Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет непрерывного и дистанционного обучения

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе
на тему

ПРЕДСТАВЛЕНИЕ МАТЕМАТИЧЕСКИХ ФОРМУЛ В ВИДЕ БИНАРНЫХ ДЕРЕВЬЕВ.

БГУИР КП 1-40 01 01

Студент: Костина Н.В.

Руководитель: асс. Болтак С.В.

Минск 2017

Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий
кафедрой ПОИТ

_____

_____

(подпись)
Лапицкая Н.В. 2016 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Костиной Наталье Владимировне

1. Тема работы «Представление математических формул в виде бинарных деревьев»
2. Срок сдачи студентом законченной работы
3. Содержание расчётно-пояснительной записки

Введение.

1. Аналитический обзор;

2. Разработка программного средства;

3. Тестирование;

4. Руководство пользователя;

5. Заключение, список литературы, приложение.

5. Консультант по курсовому проекту Болтак С.В.
6. Дата выдачи задания 12.06.2016 г.

РУКОВОДИТЕЛЬ_____ С.В. Болтак

(подпись)

Задание принял к исполнению_____ 15.02.2016 г.

(дата и подпись студента)

Содержание

ВВЕДЕНИЕ

В настоящее время бинарные (двоичные) деревья являются одним из самых востребованных вариантов структуры данных. Это обусловлено тем, что они широко используются для создания поисковых алгоритмов, при анализе электрических цепей, представления математических формул и логических выражений, для организации информации в системах управления базами данных и для представления синтаксических структур в компиляторах программ.

Целью данной курсовой работы является разработка программы, которая представляет математическую формулу в виде бинарного дерева и реализует обходы построенного дерева.

В качестве среды разработки приложения была выбрана система объектно-ориентированного приложения BorlandDelphi7.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Понятие бинарного дерева

Дерево представляет собой иерархическую структуру какой-либо совокупности элементов (узлов) и отношений, образующих иерархическую структуру узлов. [1, с. 29] Каждый узел дерева является корнем некоторого поддерева. В том случае, когда множество поддеревьев такого корня пусто, этот узел называется листом.

Бинарным называют дерево, состоящее из узлов, каждый из которых содержит, кроме данных, не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка.

1.2 Построение бинарного дерева по арифметической формуле

Любую арифметическую формулу, которая содержит переменные, знаки операций и скобки можно представить в виде бинарного дерева, в котором знак операции помещается в корне, первый операнд – в левом поддереве, а второй операнд – в правом. Скобки при этом опускаются. В результате все переменные оказывается в листьях, а знаки операций – во внутренних вершинах.

Бинарное дерево по арифметической формуле строится с учетом правила приоритетности операций: построение корня дерева либо поддерева начинается с последней операцией наименьшего приоритета. Под приоритетом операции понимается порядок выполнения операций в выражении – первыми выполняются операции с более высоким приоритетом. Выражения в скобках имеют наивысший приоритет.

В таблице 1.1 приведены операции и соответствующий им приоритет.

Таблица 1.1

Операция	Приоритет
^	3
*, /	2
+, -	1

Построение бинарного дерева по выражению выполняется следующим образом (подразумевается, что введенное выражение корректно):

а) в символьной строке, содержащей запись выражения, определяется положение операции с минимальным приоритетом, записанной вне круглых скобок (если строка содержит несколько операций одинакового приоритета, то выбирается последняя);

б) операция записывается в корень дерева, а строка условно делится на две части: первая часть содержит первый операнд, вторая – второй операнд, при этом по необходимости выполняется операция снятия скобок;

в) в каждой части вновь определяется положение операции с минимальным приоритетом и т.д.

Если полученные на очередном шаге части строки не содержат операций, то это значит, что осталась либо переменная, либо константа. Процесс заканчивается, а переменная либо константа записывается в вершины очередного уровня в качестве листьев. [2, с. 248]

Например, пусть дано следующее арифметическое выражение:

a*(b-c)-d/e

На рисунке 1.1 приведено бинарное дерево, построенное по данному выражению.

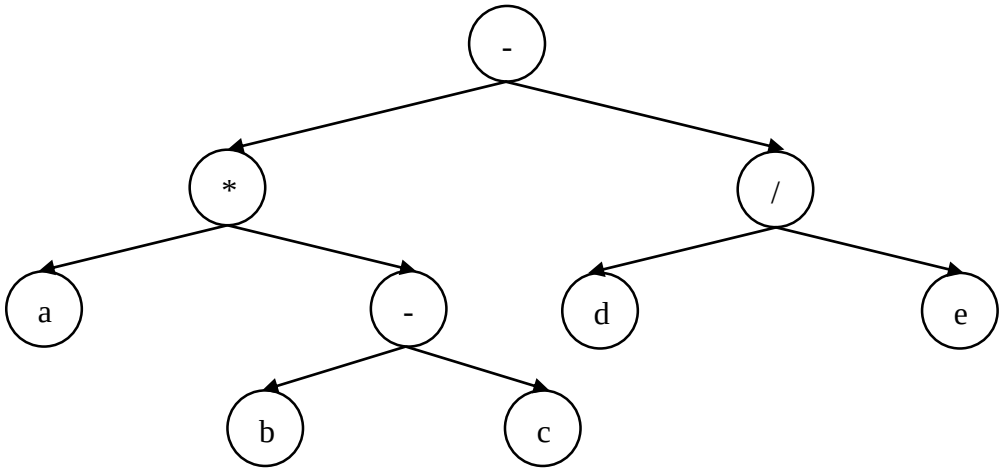


Рисунок 1.1

1.3 Обходы бинарного дерева

Для бинарных деревьев существуют несколько общепринятых способов обхода:

а) обход дерева в прямом порядке (сверху вниз);

б) обход дерева в симметричном порядке (слева направо);

в) обход дерева в обратном порядке (снизу вверх).

Три варианта обхода этого дерева порождают три известные формы записи арифметического выражения: префиксную, инфиксную и постфиксную запись соответственно.

Таблица 1.2 демонстрирует пример префиксной, инфиксной и постфиксной записей арифметического выражения a*(b-c).

Таблица 1.2

инфиксная запись	префиксная запись	постфиксная запись
a*b-c	*a-bc	abc-*

Обходы можно осуществлять как рекурсивным, так и нерекурсивным методом.

Рекурсивный метод заключается в том, что любое действие, выполняемое над вершиной, должно быть выполнено также и по отношению ко всем его поддеревьям, а значит, алгоритм должен быть рекурсивно выполнен по отношению ко всем потомкам этой вершины.

Нерекурсивный метод обходов использует стек, работающий по принципу LIFO (последним пришел, первым ушел), в который записываются узлы по мере их прохождения.

