REPORT ASSIGNMENT HÌNH THỨC 2

Họ tên: Nguyễn Anh Tuấn
MSSV: 20200400

Requirement:
Cài đặt cây nhị phân để tính toán giá trị biểu thức

Bài làm:

Code và giải thích từng hàm:

1. Khai báo thư viện:

```
#include <queue>
#include <iostream>
#include <string>
#include <regex>

using namespace std;
```

2. Khởi tạo cấu trúc cây với data là dạng string vì có thể node mang data là 1 số hoặc 1 phép tính + - / *:

```
struct Node {
    string data;
    Node* left;
    Node* right;
};

struct Tree {
    Node* root;
};
```

3. Tạo node từ giá trị đi kèm :

```
Node* getNode(string value){
    Node* node = new Node;
    node->data = value;
    node->left = NULL;
    node->right = NULL;
    return node;
}
```

4. Hàm tạo cây từ n phần tử nhập vào:

```
void createTree(Tree* tree, int n){
    cout << "Nhap gia tri tung phan tu: \t";
    for ( int i = 0 ; i < n ; i++){
        int tmp;
        cin >> tmp;
        addNode(*tree, tmp);
    }
}
```

5. Hàm thêm node vào cây dùng hàng đợi để thêm và kiểm tra từ trên xuống dưới từ trái qua phải liệu rằng node đã có con hay chưa và nếu chưa thì đã có con trái hay con phải chưa để thêm node mới vào:

```
void addNode(Tree& tree, string n){
    Node* node = getNode(n);
    if ( tree.root == NULL ){
        tree.root = node;
        return;
    }
    queue<Node*> q;
    q.push(tree.root);
    while (!q.empty()){
        Node* tmp = q.front(); q.pop();
        if ( tmp->left == NULL ){
            tmp->left = node; return;
        } else {
            q.push(tmp->left);
        }
        if ( tmp->right == NULL ){
            tmp->right = node; return;
        } else {
            q.push(tmp->right);
        }
    }
}
```

6. Hàm khởi tạo cây rỗng:

```
void init(Node* root){
    root = NULL;
}
```

7. Hàm duyệt cây theo thứ tự Left Node Right (InOrder):

```
void traverse(Node* root){
    if ( root == NULL ) return;
    traverse(root->left);
    cout << root->data << " ";
    traverse(root->right);
}
```

8. Hàm kiểm tra tất cả ký tự trong chuỗi liệu có phải là số (number) hết hay không :

```
bool isNumber(string& s) {
    regex e("^-?[0-9]+$");
    return regex_match(s, e);
}
```

Sử dụng thư viện regex, trong đó s là chuỗi mình đang muốn xét, e là đối tượng biểu thức được tạo bằng mẫu ***^-?[0-9]+$***
, trong đó:

- Dấu **^** và ***$*** là 2 dấu neo xác định tương ứng mở đầu và kết thúc của chuỗi đó nghĩa là phần bên trong 2 dấu là phải khớp với toàn bộ chuỗi.
- Dấu ***-?*** cho biết có thể có hoặc không 1 dấu âm trong chuỗi -> để nhận số âm hoặc dương.
- Các ký tự ***[0-9]*** chỉ khớp với các số từ 0 đến 9
- Dấu ***+*** cho biết phải có ít nhất 1 hoặc nhiều chuỗi chữ hoặc số phải có trong chuỗi xét, trong trường hợp này ***[0-9]+*** là xét chuỗi phải có 1 hoặc nhiều số trong chuỗi và chỉ những ký tự số 0->9 được cho phép.

⇨ Cả chuỗi phải có ít nhất 1 hoặc nhiều ký tự từ 0->9 và bắt đầu và kết thúc bằng chính chúng, nếu như bằng ký tự khác thì chuỗi xét sẽ không thoã

Hàm ***regex_match*** sẽ với các thông số chuỗi cần xét và pattern xét chuỗi sẽ trả về gái trị true nếu chuỗi thoả pattern, false nếu ngược lại; do đó, hàm isNumber sẽ trả về giá trị true nếu cả chuỗi đều mang kiểu số.

