



**数据结构课程设计报告**

**课程名称 数据结构**

**题目名称 计算器的实现**

**学生学院 计算机学院**

**专业班级 2019计算机科学与技术1班**

**学 号 3119004760**

**学生姓名 叶嘉轩**

**指导教师 李杨**

**2020 年 12 月 26 日**

**课程设计题目: 计算器的实现**

**编译环境**

编程语言使用Objective-C语言

编译器是Xcode12.1

界面是由iPhone模拟器展示的

##### 一、问题描述

实现一个计算器，能够展示出ui界面，并且利用栈和二叉树以及后缀表达式实现加减乘除的运算。

二、基本要求

1. 设计好计算器的ui界面，能够显示按键以及输入框和输出框，而且输入框和输出框能够实时更新
2. 字体可以随着输入内容的多少不断改变，输入的内容越多字越小，当输入的内容超过输入框范围时光标能够定位到输入框的尾部，展示出最新输入的内容，而且能够通过滑动回看之前的内容
3. 容错处理：1.在运算中如果有/0或者其他不合法的运算时输出框会直接显示错误，不进 行计算

2.在输入内容时如果出现类似连续输入两个操作符的情况，在输入第二个时 会输入失败

4）将输入的中缀表达式要转成后缀表达式再进行后继处理

5）了解栈和二叉树，并实现运算需要的相关接口

6）实现加减乘除以及括号的运算，运算过程中利用栈和二叉树，不能直接利用数组取值直接计算

##### 三、拓展要求

1）实现带括号的运算

2）实现更多的运算，比如平方，根号等等

##### 四、完成情况

1. 完成所有基本要求
2. 完成了第一项扩展要求

##### 五、概要设计

1. 数据类型的定义：

//二叉树

@interface TreeNode : NSObject

@property (nonatomic, copy) NSString \*data;//当前结点的数值

@property (nonatomic, strong) TreeNode \*leftTreeNode;//当前结点的左子树

@property (nonatomic, strong) TreeNode \*rightTreeNode;//当前结点的右子树

@end

//栈

@interface Stack : NSObject

//栈是用数组进行操作

@property (nonatomic, strong) NSMutableArray \*array;

@property (nonatomic, strong) NSMutableArray \*tempArray;

@end

1. 设计思路

程序运行起来，模拟器上就会展示计算器的ui界面，点击界面上的按钮输入框实时更新新输入的内容，当用户按下按键=时，首先会将输入的内容进行判错，若输入的格式正确则将利用栈将中缀表达式转变为后缀表达式，再将转换好的后缀表达式构造成一棵二叉树，在利用递归的方式将结果计算出来，然后把结果更新到输出框上。

1. 接口设计：

// 二叉树

//初始化二叉树

- (id)buildTree:(NSString \*)rootData;

//插入元素到左子树

- (id)insertToLeftTree:(NSString \*)leftNode : (TreeNode \*)treeNode;

//插入结点到右子树

- (void)insertNodeToRightTree:(TreeNode \*)treeNode;

//插入结点到左子树

- (void)insertNodeToLeftTree:(TreeNode \*)treeNode;

//插入元素到右子树

- (id)insertToRightTree:(NSString \*)rightNode : (TreeNode \*)treeNode;

//判断左子树是否为空

- (BOOL)leftTreeIsEmpty:(TreeNode \*)leftNode;

//判断右子树是否为空

- (BOOL)rightTreeIsEmpty:(TreeNode \*)rightNode;

//递归遍历二叉树输出结果

- (float)inOrderTraverse:(TreeNode \*)treeNode;

//销毁根结点

- (void)DestoryTree;

//更新根结点

- (void)changeRootNode:(TreeNode \*)node;

//栈

//操作符入栈

- (BOOL)PushIntoStack:(NSString \*)str;

//数字入栈

- (void)PushmIntoStack:(id)ids;

//判断栈是否为空

- (BOOL)StackIsEmpty;

//出栈

- (id)PopStack;

//出栈（构造二叉树时用的）

- (id)PopTempStack;

//获取栈顶元素

- (id)getStackHead;

//判断优先级

- (BOOL)isHigh:(NSString \*)chars;

//销毁栈

- (void)DestroyStack;

//将输入内容进行处理，将字符分格开方便处理

- (void) dealWithText:(NSString \*)Equation;

//将中缀表达式转成后缀表达式

- (void) TranslateToInversePolandForClculate:(NSMutableArray \*)array；

//根据后缀表达式构造二叉树

- (void)BuildTree:(NSMutableArray \*)array;

六、程序源代码

**ViewController .h文件**

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@end

**ViewController .m文件**

//

//  ViewController.m

//  OCCalculator

//

//  Created by MacBook pro on 2020/11/22.

//

//设备的宽高

#define SCREENWIDTH [UIScreen mainScreen].bounds.size.width

#define SCREENHEIGHT [UIScreen mainScreen].bounds.size.height

#import "ViewController.h"

#import <SDAutoLayout/SDAutoLayout.h>

#import "CollectionViewCell.h"

#import "Stack.h"

#import "TreeNode.h"

#import "BinaryTree.h"

@interface ViewController ()<UICollectionViewDataSource,UICollectionViewDelegate,UITextViewDelegate>

@property (strong, nonatomic) UITextView \*textView;

@property (strong, nonatomic) UICollectionView \*collectionView;

@property (copy,   nonatomic) NSArray \*array;

@property (strong, nonatomic) UILabel \*outPutLabel;

@property (strong, nonatomic) NSArray \*SignsArray;

@property (strong, nonatomic) Stack \*stack;

@property (strong, nonatomic) TreeNode \*treeNode;

@property (strong, nonatomic) BinaryTree \*binaryTree;

@property (strong, nonatomic) Stack \*tempStack;

@property (strong, nonatomic) TreeNode \*tempNode;

@end

@implementation ViewController

- (TreeNode \*)tempNode {

    if (!\_tempNode) {

        \_tempNode = [[TreeNode alloc] init];

    }

    return \_tempNode;

}

- (Stack \*)tempStack {

    if (!\_tempStack) {

        \_tempStack = [[Stack alloc] init];

    }

    return \_tempStack;

}

- (BinaryTree \*)binaryTree {

    if (!\_binaryTree) {

        \_binaryTree = [[BinaryTree alloc] init];

    }

    return \_binaryTree;

}

- (TreeNode \*)treeNode {

    if (!\_treeNode) {

        \_treeNode = [[TreeNode alloc] init];

    }

    return \_treeNode;

}

- (Stack \*)stack {

    if (!\_stack) {

        \_stack = [[Stack alloc] init];

    }

    return \_stack;