Для осуществления обходов было принято решение использовать рекурсивный алгоритм, так как, во-первых, он логически проще метода итерации, а во-вторых рекурсивно реализованные алгоритмы имеют более лаконичную запись.

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Организация входных и выходных данных

Входной информацией в данной программе является введенная арифметическая формула. При записи формулы допускается использование круглых скобок. Арифметические операции записываются обычным способом, то есть + означает сложение, - – вычитание, * – умножение, / – деление. Для возведения в степень используется символ ^, например, a^2.

Выходной информацией в зависимости от поставленной и решаемой пользователем задачи может являться:

а) построенное бинарное дерево;
б) графическое изображение построенного бинарного дерева;
в) результаты обходов, выведенные на экран;
г) текстовый файл с результатами обходов.

2.2 Структура узла бинарного дерева

Узел дерева является записью, состоящей из трёх полей: информационной части (data) и двух ссылок, указывающих на левое (left) и правое (right) поддеревья данного узла. В программе узел дерева реализован в следующем виде:

```
type
  PNode = ^Node;
  node = record
    data: string[5];
    left, right: PNode;
  end
```

2.3 Разработка структуры программы

Укрупненный алгоритм программы приведен на рисунке 2.1

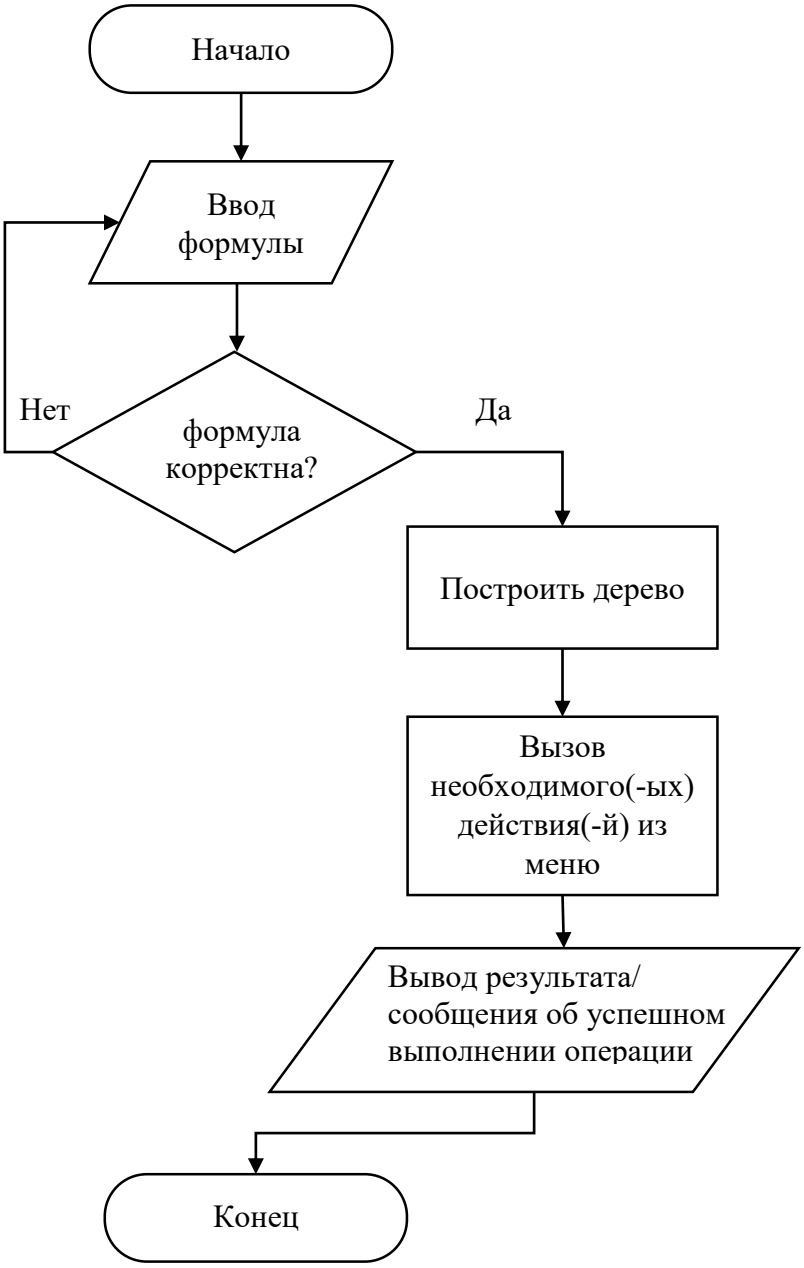


Рисунок 2.1

Для создания данной программы целесообразно использовать модульный принцип построения программы, то есть разбить программу на отдельные части, каждая из которых выполняет какую-то законченную задачу, либо ее малую часть. Таким образом, было решено создать следующие модули программы:

а) модуль Check – проверяет корректность введенного арифметического выражения;

б) модуль Tree – содержит процедуры построения бинарного дерева, вывода его на экран в виде графа и реализации обходов;

в) модуль Menu – содержит главную форму.

Более детальное описание модулей приведено в разделе 2.4.

2.4 Описание программных модулей

2.4.1 Модуль Unit Check

В первую очередь необходимо удалить лишние пробелы из формулы. Процедура probel (var s:string) на вход получает введенную формулу и возвращает эту же формулу, но без пробелов. Затем необходимо проверить выражение на корректность. Для выполнения этой задачи было принято решение построить конечный автомат, имеющий три допускающих состояния (0, 1 и 2 в программной реализации) плюс состояние 3, которое завершает проверку выражения. В каждом из трех состояний (0,1 и 2) допустимыми являются лишь некоторые символы. Наличие же на входе другого символа говорит о некорректности введенного выражения.

Стартовое нулевое состояние говорит о том, что выражение может начинаться с цифры, буквы или открывающей скобки. В первом случае следует пропустить число и перейти в состояние 1, во втором случае необходимо проверить, не введены ли подряд две и более буквы, если это так, то выражение некорректно. Открывающаяся же скобка оставляет автомат в том же состоянии.

В первом состоянии допустимыми являются лишь знаки арифметических операций и закрывающаяся скобка. Знак операции переводит автомат в состояние 0, а закрывающаяся скобка – оставляет его в состоянии 1. Если текущий символ – последний в анализируемой строке, то автомат переводится в состояние 2, допустимым для которого являются буквы, цифры и закрывающаяся скобка.