9. Hàm kiểm tra chuỗi có phải là các phép ***+ / * -*** hay không, nếu chuỗi bằng 1 trong 4 thì trả về true:

```
bool isOperator(string& s) {
    return s == "+" || s == "-" || s == "*" || s == "/";
}
```

10. Hàm tạo cây n node với data nhập từ bàn phím, mỗi lần nhập vào biến tmp thì sẽ check liệu biến đó có hợp lệ là số hoặc là các phép toán hay không, nếu không sẽ cho nhập lại đến khi nào hợp lệ thì thôi:

```
void createTree(Tree* tree, int n){
    cout << "Nhap gia tri tung phan tu: \n";
    for ( int i = 0 ; i < n ; i++){
        string tmp;
        cin >> tmp;
        while (!isOperator(tmp) && !isNumber(tmp)) {
            cout << "Nhap lap dang number hoac dang +*-/: ";
            cin >> tmp;
        }
        addNode(*tree, tmp);
    }
}
```

11. Hàm tạo biểu thức từ cây **<u>cùng với</u>** các dấu mở ngoặc "(" và đóng ngoặc ")" đúng vị trí bằng cách duyệt InOrder (Left Node Right):

```
void expressionBuilder(Node* root, string& str) {
    if (root == nullptr) return;
    if (root->left != nullptr) {
        str += "(";
        expressionBuilder(root->left, str);
    }
    str += root->data;
    if (root->right != nullptr) {
        expressionBuilder(root->right, str);
        str += ")";
    }
}
```

12. Hàm tạo chuỗi biểu thức từ cây **<u>không có</u>** các dấu mở ngoặc "(" và đóng ngoặc ")" thay vào đó là dấu _ đúng vị trí bằng cách duyệt InOrder (Left Node Right) để kiểm tra liệu biểu thức có hợp lệ hay không:

```
void stringBuilder(Node* root, string& str) {
    if (root == nullptr) return;
    if (root->left != nullptr) stringBuilder(root->left, str);
    str += root->data + "_";
    if (root->right != nullptr) stringBuilder(root->right, str);
}
```

13.Hàm kiểm tra chuỗi có hợp lệ hay không:

```
bool check(string& s) {
    int n = s.length();
    int i = 0;
    while (i < n){
        string curr, next;
        bool checkFirst = false;
        while ( s[i] != '_' && i < n) {
            if ( i == 0 ) checkFirst = true;
            curr += s[i];
            i++;
        }
        if ( checkFirst && isOperator(curr)) return false;
        if ( i == n-1 ) break;
        i++;
        while (s[i] != '_' && i < n){
            next += s[i];
            i++;
        }
        if ((isNumber(curr) && isNumber(next)) || (isOperator(curr) && isOperator(next))){
            return false;
        }
        i++;
    }
    return true;
}
```

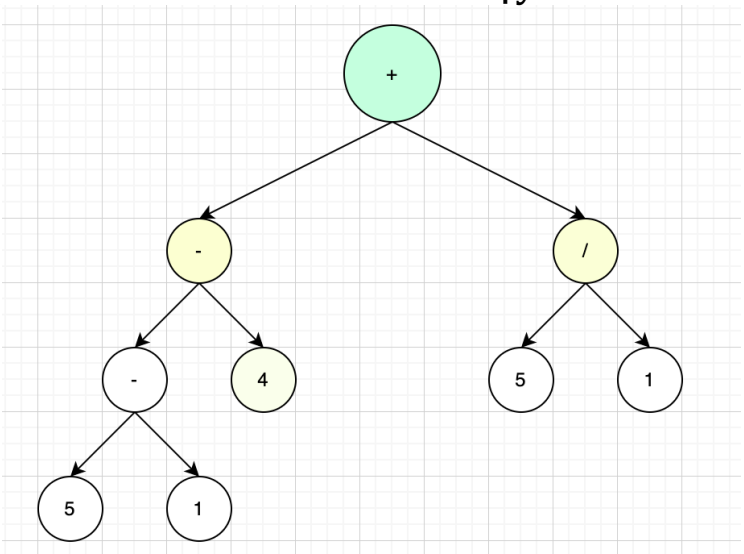
Logic triển khai hàm:

Xét cây trong hình là **<u>cây biểu thức hợp lệ</u>** ta được biểu thức exp có mở ngoặc đóng ngoặc:

$$(((5-1)-4)+(\frac{5}{1}))$$

Và s không ngoặc: $5\_-\_1\_-\_4\_+\_5\_/\_1\_$

⇨ Có thể dễ dàng nhận thấy thứ tự của phép tính là số đến phép tính đến số đến phép tính ….. nghĩa là mở đầu luôn là số và theo thứ tự tiếp theo sẽ xen kẽ nhau như vậy mới là biểu thức hợp lệ

