|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Языки программирования»  Вариант 20 | | |
|  | | Работу выполнил  студент группы ПМИ-9,10-2023 1 курса  Нуждин А. В.  «19» Июня 2024 г. |
| Работу проверил  Рубцова М. Б.  «19» Июня 2024 г. |
| Пермь 2024 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 3](#_Toc153130029)

[Код программы 3](#_Toc153130030)

[Инструкция по применению стилей и оформлению работы 4](#_Toc153130031)

# Постановка задачи

Написать программу, которая по заданной формуле строит дерево и

производит вычисления с помощью построенного дерева. Формула задана в

традиционной инфиксной записи, в ней могут быть скобки, максимальная

степень вложенности которых ограничивается числом 10. Аргументами могут

быть целые числа и переменные, задаваемые однобуквенными именами.

Допустимые операции: +, -, \*, /. Унарный минус допустим. С помощью

построенного дерева формулы упростить формулу, заменяя в ней все

поддеревья, соответствующие формулам (f1\*f3±f2\*f3) и (f1\*f2±f1\*f3) на

поддеревья, соответствующие формулам ((f1±f2)\*f3) и (f1\*(f2±f3)).

# Алгоритм решения

Словесное описание алгоритма:

Запускаем программу. Начинаем читать данные из файла «File.txt». Разделяем прочитанную строку на символы и заполняем ими список, попутно задавая значения встреченных в формуле переменных. Переменные записываются в словарь по принципу ключ = название переменной, значение = значение переменной.

После этого завершаем обработку специальной функцией: Проходимся по элементам списка и, если встречаем число или отрицательное число запускаем цикл, который объединяет все последующие числа в одно и завершается, когда встречает символ, не являющийся числом. Если объединённое «число» не является числом, получаем ошибку. По завершению мы получаем подходящий нашим целям список. Далее следует построение дерева с помощью трёх рекурсивных функций.

Первая функция идёт по следующему алгоритму: получаем результат действия второй функции (далее обозначается, как «А»), в процессе которой часть элементов списка собирается в одно поддерево. Затем, если следующий элемент списка является знаком «+» или «-» мы возвращаем новое поддерево, корнем которого является либо знак «+», либо знак «-», а листьями поддерево А и результат действия рекурсии первой функции. В ином случае, мы просто возвращаем поддерево А.

Вторая функция идёт по следующему алгоритму: получаем результат действия третьей функции (далее обозначается, как «Б»), в процессе которой часть элементов списка собирается в одно поддерево. Затем, если следующий элемент списка является знаком «\*» или «/» мы возвращаем новое поддерево, корнем которого является либо знак «\*», либо знак «/», а листьями поддерево Б и результат действия рекурсии второй функции. В ином случае, мы просто возвращаем поддерево Б.

Третья функция идёт по следующему алгоритму:

Если первым элементом списка является «(», то мы удаляем этот символ и запускаем первую функцию. В результате работы которой мы получаем новое поддерево. Т.к. программа завершит свою работу ровно перед незнакомым символом (в нашем случае символом «)»), мы можем просто удалить закрывающий символ и вернут получившееся поддерево.

В ином случае мы копируем первый элемент списка, и, если он подходит по нашим критериям (является числом или переменной), мы удаляем оригинал начала списка и возвращаем поддерево, созданное из сохранённой ранее копии.

По завершению всех рекурсий в списке мы получаем готовое дерево с корнями в виде знаков и листьями в качестве переменных. После этого запускается специальная функция, проходящая по ветвям дерева и, если количество ветвей вглубь превысит заданное значение в 10 элементов, возвращает ошибку. Если этого не происходит запускается функция «Интерфейс», содержащая следующие команды:

1. Выйти из приложения;
2. Поиск результата данного нам выражения;
3. Изменение значения одной переменной в выражении, на выбор;
4. Перевод формул формата (f1 \* f3 ± f2 \* f3) в формат ((f1 ± f2) \* f3);
5. Вывести дерево.

Реализация функций:

Функция под номером «1».