Остается только проверить, чтобы общее число открывающихся скобок было равно числу закрывающихся скобок, а при встрече очередной закрывающейся скобки общее их количество к этому моменту не может превышать количество уже встретившихся открывающихся скобок. Для этого вводится целочисленная переменная – счетчик скобок (в программе k), первоначально равная нулю. Если в выражении встретилась открывающаяся скобка (состояние 0), то к счетчику прибавляется единица, если закрывающаяся (состояние 1), то единица вычитается. Данная переменная по ходу вычислений не должна принимать отрицательных значений, а при завершении просмотра (состояние 2) должна быть в точности равна нулю. [3]

Принцип работы описанного автомата приведён в таблице 2.1.

Таблица 2.1

Состояние	Символ	Операция	Новое состояние
0	последний символ		2
	(	k := k + 1, перейти к следующему символу	0
	буква или цифра	пропустить букву или число, перейти к следующему символу	1
	остальные символы	ошибка, конец работы	
1	последний символ		2
	)	k := k – 1, проверить, что k ≥ 0, перейти к следующему символу	1
	+, -, *, /,^	перейти к следующему символу	0
	остальные символы	ошибка, конец работы	
2	), буква или цифра	если k = 0, то выражение корректно, в противном случае – ошибка, конец	
	остальные символы	ошибка, конец работы	

Работу приведенного автомата реализует функция Checks(s:string):boolean. На вход подаётся строка с формулой s. На выходе имеем значение «true», если формула корректна, и «false», если формула записана неправильно.

2.4.2 Модуль Unit Tree

Данный модуль работает с бинарным деревом: строит его, выводит на экран, осуществляет обходы.

Для построения бинарного дерева используется алгоритм, описанный в разделе 1.2. Реализация данного алгоритма сводится к выполнению следующих функций и процедур:

а) Функция Priority (c: char): integer отвечает за присвоение операции соответствующего приоритета. На вход подаётся символ c. Функция

возвращает значение приоритета. Если символ не является операцией, то функция возвращает значение «10».

б) Функция LastOperation (expr: string; first, last: integer): integer. На вход получает строку expr, номера первого first и последнего last символов строки. Функция возвращает номер символа, который является последней операцией.

При поиске последней операции с наименьшим приоритетом необходимо учесть скобки. Выражение в скобках имеет высший приоритет, поэтому при поиске операции с минимальным приоритетом его не надо брать во внимание. Самый простой способ добиться этого – ввести счетчик открытых скобок (ch). В начале он равен нулю, с каждой найденной открывающей скобкой он увеличивается его на единицу, а с каждой закрывающей – уменьшать на единицу. Рассматриваются только те операции, которые найдены при ch = 0, то есть расположены вне скобок.

Алгоритм функции LastOperation представлен на рисунке 2.2.

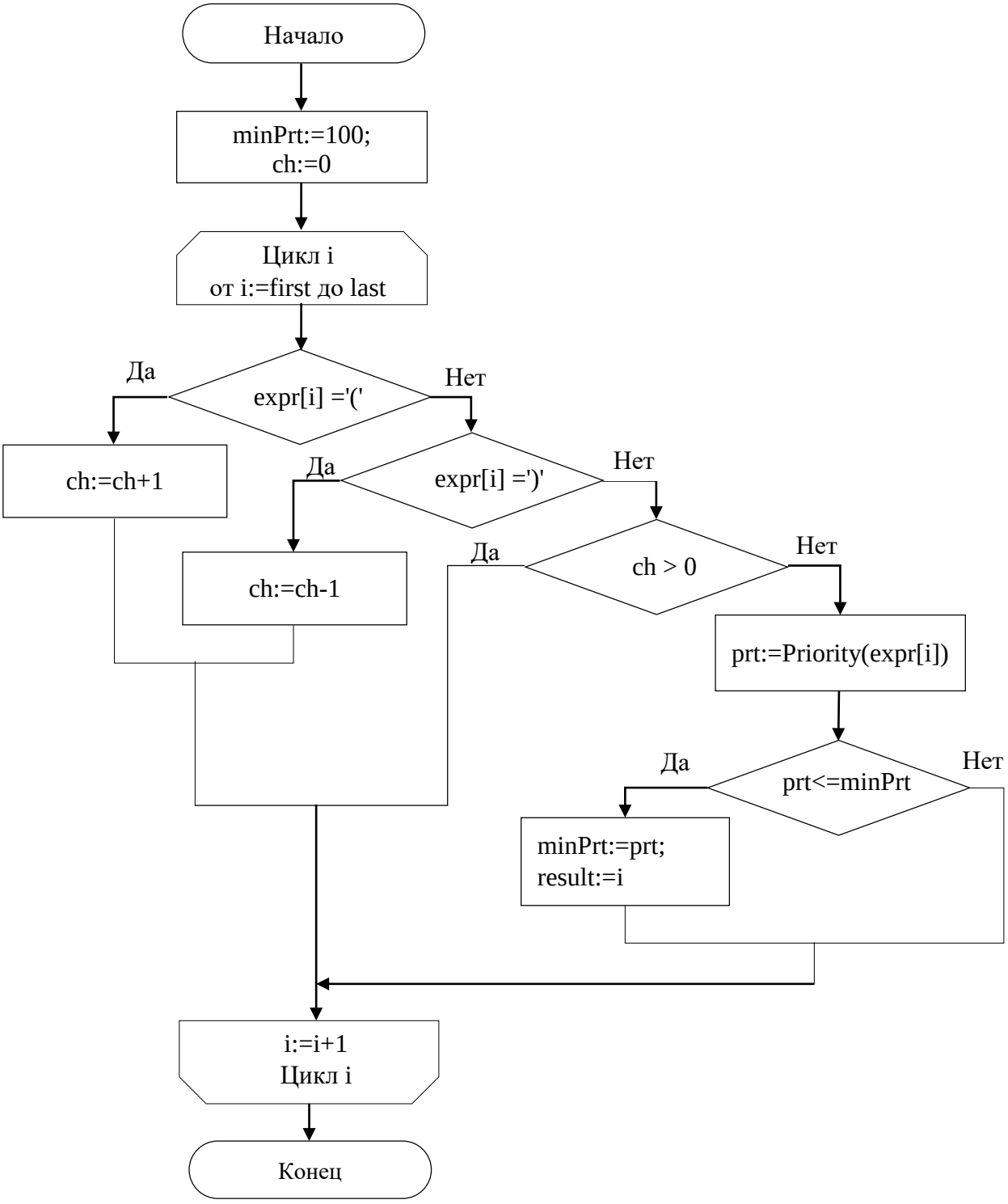


Рисунок 2.2

в) Процедура MakeTree (r: Pnode; expr:string; first,last: integer) преобразовывает арифметическое выражение в бинарное дерево. Процедура является рекурсивной. На вход получает указатель на корень дерева r, строку expr, порядковый номер первого first и последнего last символа. Ниже приведен алгоритм процедуры MakeTree:

1) Если first = last (остался один символ), то создать новый узел и записать в него этот элемент. Иначе перейти к шагу 2.

2) Если first <> last (осталось более одного символа), то определить в строке положение операции с минимальным приоритетом при помощи функции LastOperation. Перейти к шагу 3.

3) Если в результате поиска минимальный приоритет приобретает значение 100 (выражение заключено в скобки), то снять скобки (first:=first+1; last:=last-1) и вновь вызвать функцию LastOperation. Иначе перейти к шагу 4.

4) Если в результате поиска минимальный приоритет приобретает значение не равное 100 (выражение не заключено в скобки) и если минимальный приоритет не равен 10 (выражение не является числом), то записать найденную операцию в узел r, создать левое поддерево и рекурсивно применить к нему этот алгоритм, разобрав выражение из элементов массива с номерами от first до k–1, создать правое поддерево и рекурсивно применить к нему этот алгоритм, разобрав выражение из элементов массива с номерами от k+1 до last. Если минимальный приоритет равен 10 (осталось многозначное число), то записать число в узел r и присвоить правому и левому поддереву значение nil.

г) Процедура DrawTree (p:PNode; l,c,r,y:integer) рисует дерево в виде графа на компоненте Field формы Form1 при помощи свойства Canvas. Процедура рекурсивна: сначала она рисует текущий узел, а затем рекурсивно вызывает себя для каждого из дочерних узлов. Вход – адрес корня дерева p, левая граница поля l, горизонтальная позиция центра круга c, правая граница поля r, вертикальная позиция центра круга y. Для вывода дерева на экран необходимо, чтобы родительский узел был центрирован относительно дочерних узлов. За определение центра круга отвечают функции Center и min.

д) Процедура Preorder (p:PNode; var q:string) осуществляет прямой обход дерева. На вход подаётся адрес корня дерева p и параметр-переменная q, которая хранит результат обхода. Алгоритм процедуры Preorder:

1) Обращение к узлу.
2) Рекурсивный прямой обход левого поддерева.
3) Рекурсивный прямой обход правого поддерева.

е) Процедура Postorder (p:PNode; var q:string) осуществляет обратный обход. На вход подаётся адрес корня дерева p и параметр-переменная q, которая хранит результат обхода. Алгоритм процедуры Postorder:

1) Рекурсивный обратный обход левого поддерева.

2) Рекурсивный обратный обход правого поддерева.
3) Обращение к узлу.

ж) Процедура Inorder (p:PNode; var q:string) осуществляет симметричный обход. На вход подаётся адрес корня дерева p и параметр-переменная q, которая хранит результат обхода. Алгоритм процедуры Inorder:

1) Рекурсивный симметричный обход левого поддерева.
2) Обращение к узлу.
3) Рекурсивный симметричный обход правого поддерева.

2.4.3 Модуль Unit Menu

Данный модуль содержит следующие глобальные переменные:

Form1: TForm1 – форма;

s: string – строка, которая содержит введенное арифметическое выражение;

q: string – хранит результат обхода;

root: PNode – указатель на корень дерева;

f: textFile – текстовый файл с результатами обходов.

Для создания формы, которая будет удовлетворять поставленные в условии задачи, необходимы:

а) поле для ввода арифметической формулы;

б) набор компонентов, составляющих меню для вызова следующих основных процедур:

1) вывод полученного дерева на экран;
2) симметричный обход и вывод полученного выражения на экран;
3) прямой обход и вывод полученного выражения на экран;
4) обратный обход и вывод полученного выражения на экран;
5) запись результатов обходов в файл.

в) один компонент для визуального представления бинарного дерева в виде графа;

г) три поля для вывода результатов симметричного, прямого и обратного обходов;

д) компонент для очистки полей с формулой и результатами обходов. компонент используется перед началом работы с новой формулой;

е) компонент для выхода из программы;

ж) диалоговые окна для сообщения пользователю об успешном выполнении процедуры либо об ошибке.

Таким образом, было принято решение создать визуальные компоненты, которые представлены в таблице 2.2.

Таблица 2.2

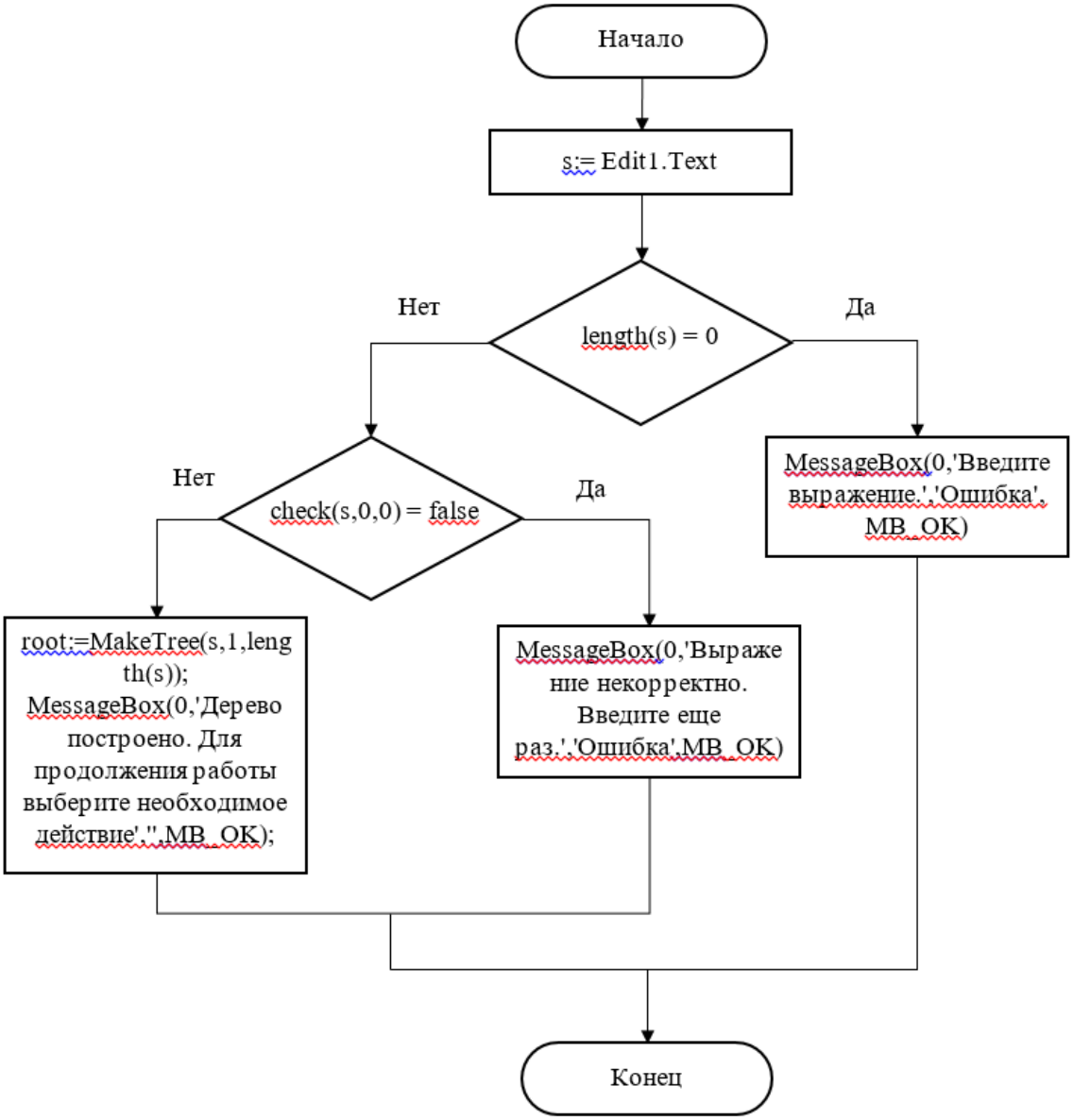
Имя компонента	Тип компонента	Пояснение
bt_build	TButton	кнопка с надписью «Построить»
bt_Clear	TButton	кнопка с надписью «Очистить»
bt_draw_tree	TButton	кнопка с надписью «Вывод дерева на экран»
bt_inorder	TButton	кнопка с надписью «Симметричный обход и вывод»
bt_preorder	TButton	кнопка с надписью «Прямой обход и вывод»
bt_postoreder	TButton	кнопка с надписью «Обратный обход и вывод»
bt_in_file	TButton	кнопка с надписью «Запись результатов обходов в файл»
bt_Close	TButton	кнопка с надписью «Выход»
Edit1	TEdit	поле для ввода арифметической формулы
edit_inorder	TEdit	поле вывода результата симметричного обхода
edit_preorder	TEdit	поле вывода результата прямого обхода
edit_postorder	TEdit	поле вывода результата обратного обхода
lb_vvod	TLabel	надпись «Для начала работы введите формулу»
lb_result	TLabel	надпись
lb_preorder	TLabel	надпись «Прямой обход»
lb_postorder	TLabel	надпись «Обратный обход»
lb_inorder	TLabel	надпись «Симметричный обход»
Label1	TLabel	надпись «Допустимые операции: + - * / ^»
lb_vvod	TLabel	надпись «Для начала работы введите формулу:»

field	TImage	поле, на котором выводится дерево в виде графа
Bevel1, Bevel2, Bevel3	TBevel	разделительные полосы

Компонентам TButton соответствуют обработчики событий, которые приведены в таблице 2.3.

Таблица 2.3

Наименование	Назначение
TForm1.bt_ClearClick	1. очистка поля ввода 2. очистка полей с результатами обходов 3. удаление изображения дерева в виде графа 4. присвоение пустого значения указателю на корень дерева
TForm1.bt_buildClick	1. проверка правильности введенной формулы при помощи функции Checks 2. вызов функции MakeTree, которая строит бинарное дерево
TForm1.bt_draw_treeClick	вызов процедуры DrawTree, которая выводит дерево на компонент Field
TForm1.bt_inorderClick	вызов процедуры Inorder, которая реализует симметричный обход
TForm1.bt_preorderClick	вызов процедуры Preorder, которая реализует симметричный обход
TForm1.bt_postorederClick	вызов процедуры Postorder, которая реализует симметричный обход
TForm1.bt_in_fileClick	запись результатов обходов в файл



Ниже изображены блок-схемы алгоритмов обработчиков событий.

Рисунок 2.3 – Алгоритм обработчика TForm1.bt_ClearClick

Рисунок 2.4 – Алгоритм обработчика TForm1.bt_buildClick

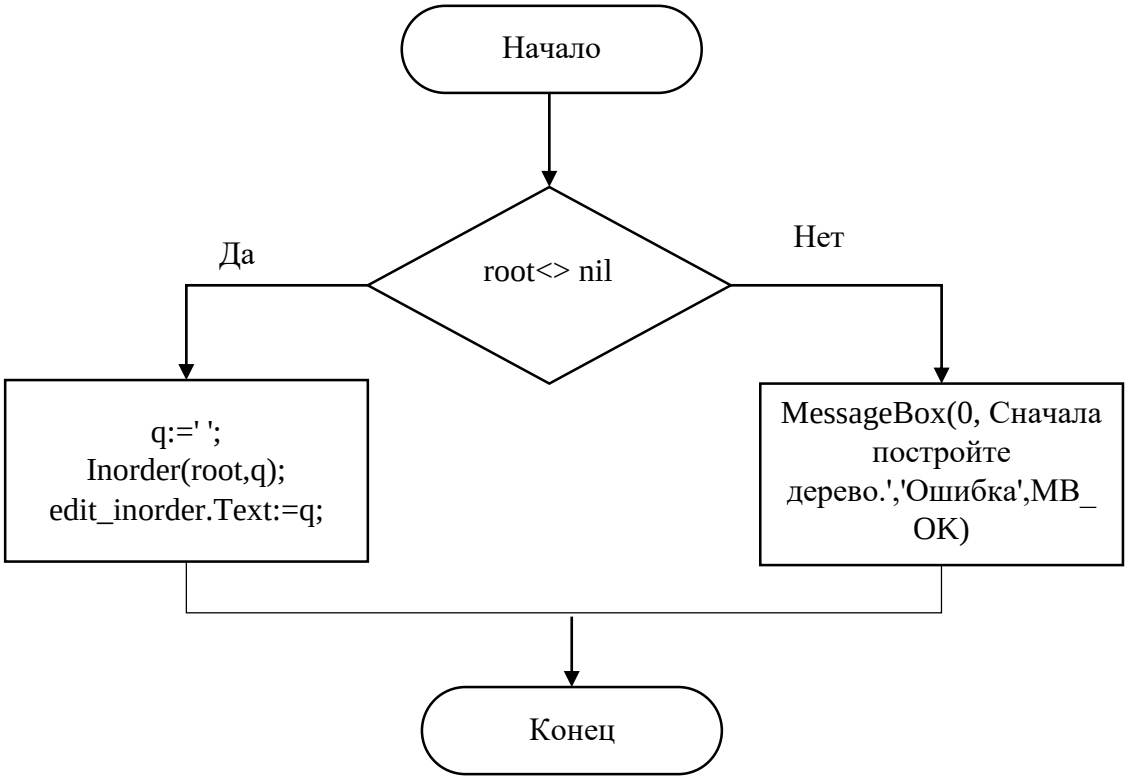


Рисунок 2.5 – Алгоритм обработчика TForm1.bt_inorderClick

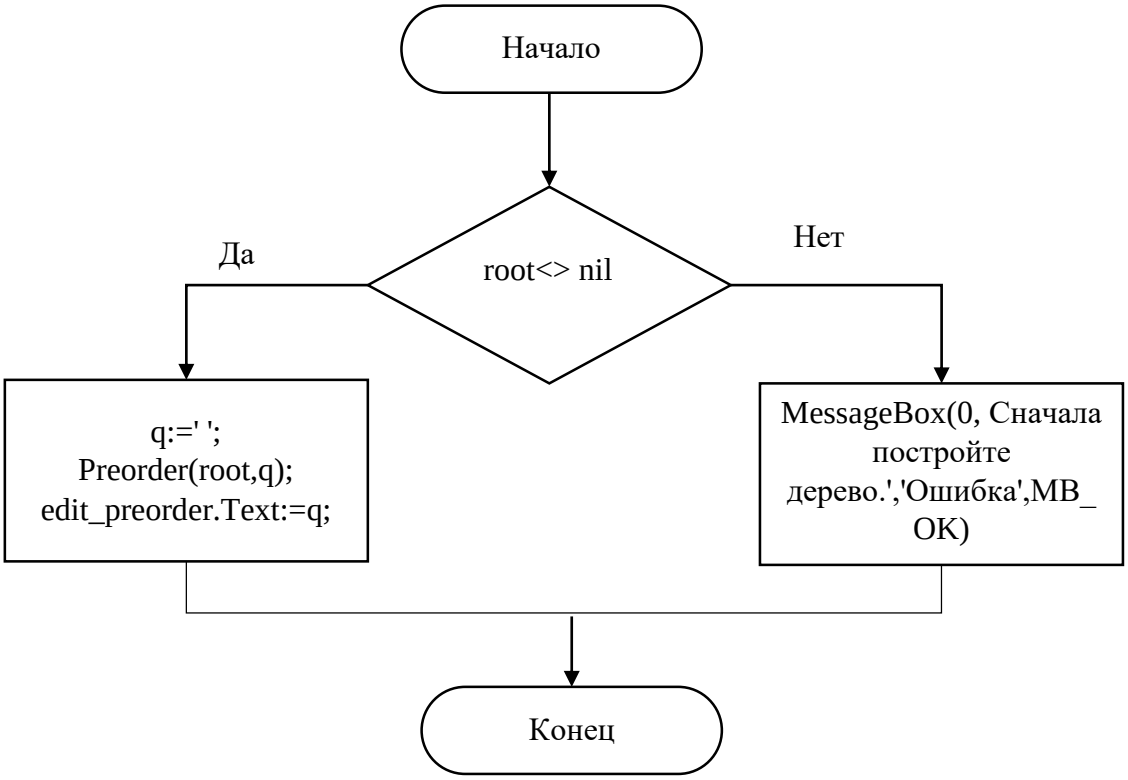


Рисунок 2.6 – Алгоритм обработчика TForm1.bt_preorderClick

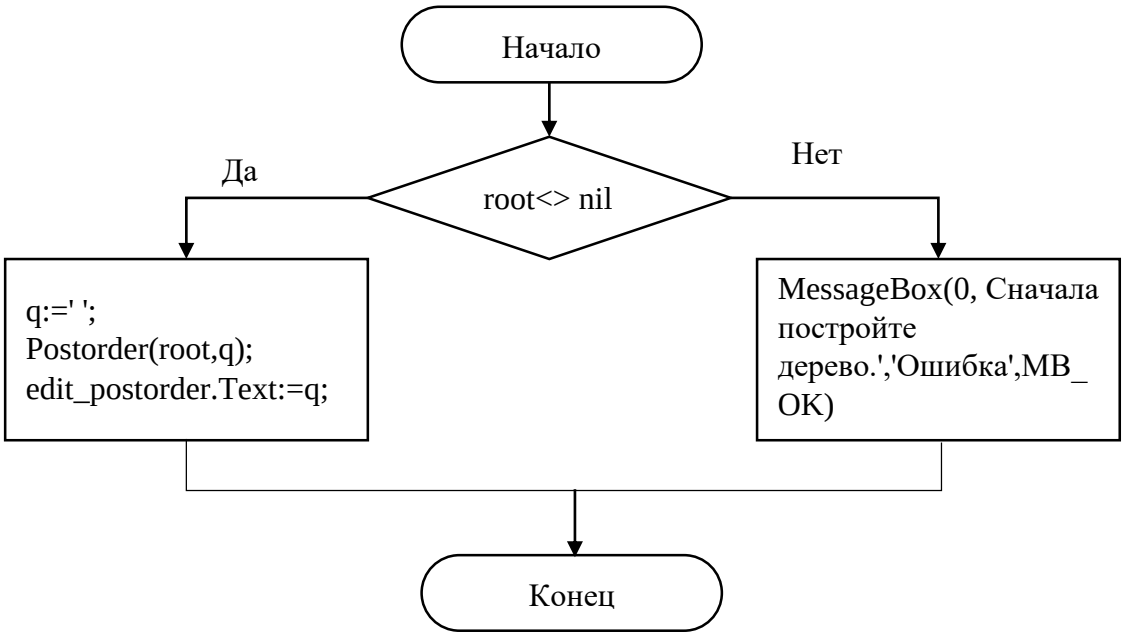


Рисунок 2.7 – Алгоритм обработчика TForm1.bt_postorederClick

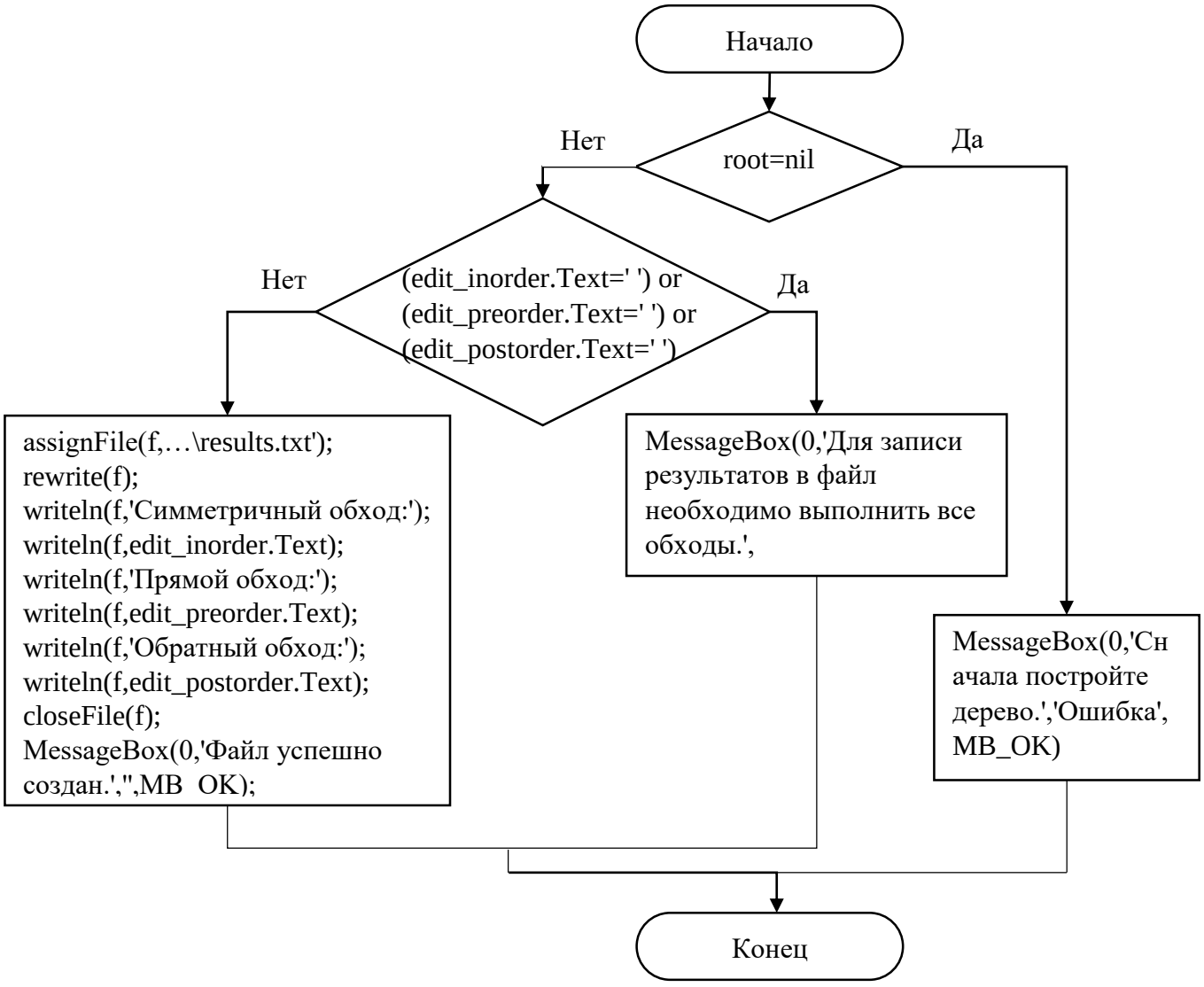


Рисунок 2.8 – Алгоритм обработчика TForm1.bt_in_fileClick

3 ТЕСТИРОВАНИЕ

Тестирование работоспособности программы основывалось на вводе некорректных исходных данных и непоследовательных нажатиях кнопок. Все попытки были отклонены программой. В таблице 3.1 перечислены проводимые тесты и их результат.

Таблица 3.1

Номер теста	Тест	Результат
1.	ввод некорректной формулы	сообщение о вводе некорректной формулы и предложение ввести формулу ещё раз
2.	нажатие на кнопку «Построить» до ввода формулы	сообщение о необходимости сначала ввести формулу
3.	нажатие на кнопку «Вывод дерева на экран» до ввода формулы	
4.	нажатие на кнопку «Симметричный обход и вывод» до ввода формулы	
5.	нажатие на кнопку «Прямой обход и вывод» до ввода формулы	
6.	нажатие на кнопку «Обратный обход и вывод» до ввода формулы	
7.	нажатие на кнопку «Симметричный обход и вывод» до построения дерева	сообщение о необходимости сначала построить дерево
8.	нажатие на кнопку «Прямой обход и вывод» до построения дерева	
9.	нажатие на кнопку «Обратный обход и вывод» построения дерева	
10.	запись результатов в файл до построения дерева	сообщение о необходимости сначала построить дерево
11.	запись результатов в файл до совершения всех обходов	сообщение о необходимости осуществлении всех трех обходов

4 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Для запуска программы выберите файл “Project1.exe”. После запуска программы в верхнем поле введите математическую формулу (рисунок 3.1).

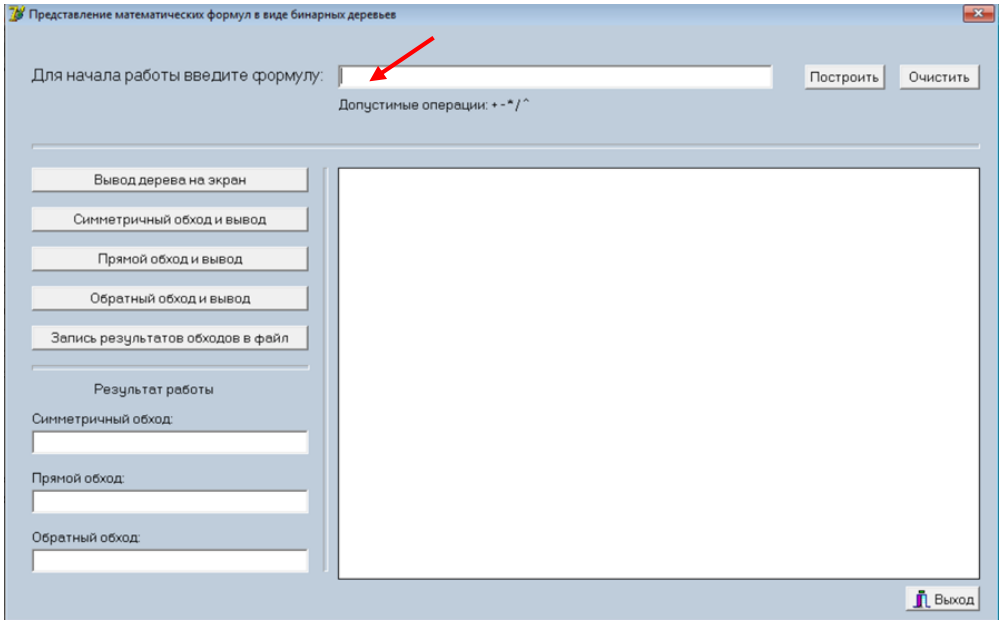


Рисунок 3.1

Для построения дерева нажмите кнопку «Построить» (рисунок 3.2).