}

- (NSArray \*)SignsArray {

    if (\_SignsArray == NULL) {

        NSArray \*SignsArray = @[@"+",@"-",@"×",@"÷",@"√",@"ln"];

        \_SignsArray = SignsArray;

    }

    return \_SignsArray;

}

- (UITextView \*)textView {

    if (!\_textView) {

        \_textView = [[UITextView alloc] init];

    }

    return \_textView;

}

- (NSArray \*)array {

    if (!\_array) {

        \_array = [NSArray array];

        \_array = @[@"C",@"(",@")",@"÷",@"7",@"8",@"9",@"×",@"4",@"5",@"6",@"-",@"1",@"2",@"3",@"+",@"0",@".",@"="];

    }

    return \_array;

}

- (UILabel \*)outPutLabel {

    if (!\_outPutLabel) {

        \_outPutLabel = [[UILabel alloc] init];

    }

    return \_outPutLabel;

}

- (void)viewDidLoad {

    [super viewDidLoad];

    [self setUpUI];

}

- (void)setUpUI {

    //背景view的颜色

    self.view.backgroundColor = [UIColor blackColor];

    //输入框

    self.textView.backgroundColor = [UIColor blackColor];

    [self.view addSubview:self.textView];

    self.textView.editable = NO;

    self.textView.textColor = [UIColor whiteColor];

    [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:80]];

    // 添加监视

    [self.textView addObserver:self forKeyPath:@"text" options:NSKeyValueObservingOptionNew context:nil];

    self.textView.sd\_layout.topSpaceToView(self.navigationController.navigationBar, 80).leftSpaceToView(self.view, 10).rightSpaceToView(self.view, 10).heightIs(SCREENHEIGHT/3-50);

    //结果输出框

    self.outPutLabel.backgroundColor = [UIColor blackColor];

    self.outPutLabel.textColor = [UIColor whiteColor];

    [self.outPutLabel setFont:[UIFont fontWithName:@"Helvetica-Bold" size:32]];

    self.outPutLabel.textAlignment = NSTextAlignmentRight;

    [self.view addSubview:self.outPutLabel];

    self.outPutLabel.sd\_layout.leftEqualToView(self.textView).rightEqualToView(self.textView)

    .topSpaceToView(self.textView, 5).heightIs(50);

    //底部的按钮

    UICollectionViewFlowLayout \*layout = [[UICollectionViewFlowLayout alloc] init];

    //设置内边距

    layout.sectionInset = UIEdgeInsetsMake(12, 3, 12, 3);

    //设置item大小

    layout.itemSize = CGSizeMake(SCREENWIDTH/5, SCREENWIDTH/5);

    self.collectionView = [[UICollectionView alloc] initWithFrame:CGRectZero collectionViewLayout:layout];

    self.collectionView.backgroundColor = [UIColor blackColor];

    self.collectionView.dataSource = self;

    self.collectionView.delegate = self;

    [self.collectionView registerClass:[CollectionViewCell class]

     forCellWithReuseIdentifier:@"cell"];

    [self.view addSubview:self.collectionView];

    self.collectionView.sd\_layout.topSpaceToView(self.outPutLabel, 0).leftEqualToView(self.textView).rightEqualToView(self.textView).bottomEqualToView(self.view);

}

#pragma mark - uicollectionViewDelegate & DataSource

- (NSInteger)numberOfSectionsInCollectionView:(UICollectionView \*)collectionView{

    return 1;

}

- (NSInteger)collectionView:(UICollectionView \*)collectionView numberOfItemsInSection:(NSInteger)section {

    return 19;

}

- (UICollectionViewCell \*)collectionView:(UICollectionView \*)collectionView cellForItemAtIndexPath:(NSIndexPath \*)indexPath {

    CollectionViewCell \*cell = [\_collectionView dequeueReusableCellWithReuseIdentifier:@"cell" forIndexPath:indexPath];

    if (indexPath.row==0||indexPath.row==1||indexPath.row==2) {

        cell.backgroundColor = [UIColor colorWithRed:163/255.0 green:164/255.0 blue:165/255.0 alpha:1];

    }else if (indexPath.row==3||indexPath.row==7||indexPath.row==11||indexPath.row==15||indexPath.row==18){

        cell.backgroundColor = [UIColor colorWithRed:244/255.0 green:159/255.0 blue:63/255.0 alpha:1];

    }else {

        cell.backgroundColor = [UIColor colorWithRed:51/255.0 green:52/255.0 blue:53/255.0 alpha:1];

    }

    if (indexPath.row==16) {

        cell.layer.cornerRadius = SCREENWIDTH/10;

    }else {

        cell.layer.cornerRadius = SCREENWIDTH/10;

    }

    cell.titleLabel.text = self.array[indexPath.row];

    return cell;

}

-(CGSize)collectionView:(UICollectionView \*)collectionView layout:(UICollectionViewLayout \*)collectionViewLayout sizeForItemAtIndexPath:(NSIndexPath \*)indexPath {

    if (indexPath.row == 16) {

        return CGSizeMake(SCREENWIDTH/2-10, SCREENWIDTH/5);

    }

    return CGSizeMake(SCREENWIDTH/5, SCREENWIDTH/5);

}

- (void)collectionView:(UICollectionView \*)collectionView didSelectItemAtIndexPath:(NSIndexPath \*)indexPath {

    NSString \*text = self.textView.text;

    if (!text.length) {

        //一开始不能输入操作符

        NSString \*Chars = self.array[indexPath.row];

        BOOL flag = ([Chars isEqual:@"÷"] || [Chars isEqual:@"×"] ||[Chars isEqual:@"+"] || [Chars isEqual:@"-"]||[Chars isEqual:@"="] || [Chars isEqual:@"."] || [Chars isEqual:@"～"] || [Chars isEqual:@"C"] || [Chars isEqual:@"←"] || [Chars isEqual:@")"]);

        if (flag) {

            //第一个是操作符不做任何处理

        }else {

            //第一个不是操作符

            text = [text stringByAppendingString:self.array[indexPath.row]];

            self.textView.text = text;

        }

    }else {

        if ([self.array[indexPath.row] isEqual:@"～"] || [[self.textView.text substringFromIndex:self.textView.text.length - 1] isEqual:@"="]) {

            //输入的是～不做任何处理

            if ([self.array[indexPath.row]  isEqual: @"C"]) {

                self.textView.text = @"";

                [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:80]];

                self.outPutLabel.text = @"";

            }else if ([self.array[indexPath.row]  isEqual: @"←"]){

                if (self.textView.hasText) {

                    NSString \*text = self.textView.text;

                    text = [text substringToIndex:text.length-1];

                    self.textView.text = text;

                }

            }

        }else {

            //判断操作符是否重叠

            NSString \*lastStr = [text substringFromIndex:text.length-1];

            if (([lastStr isEqual:@"÷"] || [lastStr isEqual:@"×"] ||[lastStr isEqual:@"+"] || [lastStr isEqual:@"-"] || [lastStr isEqual:@"."] || [lastStr isEqual:@"="]) && ([self.array[indexPath.row] isEqual:@"÷"] || [self.array[indexPath.row] isEqual:@"×"] ||[self.array[indexPath.row] isEqual:@"+"] || [self.array[indexPath.row] isEqual:@"-"] || [self.array[indexPath.row] isEqual:@"."] || [self.array[indexPath.row] isEqual:@"="])) {

                //重叠

            }else {

                text = [text stringByAppendingString:self.array[indexPath.row]];

                if ([self.array[indexPath.row]  isEqual: @"C"]) {

                    self.textView.text = @"";

                    [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:80]];

                    self.outPutLabel.text = @"";

                }else if ([self.array[indexPath.row]  isEqual: @"←"]){

                    if (self.textView.hasText) {

                        NSString \*text = self.textView.text;

                        text = [text substringToIndex:text.length-1];

                        self.textView.text = text;

                    }

                }else {

                    self.textView.text = text;

                }

                if ([self.array[indexPath.row] isEqual:@"="]) {

                    if (self.textView.hasText) {

                        //判断是否包含不合法字符串

                        if ([self.textView.text rangeOfString:@"÷0"].location != NSNotFound) {

                            self.outPutLabel.text = @"错误";

                        }else {

                            [self Calculate:self.textView.text];

                        }

                    }

                }

            }

        }

    }

    //textView自动跑到最后一行

    [\_textView scrollRectToVisible:CGRectMake(0, \_textView.contentSize.height-15, \_textView.contentSize.width, 10) animated:YES];

}

// 接收textView输入框值的变化

- (void)observeValueForKeyPath:(NSString \*)keyPath ofObject:(id)object change:(NSDictionary \*)change context:(void \*)context {

    NSString \*text = [object text];

    NSUInteger count = text.length;

    if (count<=9) {

        switch (count) {

            case 9:{

                [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:40]];

                break;

            }

            case 8:{

                [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:50]];

                break;

            }

            case 7:{

                [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:60]];

                break;

            }

            case 6:{

                [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:70]];

                break;

            }

            default:

                break;

        }

    }else {

        [self.textView setFont:[UIFont fontWithName:@"Helvetica-Bold" size:38]];

    }

}

- (void)Calculate :(NSString \*)text {

    [self dealWithText:text];

}

//处理字符串

- (void) dealWithText:(NSString \*)Equation {

    Equation = [Equation stringByReplacingOccurrencesOfString:@"+" withString:@"|+|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@"-" withString:@"|-|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@"×" withString:@"|\*|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@"÷" withString:@"|/|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@"(" withString:@"|(|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@")" withString:@"|)|"];

    Equation = [Equation stringByReplacingOccurrencesOfString:@"=" withString:@""];

    NSMutableArray \*array = [Equation componentsSeparatedByString:@"|"].mutableCopy;

    [array removeObject:@""];

    if ([array[0] isEqualToString:@"-"]||[array[0] isEqualToString:@"+"]) {

        [array insertObject:@"0" atIndex:0];

    }

    //判断是否只有一个数字的运算

    if (array.count == 1) {

        self.outPutLabel.text = array[0];

    }else {

        //将中缀表达式转换为后缀表达式

        [self TranslateToInversePolandForClculate:array];

    }

}

- (void) TranslateToInversePolandForClculate:(NSMutableArray \*)array {

    //NSMutableArray \*Stack = [NSMutableArray array];

    NSMutableArray \*outPutArray = [NSMutableArray array];

    for (int i=0; i<array.count; i++) {

        NSString \*Chars = array[i];

        //操作符号判断

        BOOL flag = ([Chars isEqual:@"/"] || [Chars isEqual:@"\*"] ||[Chars isEqual:@"+"] || [Chars isEqual:@"-"] || [Chars isEqual:@"("] || [Chars isEqual:@")"] );

        //如果只是数字那就压入栈

        if (!flag) {

            [outPutArray addObject:Chars];

        }

        //如果是操作符

        if (flag) {

            while ([Chars isEqual:@"/"] || [Chars isEqual:@"\*"] ||[Chars isEqual:@"+"] || [Chars isEqual:@"-"] || [Chars isEqual:@"("] || [Chars isEqual:@")"]) {

                //先判断是否入栈成功

                if ([self.stack PushIntoStack:Chars]) {

                    //入栈成功

                    break;

                }else {

                    //入栈失败

                    //出栈

                    while ([self.stack StackIsEmpty] == NO) {

                        if ([self.stack isHigh:Chars] == NO || [Chars isEqual:@")"]) {

                            if ([Chars isEqual:@")"]) {

                                NSString \*chars = [self.stack PopStack];

                                while (![chars isEqual:@"("] && chars.length) {

                                    [outPutArray addObject:chars];

                                    chars = [self.stack PopStack];

                                }

                            }else {

                                if ([self.stack.array.lastObject isEqual:@"("]) {

                                }else {

                                    [outPutArray addObject:[self.stack PopStack]];

                                }

                            }

                        }else {

                            break;

                        }

                    }

                    //入栈

                    //如果操作符是右括号），则不需要入栈

                    if ([Chars isEqual:@")"]) {

                    }else {

                        [self.stack PushIntoStack:Chars];

                    }

                    break;

                }

            }

        }

    }

    //栈中还残留其他操作符

    while ([self.stack StackIsEmpty] == NO) {

        [outPutArray addObject:[self.stack PopStack]];

    }

    //销毁栈

    [self.stack DestroyStack];

    [self BuildTree:outPutArray];

}

- (void)BuildTree:(NSMutableArray \*)array {

    [self.stack DestroyStack];

    [self.tempStack DestroyStack];

    NSString \*Chars,\*chars1,\*chars2;

    for (int i=0; i<array.count; i++) {

        Chars = array[i];

        //操作符号判断

        BOOL flag = ([Chars isEqual:@"/"] || [Chars isEqual:@"\*"] ||[Chars isEqual:@"+"] || [Chars isEqual:@"-"]);

        //如果只是数字那就压入栈

        if (!flag) {

            //[outPutArray addObject:Chars];

            [self.tempStack PushmIntoStack:Chars];

        }else {

            //操作符

            //重新构建二叉树的根结点

            //先判断是否是第一次构建根结点

            if (!self.binaryTree.rootNode.data.length) {

                //第一次构建根结点

                self.treeNode = [self.binaryTree buildTree:Chars];

                //栈元素出栈

                chars2 = [self.tempStack PopTempStack];

                chars1 = [self.tempStack PopTempStack];

                //插入右结点

                [self.binaryTree insertToRightTree:chars2 :self.treeNode];

                //插入左结点

                [self.binaryTree insertToLeftTree:chars1 :self.treeNode];

                //操作入栈

                [self.tempStack PushmIntoStack:self.binaryTree.rootNode];

            }else {

                TreeNode \*treeNode = [[TreeNode alloc] init];

                treeNode.data = Chars;

                //栈顶元素出栈

                //右子树

                if ([[self.tempStack getStackHead] isKindOfClass:[NSString class]]) {

                    chars1 = [self.tempStack PopTempStack];

                    //更新右结点

                    //插入右结点

                    [self.binaryTree insertToRightTree:chars1 :treeNode];

                }else {

                    TreeNode \*tempNode = [[TreeNode alloc] init];

                    tempNode = [self.tempStack PopTempStack];

                    //更新右结点

                    treeNode.rightTreeNode = tempNode;

                }

                //栈顶元素再次出栈

                if ([[self.tempStack getStackHead] isKindOfClass:[NSString class]]) {

                    chars1 = [self.tempStack PopTempStack];

                    //更新左结点

                    [self.binaryTree insertToLeftTree:chars1 :treeNode];

                }else {

                    TreeNode \*tempNode = [[TreeNode alloc] init];

                    tempNode = [self.tempStack PopTempStack];

                    treeNode.leftTreeNode = tempNode;

                }

                //判断是否是最后一个操作符

                if (i == array.count - 1) {

                    //更新根结点

                    [self.binaryTree changeRootNode:treeNode];

                }else {

                    //将操作添加到栈中

                    [self.tempStack PushmIntoStack:treeNode];

                }

            }

        }

    }

    float value = [self.binaryTree inOrderTraverse:self.binaryTree.rootNode];

    NSString \*outPut = [NSString stringWithFormat:@"%f",value];

    NSDecimalNumber \*nsd = [NSDecimalNumber decimalNumberWithString:outPut];

    self.outPutLabel.text = [nsd stringValue];

    //销毁树

    [self.binaryTree DestoryTree];

}

@end

**TreeNode.h文件**

#import <Foundation/Foundation.h>

@interface TreeNode : NSObject

@property (nonatomic, copy) NSString \*data;//当前结点的数值

@property (nonatomic, strong) TreeNode \*leftTreeNode;//当前结点的左子树

@property (nonatomic, strong) TreeNode \*rightTreeNode;//当前结点的右子树

@end

**TreeNode.m文件**

#import "TreeNode.h"

@implementation TreeNode

@end

**BinaryTree.h文件**

//

//  BinaryTree.h

//  OCCalculator

//

//  Created by MacBook pro on 2020/12/18.

//

#import <Foundation/Foundation.h>

#import "TreeNode.h"

NS\_ASSUME\_NONNULL\_BEGIN

@interface BinaryTree : NSObject

@property (nonatomic, strong) TreeNode \*rootNode;

@property (nonatomic, copy)   NSString \*chars;

//初始化二叉树

- (id)buildTree:(NSString \*)rootData;

//插入元素到左子树

- (id)insertToLeftTree:(NSString \*)leftNode : (TreeNode \*)treeNode;

//插入结点到右子树

- (void)insertNodeToRightTree:(TreeNode \*)treeNode;

//插入结点到左子树

- (void)insertNodeToLeftTree:(TreeNode \*)treeNode;

//插入元素到右子树

- (id)insertToRightTree:(NSString \*)rightNode : (TreeNode \*)treeNode;

//判断左子树是否为空

- (BOOL)leftTreeIsEmpty:(TreeNode \*)leftNode;

//判断右子树是否为空

- (BOOL)rightTreeIsEmpty:(TreeNode \*)rightNode;

//递归遍历二叉树输出结果

- (float)inOrderTraverse:(TreeNode \*)treeNode;

//销毁根结点

- (void)DestoryTree;

//更新根结点

- (void)changeRootNode:(TreeNode \*)node;

@end

NS\_ASSUME\_NONNULL\_END

**BinaryTree.m文件**

//

//  BinaryTree.m

//  OCCalculator

//

//  Created by MacBook pro on 2020/12/18.

//

#import "BinaryTree.h"

@implementation BinaryTree

//初始化二叉树

- (id)buildTree:(NSString \*)rootData {

    self.rootNode = [[TreeNode alloc] init];

    self.rootNode.data = rootData;

    return self.rootNode;

}

//插入元素到左子树

- (id)insertToLeftTree:(NSString \*)leftNode : (TreeNode \*)treeNode {

    treeNode.leftTreeNode = [[TreeNode alloc] init];

    treeNode.leftTreeNode.data = leftNode;

    return treeNode.leftTreeNode;

}

//插入结点到右子树

- (void)insertNodeToRightTree:(TreeNode \*)treeNode {

    treeNode.rightTreeNode = self.rootNode;

    self.rootNode = treeNode;

}

//插入结点到左子树

- (void)insertNodeToLeftTree:(TreeNode \*)treeNode {

    treeNode.leftTreeNode = self.rootNode;

    self.rootNode = treeNode;

}

//插入元素到右子树

- (id)insertToRightTree:(NSString \*)rightNode : (TreeNode \*)treeNode {

    treeNode.rightTreeNode = [[TreeNode alloc] init];

    treeNode.rightTreeNode.data = rightNode;

    return treeNode.rightTreeNode;

}

//判断左子树是否为空

- (BOOL)leftTreeIsEmpty:(TreeNode \*)leftNode {

    if (!leftNode.leftTreeNode.data.length) {

        return YES;

    }

    return NO;

}

//判断右子树是否为空

- (BOOL)rightTreeIsEmpty:(TreeNode \*)rightNode {

    if (!rightNode.rightTreeNode.data.length) {

        return YES;

    }

    return NO;

}

//递归遍历二叉树

- (float)inOrderTraverse:(TreeNode \*)treeNode {

    if (!treeNode.data.length) {

        return 0;

    }else {

        self.chars = treeNode.data;

        if ([self.chars isEqual:@"+"]) {

            //return [self inOrderTraverse:treeNode.leftTreeNode] + [self inOrderTraverse:treeNode.rightTreeNode];

            NSDecimalNumber \*num1 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.leftTreeNode]]];

            NSDecimalNumber \*num2 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.rightTreeNode]]];

            NSDecimalNumber \*num = [num1 decimalNumberByAdding:num2];

            return num.floatValue;

        }

        if ([self.chars isEqual:@"-"]) {

            //return [self inOrderTraverse:treeNode.leftTreeNode] - [self inOrderTraverse:treeNode.rightTreeNode];

            NSDecimalNumber \*num1 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.leftTreeNode]]];

            NSDecimalNumber \*num2 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.rightTreeNode]]];

            NSDecimalNumber \*num = [num1 decimalNumberBySubtracting:num2];

            return num.floatValue;

        }

        if ([self.chars isEqual:@"\*"]) {

            //return [self inOrderTraverse:treeNode.leftTreeNode] \* [self inOrderTraverse:treeNode.rightTreeNode];

            NSDecimalNumber \*num1 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.leftTreeNode]]];

            NSDecimalNumber \*num2 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.rightTreeNode]]];

            NSDecimalNumber \*num = [num1 decimalNumberByMultiplyingBy:num2];

            return num.floatValue;

        }

        if ([self.chars isEqual:@"/"]) {

            //return [self inOrderTraverse:treeNode.leftTreeNode] / [self inOrderTraverse:treeNode.rightTreeNode];

            NSDecimalNumber \*num1 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.leftTreeNode]]];

            NSDecimalNumber \*num2 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",[self inOrderTraverse:treeNode.rightTreeNode]]];

            NSDecimalNumber \*num = [num1 decimalNumberByDividingBy:num2];

            return num.floatValue;

        }else {

            //return treeNode.data.floatValue;

            NSDecimalNumber \*num1 = [NSDecimalNumber decimalNumberWithString:[NSString stringWithFormat:@"%f",treeNode.data.floatValue]];

            return num1.floatValue;

        }

    }

}

//销毁根结点

- (void)DestoryTree {

    self.rootNode = [[TreeNode alloc] init];

}

//更新根结点

- (void)changeRootNode:(TreeNode \*)node {

    self.rootNode = node;

}

@end

**Stack.h文件**

//

//  Stack.h

//  OCCalculator

//

//  Created by MacBook pro on 2020/12/17.

//

#import <Foundation/Foundation.h>

NS\_ASSUME\_NONNULL\_BEGIN

@interface Stack : NSObject

//栈是用数组进行操作

@property (nonatomic, strong) NSMutableArray \*array;

@property (nonatomic, strong) NSMutableArray \*tempArray;

//操作符入栈

- (BOOL)PushIntoStack:(NSString \*)str;

//数字入栈

- (void)PushmIntoStack:(id)ids;

//判断栈是否为空

- (BOOL)StackIsEmpty;

//出栈

- (id)PopStack;

//出栈（构造二叉树时用的）

- (id)PopTempStack;

//获取栈顶元素

- (id)getStackHead;

//判断优先级

- (BOOL)isHigh:(NSString \*)chars;

//销毁栈

- (void)DestroyStack;

@end

NS\_ASSUME\_NONNULL\_END

**Stack.m文件**

//

//  Stack.m

//  OCCalculator

//

//  Created by MacBook pro on 2020/12/17.

//

#import "Stack.h"

#import "TreeNode.h"

@interface Stack()

@property (nonatomic, strong) NSArray \*operatorArray;

@property (nonatomic, copy)   NSString \*chars;

@property (nonatomic, strong) TreeNode \*treeNode;

@end

@implementation Stack

- (TreeNode \*)treeNode {

    if (!\_treeNode) {

        \_treeNode = [[TreeNode alloc] init];

    }

    return \_treeNode;

}

- (NSArray \*)operatorArray {

    if (!\_operatorArray) {

        \_operatorArray = @[@"+",@"-",@"×",@"÷"];

    }

    return \_operatorArray;

}

//栈的初始化

- (NSMutableArray \*)array {

    if (!\_array) {

        \_array = [NSMutableArray array];

    }

    return \_array;

}

- (NSMutableArray \*)tempArray {

    if (!\_tempArray) {

        \_tempArray = [NSMutableArray array];

    }

    return \_tempArray;

}

//操作符入栈

- (BOOL)PushIntoStack:(NSString \*)str {

    if ([str isEqual:@")"]) {

        return NO;

    }

    if (self.array.count == 0 || [self.operatorArray indexOfObject:str]/2 > [self.operatorArray indexOfObject:self.array[self.array.count - 1]]/2 || [str isEqual:@"("] || [self.array.lastObject isEqual:@"("]) {

        //栈为空或者操作符的优先级比栈顶元素高

        [self.array addObject:str];

        return YES;

    }

    return NO;

}

//数字入栈

- (void)PushmIntoStack:(id)ids{

    [self.tempArray addObject:ids];

}

//判断栈是否为空

- (BOOL)StackIsEmpty {

    if (self.array.count == 0) {

        return YES;

    }

    return NO;

}

//出栈

- (id)PopStack {

    if (self.array.count) {

        self.chars = self.array.lastObject;

        [self.array removeLastObject];

        return self.chars;

    }

    return nil;

}

//出栈

- (id)PopTempStack {

    if (self.tempArray.count) {

        if ([self.tempArray.lastObject isKindOfClass:[NSString class]]) {

            self.chars = self.tempArray.lastObject;

            [self.tempArray removeLastObject];

            return self.chars;

        }else {

            self.treeNode = self.tempArray.lastObject;

            [self.tempArray removeLastObject];

            return self.treeNode;

        }

    }

    return nil;

}

//获取栈顶元素

- (id)getStackHead {

    if (self.tempArray.count) {

        if ([self.tempArray.lastObject isKindOfClass:[NSString class]]) {

            self.chars = self.tempArray.lastObject;

            return self.chars;

        }else {

            self.treeNode = self.tempArray.lastObject;

            return self.treeNode;

        }

    }

    return nil;

}

//判断优先级

- (BOOL)isHigh:(NSString \*)chars {

    if ([self.operatorArray indexOfObject:chars]/2 > [self.operatorArray indexOfObject:self.array[self.array.count - 1]]/2 || [self.array.lastObject isEqual:@"("]) {

        return YES;

    }

    return NO;

}

//销毁栈

- (void)DestroyStack {

    [self.array removeAllObjects];

    [self.tempArray removeAllObjects];

}

@end

**CollectionViewCell.h文件**

#import <UIKit/UIKit.h>

NS\_ASSUME\_NONNULL\_BEGIN

@interface CollectionViewCell : UICollectionViewCell

@property (strong, nonatomic) UILabel \*titleLabel;

@end

NS\_ASSUME\_NONNULL\_END

**CollectionViewCell.m文件**

//

//  CollectionViewCell.m

//  OCCalculator

//

//  Created by MacBook pro on 2020/11/23.

//

#import "CollectionViewCell.h"

#import <SDAutoLayout/SDAutoLayout.h>

@implementation CollectionViewCell

- (instancetype)initWithFrame:(CGRect)frame {

    self = [super initWithFrame:frame];

    if (self) {

        [self setupView];

    }

    return self;

}

- (void)setupView {

    self.titleLabel = [[UILabel alloc] init];

    self.titleLabel.textColor = [UIColor whiteColor];

    [self.titleLabel setFont:[UIFont fontWithName:@"Helvetica-Bold" size:32]];

    self.titleLabel.textAlignment = NSTextAlignmentCenter;

    [self addSubview:self.titleLabel];

    self.titleLabel.sd\_layout.leftEqualToView(self).rightEqualToView(self).topEqualToView(self).bottomEqualToView(self);

}

@end

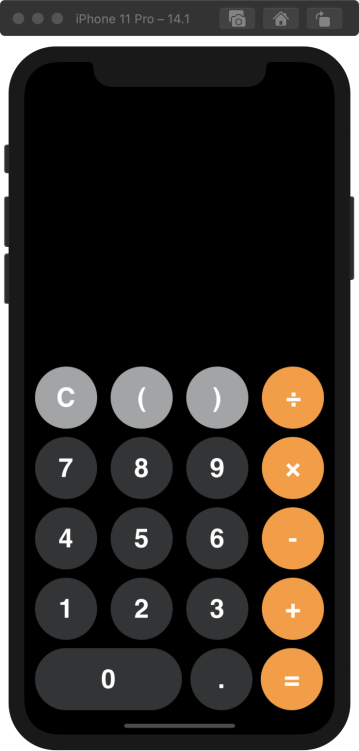
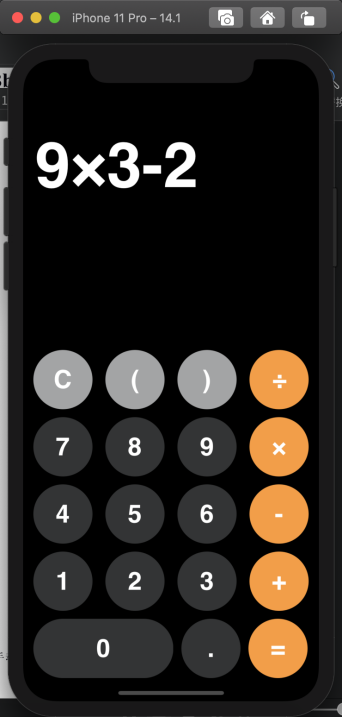
##### 七、调试分析

1. 调试中遇到的问题分析和解决：
2. C语言中是用结构体来定义栈和二叉树的结构类型的，而在Objective-C中，为了更方便地处理数据，我是用了模型数据来代替结构体。
3. 一开始容错处理没做好，比如连续输入两个操作符等等，后面加上判错处理。
4. 经验和体会

通过对计算器的实现，我加深了对利用栈将中缀表达式转后缀表达式，再利用后缀表达式构造二叉树，最后利用递归计算结果的认识，我发现将中缀表达式转成后缀表达式后，将不考虑运算符的优先级，大大提高了运算的速度，而在使用栈的过程中让我更加熟悉入栈出栈等基本操作，而在够=构造二叉树的时候则让我对树有了更进一步的认识。

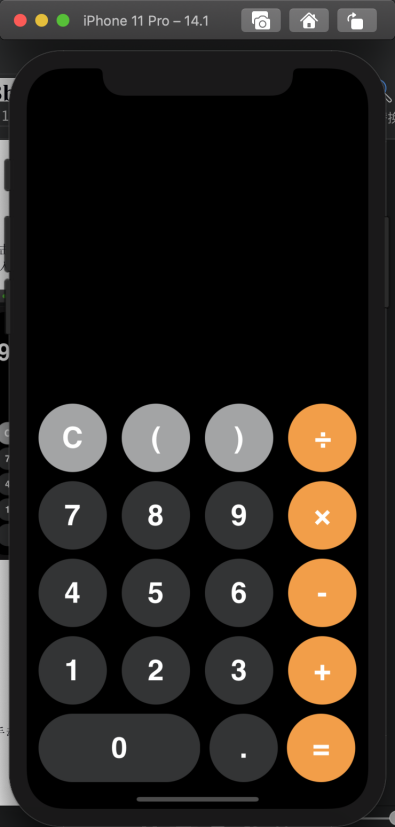
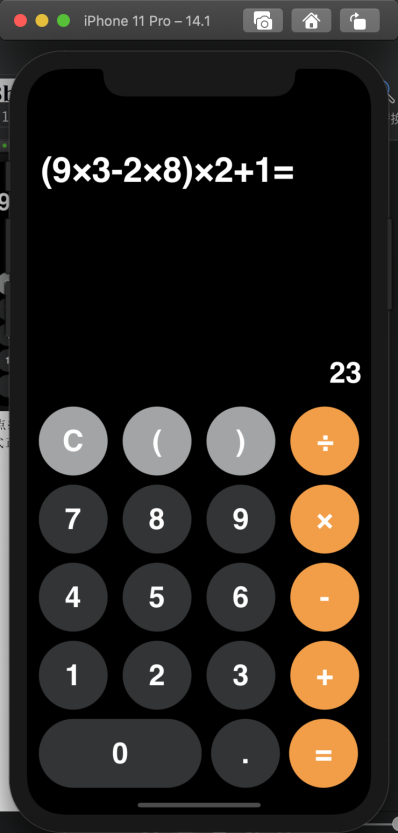
##### 八、用户使用说明

运行程序后显示如下界面 点击按键上的按钮时，输入框会实时更新新输入的内容

当点击按键C时会清空输入框的内容 而点按键=会对输入框的内容进行判错

格式正确的话会计算出正确的结果。

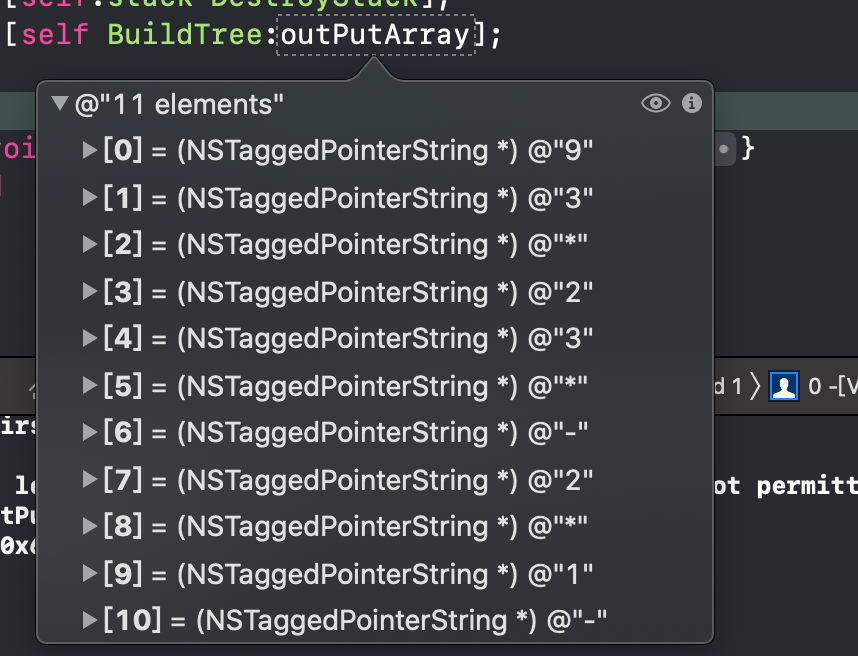
输入格式错误则输出框会显示错误



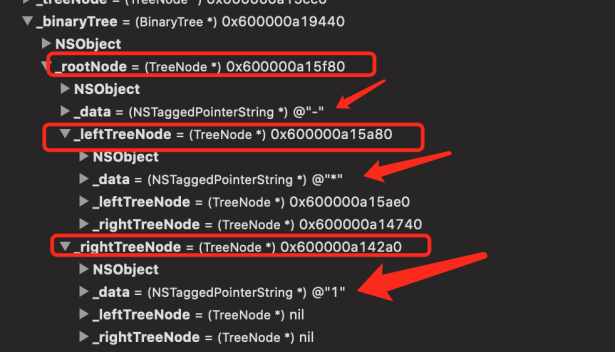
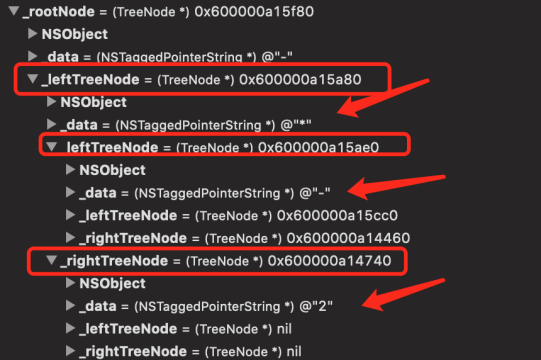
##### 测试结果

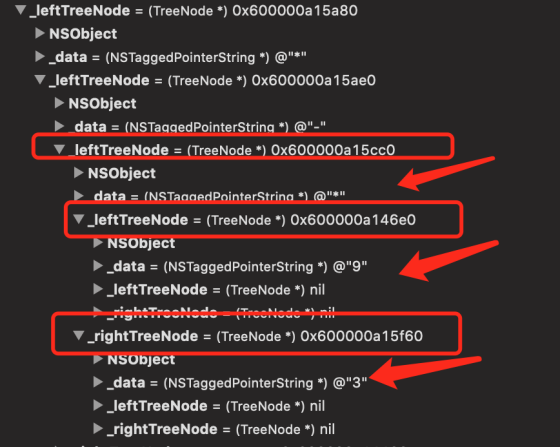
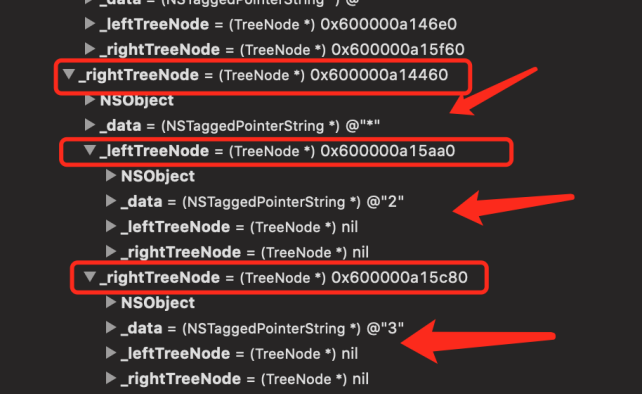
1. (9x3 - 2x3)x2-1

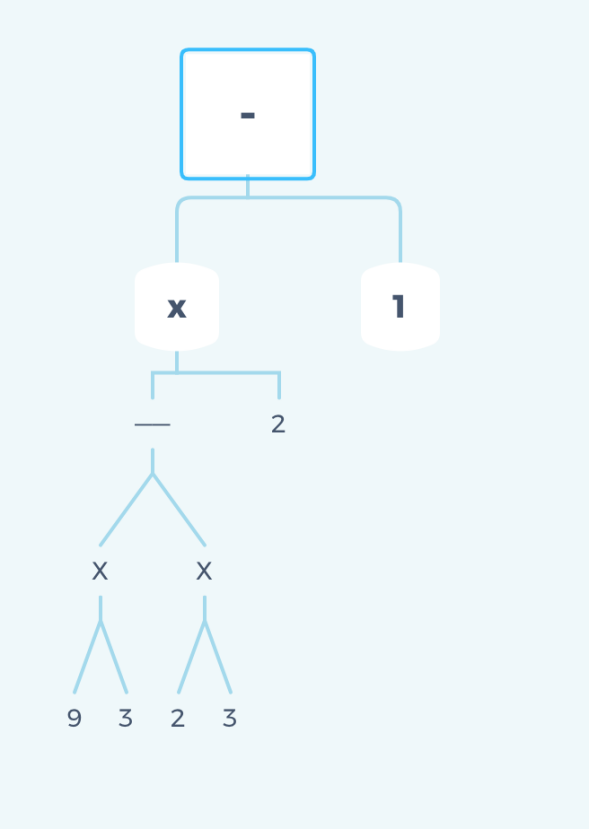
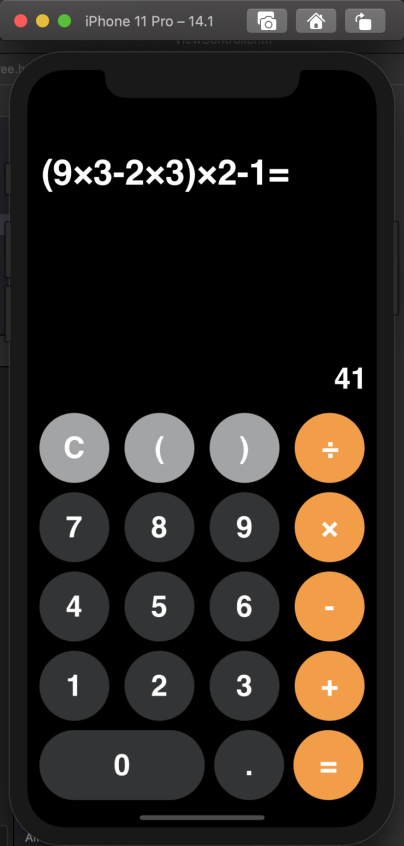
对应的后缀表达式为 93\*23\*-2\*1-