Данная функция сначала рекурсивно спускается до конца одного из поддеревьев, после чего начинает возвращать результат арифметических выражений, что состоят из двух листов в качестве числовых элементов и корнем в качестве знака действия (левое поддерево, знак, правое поддерево). Такое действие повторяется пока мы не вернёмся к первому корню. По завершению этой функции мы выводим результат всех арифметических выражений.

Функция под номером «2».

В первую очередь данная функция спрашивает пользователя какую переменную он хочет изменить и выводит полный список всех доступных переменных. Если переменных нет в выражении, он выводит пустой лист. В любой момент пользователь может написать слово «exit» и вернуться на экран интерфейса. Как только пользователь выберет изменяемую значение переменной в словаре.

Функция под номером «3».

Проверяем, являются ли следующие поддеревья пустыми ячейками (равными None) и, если равняются, возвращаем нынешнее дерево.

В ином случае проверяем соответствует ли нынешнее дерева критериям оценивания (корень равен «+» или «–», а поддеревья «\*»). Если да, то проверяем является ли значения поддеревьев пустыми ячейкам. В случае успеха возвращаем 0. После проверки на наличие чисел, проверяем, есть ли в поддеревьях совпадающие переменные и, если нет, возвращаем нынешнее дерево. В обратном случае, создаём новое дерево формата ((f1 ± f2) \* f3) на месте нынешнего дерева. Вне зависимости от всех предыдущих действий, мы запускаем рекурсию по соседним поддеревьям и возвращаем нынешнее дерево.

Функция под номером «4».

Спускаемся сначала в левое поддерево до конечного элемента.

Выводим значение.

Возвращаемся и спускаемся в правое поддерево.

Выводим значение.

Повторять пока не будут выведены все значения.

В случае каких-либо неполадок по ходу кода выходят исключения, объясняющие пользователю, что он сделал не так.

# Тестирование

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Входные данные | Ожидаемая выражение | Ожидаемый результат обработки | Выражение | Результат обработки |
| (10\*3)+(33\*3) | ((((10)\*(3))+((33)\*(3))) | ((3)\*((33)+(10))) | ((((10)\*(3))+((33)\*(3))) | ((3.0)\*((33.0)+(10.0))) |
| (10\*3)+(3.3\*3)+(3\*13) | (((10)\*(3))+((3.3)\*(3))+((3)\*(13) | **(((10.0)\*(3.0))+((3.0)\*((13.0)+(3.3))))** | **(((10)\*(3))+((3.3)\*(3))+((3)\*(13))** | **(((10.0)\*(3.0))+((3.0)\*((13.0)+(3.3))))** |
| 3+(10\*3)+((3.3\*3)+(3\*13)) | ((3.0)+(((10.0)\*(3.0))+(((3.3)\*(3.0))+((3.0)\*(13.0))))) | ((3.0)+(((10.0)\*(3.0))+((3.0)\*((13.0)+(3.3))))) | ((3.0)+(((10.0)\*(3.0))+(((3.3)\*(3.0))+((3.0)\*(13.0))))) | ((3.0)+(((10.0)\*(3.0))+((3.0)\*((13.0)+(3.3))))) |
| 20(10\*3) | "Уравнение не может быть решено, исправьте орфографические ошибки." | "Уравнение не может быть решено, исправьте орфографические ошибки." | (20) | (20) |
| (-100\*3)-(45\*-100)) | (-100\*3)-(45\*-100)) | (-100\*3)-(45\*-100)) | (-100\*3)-(45\*-100)) | (-100\*3)-(45\*-100)) |
| (((1\*20)+(2\*20))\*((3\*15)+(3\*20)))-(2\*15)+(15\*20) | (((((1.0)\*(20.0))+((2.0)\*(20.0)))\*(((3.0)\*(15.0))+((3.0)\*(20.0))))-(((2.0)\*(15.0))+((15.0)\*(20.0)))) | ((((20.0)\*((2.0)+(1.0)))\*((3.0)\*((20.0)+(15.0))))-((15.0)\*((20.0)+(2.0)))) | (((((1.0)\*(20.0))+((2.0)\*(20.0)))\*(((3.0)\*(15.0))+((3.0)\*(20.0))))-(((2.0)\*(15.0))+((15.0)\*(20.0)))) | ((((20.0)\*((2.0)+(1.0)))\*((3.0)\*((20.0)+(15.0))))-((15.0)\*((20.0)+(2.0)))) |
| 1500/(10+40) | ((1500.0)/((10.0)+(40.0))) | ((1500.0)/((10.0)+(40.0))) | ((1500.0)/((10.0)+(40.0))) | ((1500.0)/((10.0)+(40.0))) |
| (10/-2022)+((1.11\*-10)-(-10\*100.553)) | (((10.0)/(-2022.0))+(((1.11)\*(-10.0))-((-10.0)\*(100.553)))) | (((10.0)/(-2022.0))+((-10.0)\*((100.553)-(1.11)))) | (((10.0)/(-2022.0))+(((1.11)\*(-10.0))-((-10.0)\*(100.553)))) | (((10.0)/(-2022.0))+((-10.0)\*((100.553)-(1.11)))) |