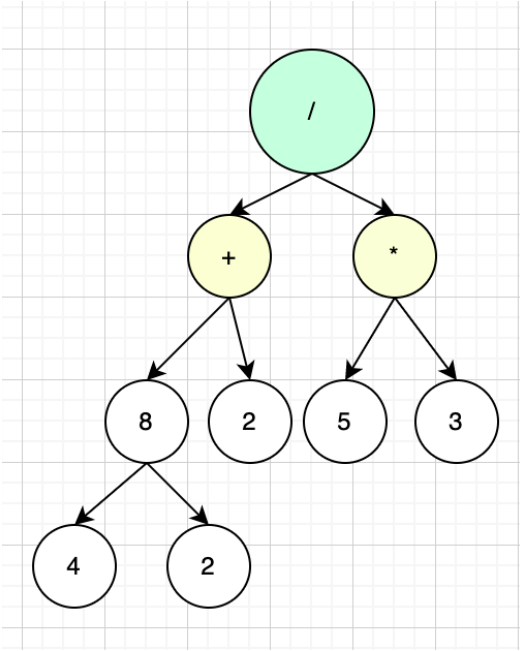


Xét cây trong hình là **<u>cây biểu thức không hợp lệ</u>** , ta được biểu thức exp có mở ngoặc đóng ngoặc:

$$(((4\ 8\ 2) + 2)/(5 * 3))$$

Và s không ngoặc: 4_8_2_ + _2_/_5_ * _3_

⇨ Nhận thấy thứ tự phép tính sai



Từ đó ta triển khai lập trình: lấy chuỗi xuất từ cây không dấu ngoặc để xét từng ký tự

- Bắt đầu với n là số lượng ký tự trong chuỗi
- Cho biến i chạy từ ký tự đầu tiên của chuỗi
- Tại mỗi lần lặp đặt 2 biến chuỗi là ***curr*** và ***next*** đại diện cho chuỗi cần xét chuỗi này với chuỗi kế tiếp cách nhau bởi dấu _
- Biến ***checkFirst*** để kiểm tra thử chuỗi ***curr*** đầu tiên có bắt đầu từ ký tự đầu tiên hay không vì theo thứ tự hoạch định phân tích logic ban đầu luôn là số thì biểu thức mới là biểu thức hợp lệ
- Sau đó tại vòng lặp con bên trong là thêm từng phần tử vào chuỗi ***curr***, nếu i bắt đầu từ 0 thì cho biến ***checkFirst*** nhận giá trị true, sau mỗi lần thêm phần tử thì i sẽ tăng 1 đơn vị, song song với việc đó là kiểm tra ban đầu liệu ký tự tại i có phải là _ hay không và I có < kích thước chuỗi không rồi mới thực thi vòng lặp con
- Kiểm tra điều kiện liệu chuỗi ***curr*** đó có bắt đầu từ 0 hay không, nếu có thì kiểm tra nó có phải là phép toán hay không, nếu phải thì sai điều kiện hoạch định ban đầu và trả về false của hàm ***check*** ngay lập tức.
- Tiếp tục kiểm tra liệu biến i bằng index cuối của chuỗi hay không, nếu có thì dừng vòng lặp rồi trả về true chứng tỏ đã duyệt hết chuỗi s và tất cả các điều kiện trong lặp thoả tính đến thời điểm xét cuối cùng

- Nhìn lại vòng lặp con, từ vòng lặp cho kí tự vào ***curr*** đó thì lúc này i đang là tại index mang kí tự "_" nên ta phải thêm 1 cho i để không thêm vào ***next*** không bị dính kí tự "_"
- Sau đó thực hiện tương tự như ***curr*** để thêm các kí tự của s vào ***next*** nhưng không kiểm tra index đầu
- Kiểm tra điều kiện nếu chuỗi curr và chuỗi next tiếp theo có cùng loại number hay phép toán thì ta trả về false để đúng hoạch định phân tích logic ban đầu
- Tương tự, sau lặp while con thứ 2 cho next thì I sẽ tại index mang ký tự "_" vì vậy phải thêm i tăng lên 1 nữa.
- Tiếp tục thực hiện vòng lặp lớn đến khi dừng.

