::adition(std::stoi(_node->leftNode->getData()),std::stoi(_node->rightNode->ge
133         }
134     }
135 
136 }
137 
138 /*
139  * Metodo que trata las operaciones que se ingresan como un solo dato
140  */
141 void BinaryExpressionTree::enterPlaneText(std::string _text){
142     std::vector<uint16_t> vectorOfNumbers = TextProcesor::obtainNumbers(_text);
143     std::vector<char> vectorOfOperations = TextProcesor::obtainOperations(_text);
144 
145     TextProcesor::printNumbers(vectorOfNumbers);
146     TextProcesor::printOperations(vectorOfOperations);
147 
148     uint8_t i2 = 0;
149     uint8_t size = vectorOfOperations.size();
150     std::string aux = std::to_string(vectorOfNumbers.at(0));
151 
152     // Ingresamos el nodo raiz
153     BETNode *tmp = new BETNode(aux);
154     BinaryExpressionTree::putNode(tmp);
155 
156     // Ingresamos los dos primeros nodos
157     aux = "";
158     aux.push_back(vectorOfOperations.at(i2));
159     std::cout <<  aux << "Put op 1 \t" << std::endl;
160     BETNode *tmp2 = new BETNode(aux);
161     putNode(tmp2);
162 
163     aux = "";
164     aux = std::to_string(vectorOfNumbers.at(i2 + 1));
165     std::cout << aux << "Put num 2 \t" <<std::endl;
166     BETNode *tmp3 = new BETNode(aux);
167     putNode(tmp3);
```

```
168
169     i2++;
170
171     std::string aux2;
172
173     // Ingresamos los nodos de dos en dos, operacion con su numero
174     while (i2 < size){
175         aux = "";
176         aux.push_back(vectorOfOperations.at(i2));
177
178         aux2 ="";
179         aux2 = std::to_string(vectorOfNumbers.at(i2 + 1));
180
181         std::cout << "Put num 3 \t" << aux << "\t" << aux2 << std::endl;
182         putTwoNode(new BETNode(aux), new BETNode(aux2));
183
184         i2++;
185     }
186 }
```

Tenemos la otra nueva clase que se encargará de trabajar como el árbol binario de expresión, tiene su constructor que también inicializa su único nodo raíz como nulo, y tiene distintos métodos para la impresión y para el ingreso de nodos.

```
1  // main.cpp Clase principal para mostrar el funcionamiento de la calculadora
2  // Curso: Tecnologia de Objetos
3  // Laboratorio: B
4  // Autor: Carlos Alberto Mestas Escarcena
5
6  #include "Calculator.h"
7  #include "BinaryExpressionTree.h"
8
9  int main(){
10     BinaryExpressionTree myTree1;
11     BinaryExpressionTree myTree2;
12     BinaryExpressionTree myTree3;
13     BinaryExpressionTree myTree4;
14
15     std::string test1 = "33+1";
16     std::string test2 = "2*5";
17     std::string test3 = "12+4*12";
18     std::string test4 = "34*3+1*90";
19
20     myTree1.enterPlaneText(test1);
21     myTree2.enterPlaneText(test2);
22     myTree3.enterPlaneText(test3);
23     myTree4.enterPlaneText(test4);
24
25     myTree1.printTree();
26     myTree2.printTree();
27     myTree3.printTree();
28     myTree4.printTree();
29
30     /*
31     std::string test1 = "12+34";
32     std::string test2 = "42+1+34";
33     std::string test3 = "1+2+3+4+5+6";
34     std::string test4 = "1123+451+12+1+10+100+45+489+12";
```

```
35      Calculator myCalculator;
36      myCalculator.operate(test1);
37      myCalculator.operate(test2);
38      myCalculator.operate(test3);
39      myCalculator.operate(test4);
40      */
41      return 0;
42  }
```

La clase main nos va a permitir corroborar el trabajo de lo implementado anteriormente, cabe resaltar que la impresión del árbol se realiza de la forma de nodo padre, nodo hijo de la izquierda y nodo hijo de la derecha.

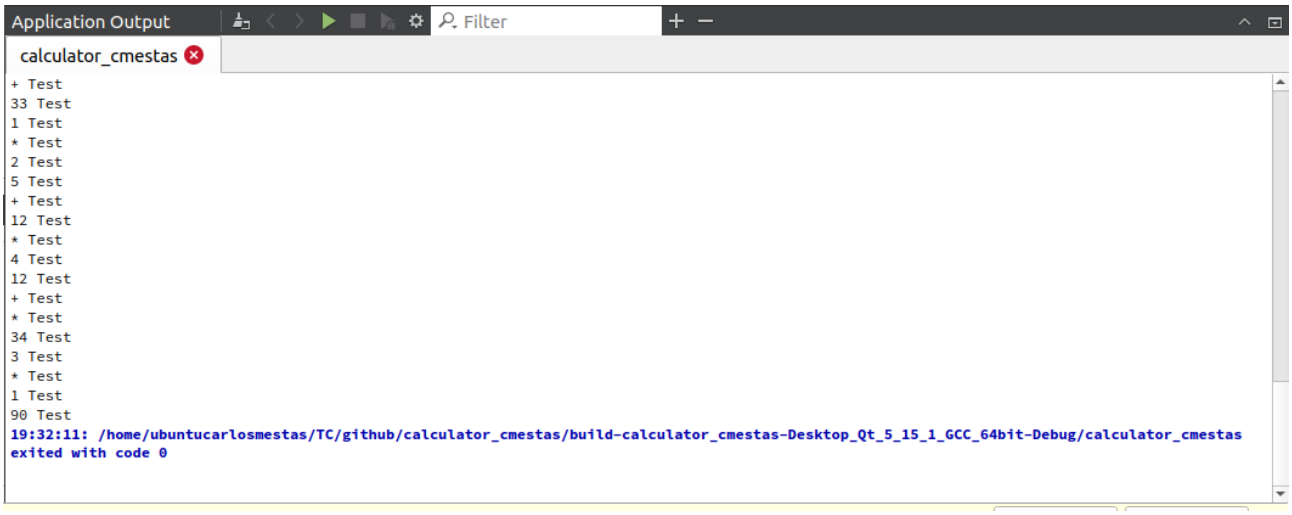

Figura 1: Ejemplo del ingreso de datos en el árbol