По итогам тестов мы можем прийти к выводу, что в 7 из 8 случаев программа выдаёт корректные данные. Последний случай является некорректным вводом пользователя и не предусматривался в постановке задачи.

# Код программы

class Node:#Главный класс, описывающий дерево.

def \_\_init\_\_(self, key,left=None,right=None):#Инициализация

self.left = left

self.right = right

self.val = key

def \_\_str\_\_(self):#Вывод

return str(self.val)

def uRavn(one,two,znack):#Функция для проведения алгебраических расчётов. получает два обекта операции и знак

if type(one)==Node:#Если первый обект является поддеревом, мы работаем только с его значением

one=one.val

if type(two)==Node:#Если второй обект является поддеревом, мы работаем только с его значением

two=two.val

if one in Edict.keys():#Если первый обект является переменной, мы работаем только с его значением

one=Edict[one]

if two in Edict.keys():#Если второй обект является переменной, мы работаем только с его значением

two=Edict[two]

if znack=="+":#Операция сложения

return float(one)+float(two)

elif znack=="-":#Операция вычитания

return float(one)-float(two)

elif znack=="\*":#Операция умножения

return float(one)\*float(two)

elif znack=="/":#Операция деления

return float(one)/float(two)

return None#Запосной вариант

def printList(L):#Вывод списка

for i in L:

print(i,end=" ")

print()

def negative\_Number(I):#Проверка, является ли передоваемый символ отрицательным числом

for i in range(10):

if str(I)==("-"+str(i)):

return True

return False

def join(L):#Данная функця обеденяет все цифры, стоящие друг рядом с другом в одно число

N=len(L)#Получаем длинну списка

i=0

temp=''

while(i< N):#Цикл кончается когда будет достаточно сокращён и все числа проверяны

if str(L[i]) in ("1234567890.") or negative\_Number(L[i]):#Если элемент списка это число или отрицательное число, сделать следующее:

try:

while (str(L[i]) in ("1234567890.") or negative\_Number(L[i])) and i<N:#Повторять пока элемент всё ещё число или мы не дошли до конца

temp+=str(L.pop(i))#Записать цыфру во временную строку

N-=1#Сокращаем длинну списка

except IndexError:#Если список внезапно закончился

L.insert(i,float(temp))#Вставить содержимое временной строки в список

return L#Вернуть список

L.insert(i,temp)#вставить временную строку на выбранную позицию списка

try:

L[i]=float(L[i])#преоброзовать вставленое число из String в Float

except ValueError:#Если число не преобразуется, вывести сообщение о ошибке

print(L[i]," это число невзможно воспринять. Исправте входные данные и перезапустите программу" )

return None

temp=''#Стираем временную строку

N+=1#Добовляем новую длинну

i+=1#Перейти к следующему элементу

return L#Вернуть список

def readVar(S):#Генератор, разделяющий строку на список

j=''#Спец переменная

temp=False

for i in S:#Пройтись по элементам строки

if temp == True and i in"1234567890":#Если прошлый символ был "-"

j+=i#Добавить цифру

temp=False#Возвращаемся к изначальным настройникам

yield j#Вернуть отрицательную цифру

continue#Продолжить

elif temp==True:

temp=False

yield j

if (j in "+-\*/()") and i=='-':#Если перед символом "-" был другой символ, не являющийся цифрой

temp=True#Передаём в переменную True

j=i#Запоминаем символ

continue#Продолжить

if i!=" " and i !="\n":#Если символ не является пустым, то:

j=i#Запоминаем символ

yield i#Вернуть символ

def get\_T(List, exp):#Проверка на соответствие символа

try:

if List[0]==exp:

List.pop(0)

return True

except:

return False

return False

def get\_N(List):#Функция вставления нового числа в дерево

if get\_T(List,"("):#Если в формуле находится скобка

x=get\_S(List)#Работта с функцией суммы

get\_T(List,")")#Убать конечную скобку

return x #Вернуть полученное ранее поддериво

else:

x=List[0]#Получить нынешнее значение

if type(x) != float and type(x) != int and not(x in Edict.keys()):#Если значение не соответствует заданым типам и не содержится в словаре переменных — Вылетаем.

return None

List[0:1]=[]#Убрать нынешнее число

return Node(x)#Вернуть число в виде поддерева

def get\_P(List):#Работа с произведением

a=get\_N(List)#Получаем числовое поддерево

if get\_T(List,"\*"):#Если дальше мы встречаем символ "\*"

return(Node("\*",a,get\_P(List)))#Возвращаем поддерево состоящее из знака умножения, найденого ранее поддерева и результата рекурсии

elif get\_T(List,"/"):#То же самое, но с делением

return(Node("/",a,get\_P(List)))

return a#Если нет ни знака умножения, ни знака деление, возвращаем найденое ранее поддерево

def get\_S(List):#Работа с суммой

a=get\_P(List)#Получаем поддерево от функции произведения, описаной выше

if get\_T(List,"+"):#Если следующим символом является знак "+"

return(Node("+",a,get\_S(List)))#Вернуть поддерево из знака "+", найденого ранее поддерева и результата рекурсии

elif get\_T(List,"-"):#То же, но для знака "-"

return(Node("-",a,get\_S(List)))

return a#Вернуть найденое поддерево

def printTree(T):#Вывод дерева путём рекурсии

if(T==None):

return 0

print("(",end=" ")

printTree(T.left)

print(T.val,end=" ")

printTree(T.right)

print(")",end=" ")

def Solve(Tree):#Получение результата нашего выражения

if Tree.left!=None or Tree.right!=None:#Если у нас есть хоть одно поддерево мы возвращаем

return uRavn(Solve(Tree.left),Solve(Tree.right),Tree.val)#Результат арифмитической операции от результата арифметических операции левого и правого поддеревьев

return Tree#Если поддеревьев нет — вернуть нынешнее дерево

def down\_Tree(T, temp=0):#Спуск по дереву

temp+=1#Проверка количества посещённый поддеревьев (действий в скобках)

if temp>10:#Если их больше 10,возвращает ошибку

raise NameError('Слишком много действий')

if T==None:

return 0

else:#Спуск по поддеревьям

down\_Tree(T.left,temp)

down\_Tree(T.right,temp)

def change(Tree):#Изменение значение переменных

print("Выберите переменную из списка или введите exit, чтобы выйти: ")

printList(Edict.keys())#Вывести список переменных

temp=True

while temp==True:#Повторять, пока не будет введена переменная, существующая в списке

try:

x=input()

if x in Edict.keys():

temp=False

elif x=="exit":#Выход

return 0

else:

print("Список не содержит данной переменной")

except UnicodeDecodeError:#Ошибка связанная с кодировкой, редкая.

print("Что-то случилось с нашей стороны, пожалуйста, повторите на английском.")

print("Перезапишим ваш элемент")