14. Hàm tính toán biểu thức trong cây nếu như đã kiểm tra biểu thức là hợp lệ:

```
double calculate(Node* root) {
    if (root == NULL) return 0;
    if (root->left == NULL && root->right == NULL) return stod(root->data);
    double left = calculate(root->left);
    double right = calculate(root->right);
    if (root->data == "+")
        return left + right;
    else if (root->data == "-")
        return left - right;
    else if (root->data == "*")
        return left * right;
    else if (root->data == "/")
        return left / right;
    return 0;
}
```

Triển khai logic lập trình:

- Bắt đầu từ root, nếu node không có con thì trả về giá trị data của node cha, điều kiện này để thực hiện kiểm tra chính khi ta xét đến node con dưới cùng của cây không có con chỉ có data để tính toán, là các số hạng, lúc này ta dùng hàm stod để chuyển từ kiểu dữ liệu string về double và trả về data tại root nhằm thực hiện tính toán cho các đệ quy sau.

- Ta thực hiện đệ quy tìm số hạng ở các node cực trái và cực phải của cây
- Nếu node đang xét có dữ liệu kiểu phép toán + - * / tức là chứa con thì hàm sẽ đệ quy tính giá trị của cây con bên trái và bên phải bằng cách gọi chính nó với các nút trái phải làm đầu vào cho hàm calculate, sau đó thực hiện tính toán và trả về kết quả.
- Nếu data lưu trự trong nút gốc không phải là một trong 4 phép toán thì hàm sẽ trả về 0

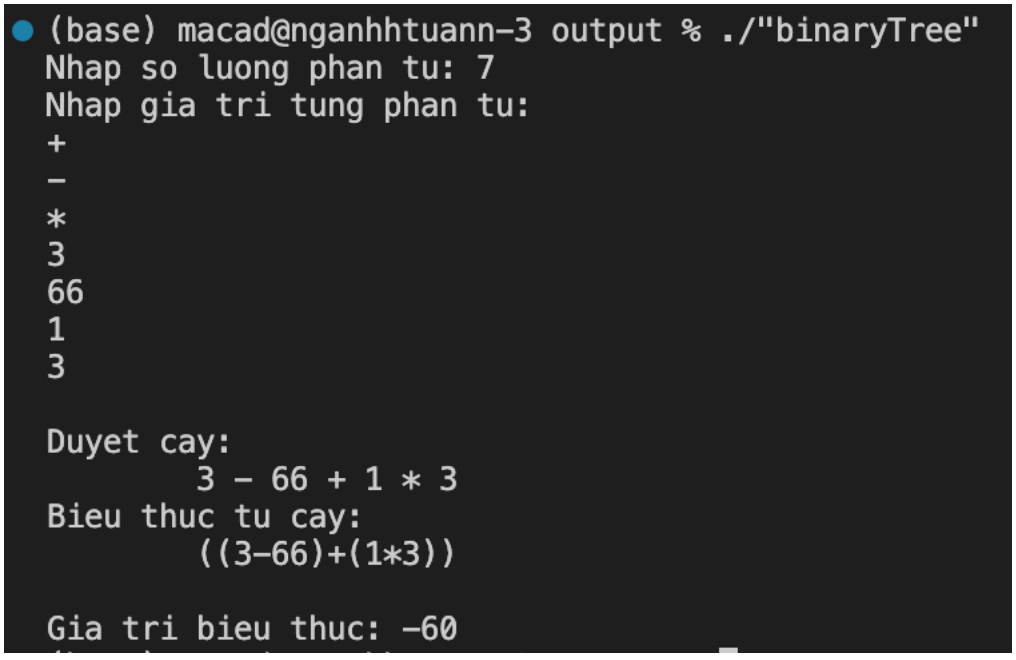
Hàm được triển khai dựa trên duyệt cây Left Right Node (PostOrder) để xử lý và tính toán.

15. Hàm main:

```
int main() {
    Tree* tree = new Tree;
    init(tree->root);

    int n;
    cout << "Nhap so luong phan tu: ";
    cin >> n;
    createTree(tree, n);

    cout << "\nDuyet cay: \n\t";
    traverse(tree->root);
    cout << endl;

    string exp;
    expressionBuilder(tree->root, exp);
    cout << "Bieu thuc tu cay: \n\t" << exp << endl;

    string s;
    stringBuilder(tree->root, s);
    if ( check(s)){
        double res = calculate(tree->root);
        cout << "\nGia tri bieu thuc: " << res << endl;
    } else {
        cout << "Bieu thuc khong hop le";
    }
    delete tree;
}
```

Kết quả hiển thị:

Cây hợp lệ:



Cây không hợp lệ: