Разбор арифметических выражений с использованием бинарных деревьев(практикум)

Проблема вычисления арифметических выражений, между пользователем, с одной стороны, и компьютером, с другой стороны, возникает при решении многих прикладных задач, например при разработке универсальных программ, решающих задачи вычислительной математики (поиск корней функции, вычисление определенного интеграла и т.п.).

Одним из способов разбора выражения является представление его в виде дерева, каждый поддерево которого обозначает одну операцию умножения в порядке убывания приоритета операций. В корне такого дерева хранится знак операции, а каждый поддерево представляет собой выражение вида:

$$(x + 1,5)(x - 10)^{(x+5)/(x+3)}$$

может быть представлено деревом вида, изображенным на рис. 7.21 (каждая в скобках обозначена символом $x^{\wedge}$/).

Построение дерева выражения выполняется следующим образом:

1) в сомнительной строке, содержащей запись выражения, определяется положение операции с минимальным приоритетом, записанной вне особых (если строка содержит несколько операций с минимальным приоритетом, то для вождения в степень выбирается первая, а для остальных — последняя);

2) операция записывается в корень дерева, а строка делится на две подстроки левую операнд и подстроки второго операнда. При этом по необходимости выполняется операция снятия скобок;

3) в подстроках операнда вновь определяется положение операции с минимальным приоритетом и т.д.

Если значения переменных шаги подстроки не содержат операций, то процесс заканчивается, а подстроки записываются в виде операнда очередного узлера в качестве листа.

Листы такого дерева могут быть двух типов: лист, содержащий имя переменной, а лист, содержащий значение константы. Вычисление выражения выполняется рекурсивно:

• если вершина — лист, то в качестве результата подставляется значение константы или переменной;

Рис. 7.22. **Представление в виде бинарного дерева выражения, содержащего элементарные функции**

Рис. 7.21. **Представление выражения в виде бинарного дерева**

• если вершина — операция, то вычисляется левый операнд, потом правый операнд, а затем над ними выполняется операция.

Примеры работы с деревом выражения почти можно допустить, что, чтобы обеспечить вычисления выражения, каждый узел дерева, кроме листа, должен содержать ссылку на функцию, вычисляющую в соответствующей вершине дерева, а аргумент представлять в виде левого подерева. Вершинам, которые являются листами, будет присваиваться в виде левого подерева. Вершинам, которые являются листами, будет представлено дерево, изображенное на рис. 7.22.

Пример 7.7. Разработать программу построения таблицы значений функции одного аргумента, определенной пользователем. Чтобы не усложнять и без того сложную программу, не будем предусматривать операции «унарный минус».

В первом разделе определим структуру информационной части записи, соответствующей вершине дерева: каждая вершина дерева хранить знак операции, адреса операндов и для констант — значение константы, причем для листьев в качестве знака операции будет хранить признак типа листа: «num» — константа или «var» — переменная.

Процедура Const_Tree конструирования дерева из выражения по сути рекурсивна: она последовательно должна разбивать строку выражения на отдельные операции и строить соответствующие подпрограммы. Параметры процедуры: адрес корня дерева / и строка выражения. Для выполнения поставленной задачи в процедуре Const_Tree необходимо осуществить поиск рассматриваемого знака операции: сначала наивного уровня приоритета, затем все более высокого. Этот поиск целесообразно реализовать как внутренние функции, которые будут получать значение поиска операции Ser_of в строку st.