对应的树结构为

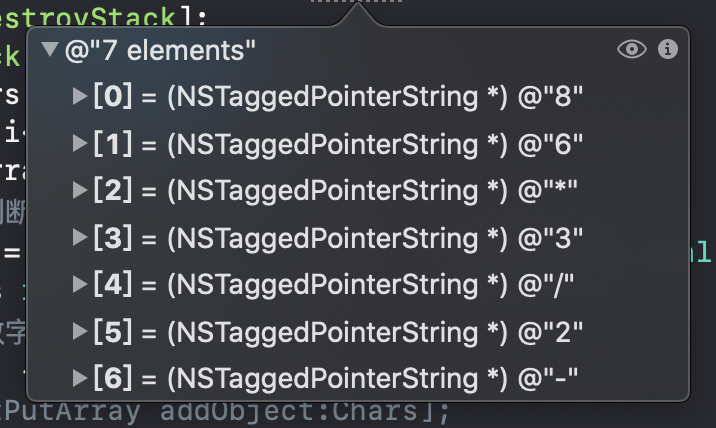
 

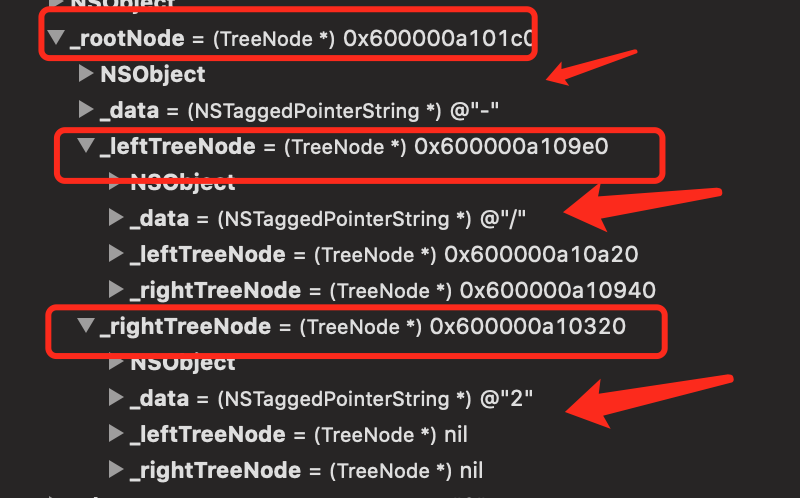
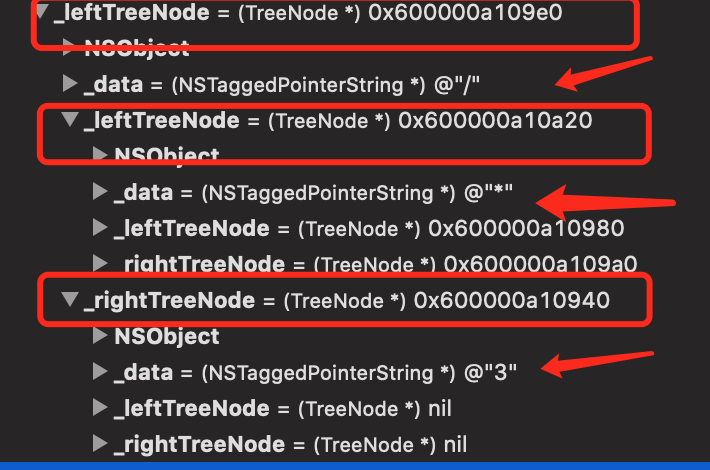
 

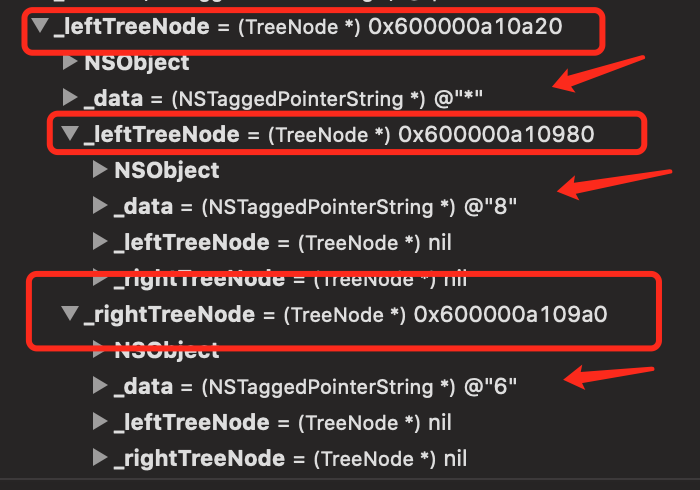
2）8x6/3-2

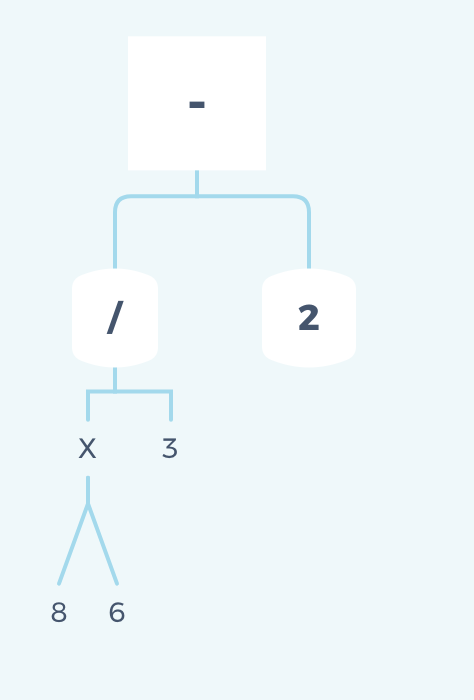
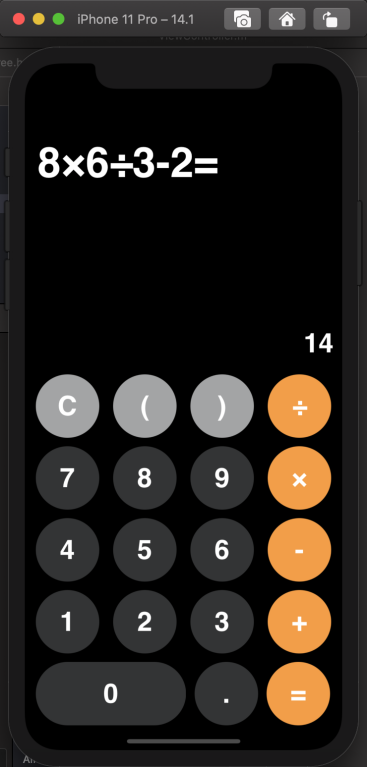
对应的后缀表达式为 86\*3/2-



对应的树结构为



3）9-(9x3/0)-1