Edict[x]= in\_nomber(x)

def in\_nomber(I):#Вводим значение переменной

while True:#Выход только через return. Необходимое зло.

try:

return float(input(f"Введите числовое значение переменной { I }:"))

except ValueError:#Если введено что-то, не переводящееся в цифры, обявить об этом

print("Данное значение не может быть записано. Попробуйте ещё раз. На этот раз введите число.")

def Interface(T,x="4"):#Интерфейс

while x!="0":#0 — Выход

if x=="1":#1 — Сумма

print("Результат: " ,Solve(T))

elif x=="2":#2 — Изменение значения переменной

change(T)

elif x=="3":#3 — перевод формул формата (f1\*f3±f2\*f3) в формат ((f1±f2)\*f3)

try:

T=Simplification(T)

except TypeError:#Если нет искомых формул

print("Здесь нечего упрощать")

printTree(T)

print()

elif x=="4":#4 — Вывыести дерево

print("Ваша формула:")

printTree(T)

print()

else:

print("Такой команды нет!")

x=input("\nВыберите команду:\n0 — Выйти;\n1 — Вычислить формулу;\n2 — Изменить значение переменон;\n3 — Упростить формулу;\n4 — Вывести формулу;\n")

def Act(T1,T2):#Поиск совпадающих элементов в двух деревьях

if T1.left.val==T2.left.val:

return T1.left,T2.right, T1.right

elif T1.left.val==T2.right.val:

return T1.left,T2.left,T1.right

elif T1.right.val==T2.left.val:

return T1.right,T2.right,T1.left

elif T1.right.val==T2.right.val:

return T1.right, T2.left,T1.left

return 0,0,0

def Simplification(T):#Перевод формул формата (f1\*f3±f2\*f3) в формат ((f1±f2)\*f3)

if T.left== None or T.right==None:#Если поддеревьев нет, вернуть нынешнее дерево

return T

if (T.val =="+" or T.val=="-") and T.left.val=="\*" and T.right.val=="\*":#Если дерево соответствует формуле (f1\*f3±f2\*f3) в люьой её ипостасии

if T.left.left==None or T.left.right==None or T.right.left==None or T.right.right==None:#Если хоть одного числа в поддереве не существует, вернуть нынешнее дерево

return T

else:#Если все условия соблюдены

f3,f1,f2=Act(T.left,T.right)#Вернуть числа поддеревьев

if f3==0 and f1==0 and f2==0:#Если совпадений не найдено, вернуть нынешнее дерево

return T

if T.val=="+":#Преобразуем требуемую формулу ((f1±f2)\*f3)

T=Node("\*",f3,Node("+",f1,f2))

else:#То же, но со знаком отрицания

T=Node("\*",f3,Node("-",f1,f2))

T.left=Simplification(T.left)#Повторить на поддеревьях

T.right=Simplification(T.right)

return T#Вернуть нынешнее дерево

Edict=dict()#Создаём словарь и список

Exp=[]

with open("File.txt","r") as var:#Считывем данные и файла

text=var.readline()

for i in readVar(text):#Проходимся по элементам прочитаного списка

try:

Exp.append(int(i))#Добавить цифру

except ValueError:

if not(i in "1234567890+-\*/().") and not(i in Edict.keys()) :

Edict[i]=in\_nomber(i)#Добавить значение переменной

Exp.append(i)#Добавить симвл

join(Exp)#Обединить стоящие рядом цифры

try:

Tree=get\_S(Exp)#Построенние дерева

down\_Tree(Tree)#Проверка глубины дерева (не более 10 спусков)

Interface(Tree)#Запуск интерфейса

except AttributeError:#Обработка ошибки

print("Уравнение не может быть решено, исправте арфаграфические ошибки.")

except NameError:#Обработка персональной ошибки

print("Превышен предельный лимит количества элементов на одну скобку в 10 действий. Напоминаю, что скобкой считаются все связаные действия")

except:

print("Мы без понятия, что только что произошло. Попробуйте выключить и включить или изменить входящую формулу")