Вычисление выражения по дереву также будет выполнялось рекурсивной функцией Count. Эта функция в качестве параметров будет получать значение адрес корня дерева и значение x. При вычислении учтем, что выполнение некоторых операций, например деление на ноль, получение логарифма отрицательного числа и т.п., невозможно, в этом случае Count будет возвращать значение параметра Key-false.

```
Program ex;
{$APPTYPE CONSOLE}
Uses SysUtils;
Type cstCharset=of char; {тип множества СИМВОЛОВ}
str80=string [80]; {тип строка длиной 80 символов}
pTop=^Top; {тип «указатель на вершину»}
Top=record {тип «вершина»}
operator: string [5]; {знак операции}
value: single; {значение константы}
left, right: pTop; {указатели на левое и правое подерево}
end;
Var st: str80; {строка — запись выражения}
Root:pTop; {корень дерева выражения}
key,bookvar: {переменная (значение значения в заданной точке) x, xn, xk, dx, y: single;
{начальное, конечное значения и шаг}
argument, {значение аргумента и функции} n, i: word; {количество точек и номер текущей точки}
{рекурсивная функция конструирования поддерева выражения с корнем / из строки st}
Procedure Const_Tree(p:pTop; st:str80);
Var nstr,p1:pTop;
SetOpisSetChar;
ps,code:integer;
st,str:str80; c:single;
{внутренние функции поиска первого вхождения операции в строке st: SetOp — множества знаков операций; функция возвращает позицию разделительного знака ent }
Function PosOpFirst(str:str80; SetOp:setCharType;
Var L,k:byte; begin
j:=0; k:=0; p:=0; {+1; while(l<length(s)) and p=0 do begin
if s[l] = ' ' then inc(j) {считываемые/не
открывающиеся скобки}
else
if (j = 0)' then inc(k) {считываемые/не
закрывающиеся скобки}
else
if(k=0) and(s[l] in SetOp) then p:=l; inc (l); end;
PosOpFirst:=p end;
{внутренние функции поиска последнего вхождения операции в строке st: SetOp — множества знаков операций; функция возвращает позицию разделительного знака ent }
Function PosOpLast(str:str80; SetOpisSetChar;
Var L,k:byte; begin
j:=0; k:=0; p:=0; for l:=1 to length(s) do begin
if s[l] = ' ' then inc(j) {считываемые/не
открывающиеся скобки}
else
if s[l] = ')' then inc(k) {считываемые/не
закрывающиеся скобки}
else if(k=0) and(s[l] in SetOp) then p:=l; inc (l); end;
PosOpLast:=p; end;
{создает оператором конструирования дерева выражения}
Begin
pc:=PosOpLast (st, [' +', ' /']) {ищем знак операции + или -}
if p=0 then
pc:=PosOpLast (st, [ищем знак операции * или /}
if p=0 then pc:=PosOpFirst( [' ^']); {ищем знак
операции ^}
if p=0 then {разделяющий знак найден}
begin
r^.operator :=s[pc]; {запись/сьмем знак операции
в вершину}
st :=copy (st, 1,pc-1); {копируем подструку первого
операнда}
if (st[1] = '(') and
(PosOpLast(st,[')'])=Length(st)) {выбрасываем скобки}
st:=copy(st,2,length(st)-2)
{копируем подструку второго операнда} str:=copy(st,pc+1,length (st)- pc); if (str[1] = '(') and
(PosOpLast(str,[')'])=Length (str)) {выбрасываем скобки}
str:=copy(str,2,length(str)-2); new(r^.left); {создаем левое поддерево}
Const_Tree (r^.left, st); {конструируем левый операнд new (r^. right); {создаем правое поддерево}
Const_Tree(r^.right,str); {конструируем правый
операнд}
end
else
{аргумент}
if st[1]='x' then begin
r^.operator:='x'; r^.left:=nil; r^.right:=nil end else begin val (st, c, code); {пытаемся получить
число}
if code=0 then {константа}
begin
r^.operator:='c'; r^.left:=nil; r^.right:=nil; r^. Value:=c; end
else {функция}
begin
pc:=Pos('(',st) ;
{выделяем имя функции}
r^.operator:=copy(st,1,pc-1); r^.right:=nil;
{вешаемое подпрограмму параметра}
st:=copy(st,pc+1,length(st) -pc-1); new(r^.left);
{конструируем параметр}
Const_Tree(r^.left,st); end;
end;
end;
{рекурсивная вычисление значения функции: если Key-false, то значение не существует}
Function Count(p:pTop; x:single; Var key,bookvar:single;
Var s,c:single; begin
if not key then {значение функции не существует}
begin
Count:=0; exit; end;
if r^.operator='c' then
Count:=r^.Value {константа}
else
if r^.operator='x' then
Count :=x {переменная x}
else
case r^.operator[1] of
'+':Count:=Count(r^.left,x,key) +
Count(r^.right,x,key);
Count:=Count(r^.left,x,key)- Count(r^.right,x,key);
'*':Count:=Count(r^.left, x, key)* Count(r^.right,x,key); begin
{практический ноль}
s:=Count(r^.left,x,key); if abs(s)
Count:=0; key:=false; end else
Count:=Count(r^.left,x,key)/s;
end;
'^':begin
s:=Count(r^.left,x,key); c:=Count(r^.right,x,key); if s<>0 then
Count:=exp(c*ln(abs(s)));
if s=0 then Count:=0 else Count:=0;
end;
'/':Count:=Count(r^.left,x,key)/ Count(r^.right,x,key); else {неопределенная операция} begin
Count:=0;
Key:=false; end end;
{основная программа}
Begin
Writeln,('Enter string');
Readln(st);
Write('Enter xn,xk,n');
Readln(xn,xk,n); new(Root);
Const_Tree(Root,st); dx:=(xk-xn)/(n-1);
Writeln,('x ', 'y');; x:=xn;
for i=1 to n do begin
key:=true;
y:=Count(Root,x,key);
if key then
Writeln,(x:5:3,y:20:3) else Writeln,(x:5:3,' does not exists.'); x:=x+dx; end;
Readln;
End;
```

Задания для самопроверки

1. Разработайте программу, которая обеспечивает быстрый поиск информации о сотрудниках фирмы с использованием бинарных деревьев: имя, фамилия, отчество, отдел, должность, служебный телефон, домашний адрес и телефон. Определите, во сколько раз быстрее будет выполняться поиск информации по сравнению с последовательным поиском (см. параграф 4.6), если количество сотрудников фирмы составляет 10, 100, 1000 человек.

2. Для программы, предыдущего задания оцените объем оперативной памяти, необходимый для хранения информации о сотрудниках фирмы. Определите долю памяти, занятой для хранения адресов элементов.

3. Дополните программу разбора выражений, чтобы с ее помощью можно было обрабатывать выражения, содержащие стандартные математические функции sin x, cos x, lg x, ln x, arcsin x, arctan x, e^x и т.д.

4. Дополните программу разбора выражений возможностью обрабатывать унарный минус.

5. Модернизируйте программу разбора выражений (пример 7.7) так, чтобы она могла разбирать выражения с двумя переменными.

Посмотреть оригинал

« Предыдущая СОДЕРЖАНИЕ Следующая »

Арифметические выражения.
Арифметические выражения всех математических языков (Паскаль, Бейсик, Фортран, Алгол, Лисп, Си, Ассемблер) Таблица 1.3. Примеры арифметических выражений в языках МАТЛАБ. Арифметические выражения в Паскале
(Информатика и строительство)

Вычисление арифметических выражений
Вычисление арифметических выражений в языке МАТЛАБ (за 10 минут) Вычисление выражения — сложная последовательность операций. Правила выражения лабораторной работы. Требуется выполнить следующие операции: Вычисление арифметических выражений. Для выполнения лабораторной работы необходимо знать: Ввод и вывод чисел. Арифметические операции. Приоритет операций. Скобки.
(Информатика и строительство)

Арифметические выражения.
Арифметические выражения. Арифметические выражения всех языков программирования выражения могут являться арифметическими и представляющие собой переменные, константы, вызовы функций, разделенные операциями и скобками. Арифметические выражения.
(Программирование)

АРИФМЕТИЧЕСКИЕ, СТРОКОВЫЕ И ЛОГИЧЕСКИЕ ВЫРАЖЕНИЯ

Арифметические выражения. Арифметические выражения. Арифметические выражения.