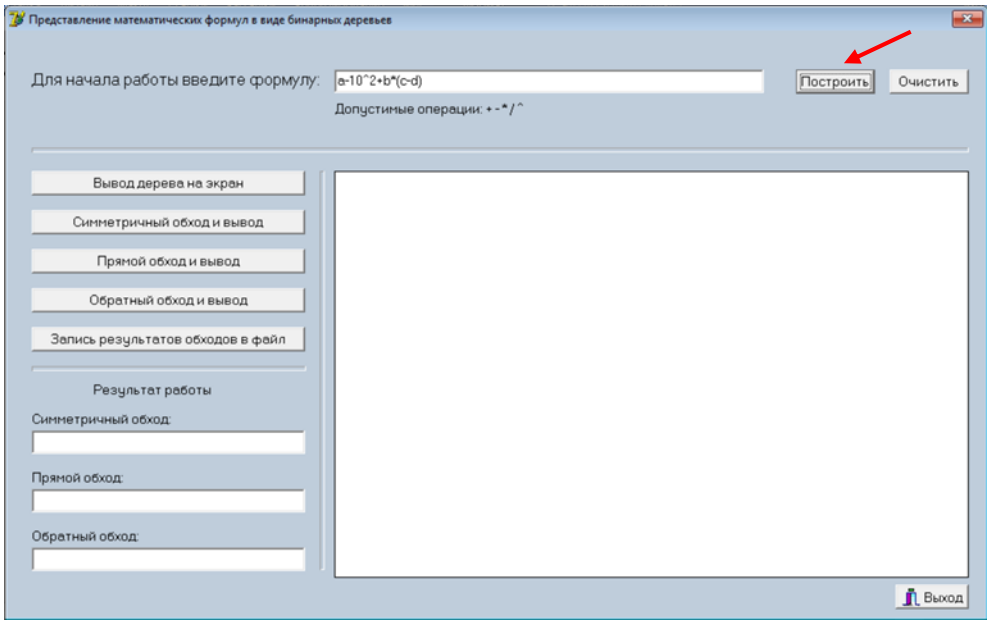


Рисунок 3.2

Программа проверяет корректность введенной формулы. Если ошибок в формуле нет, то программа сообщает, что бинарное дерево построено (рисунок 3.3).

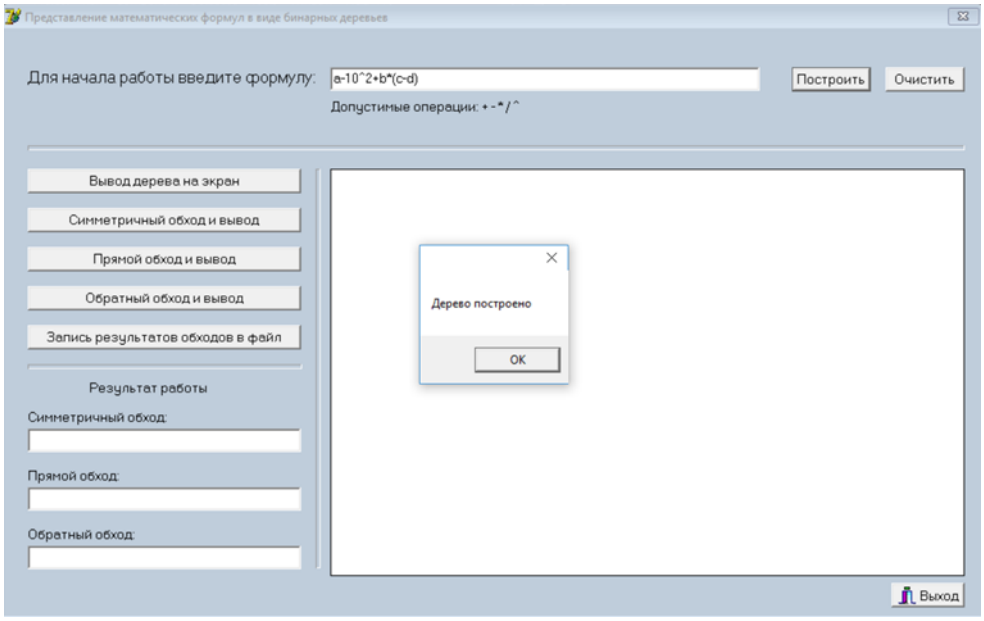


Рисунок 3.3

При вводе некорректной формулы программа сообщает об ошибке (рисунок 3.4).

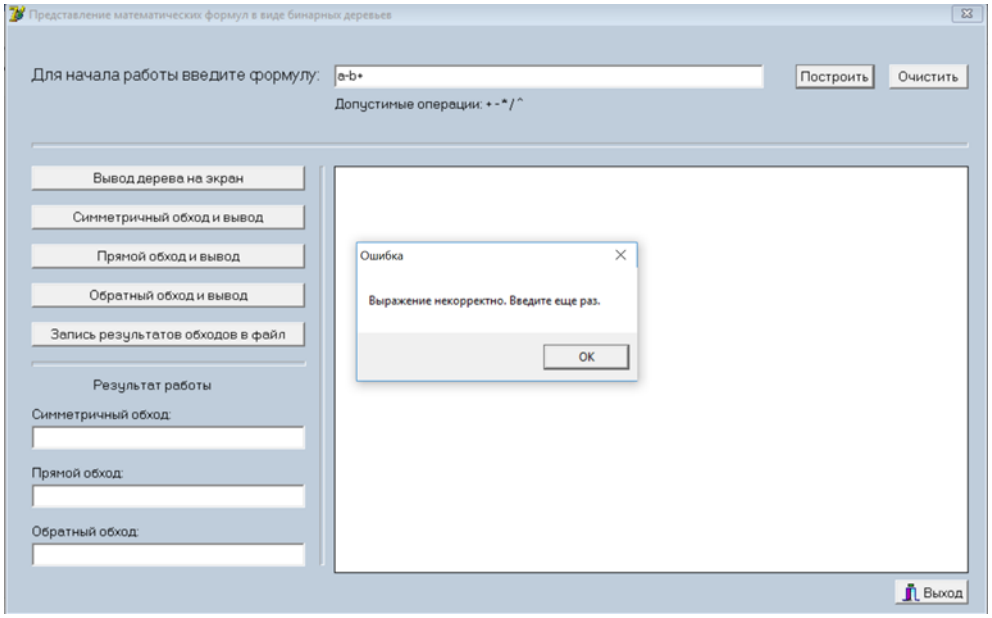


Рисунок 3.4

После ввода некорректной формулы нажмите на кнопку «Очистить» (рисунок 3.5).

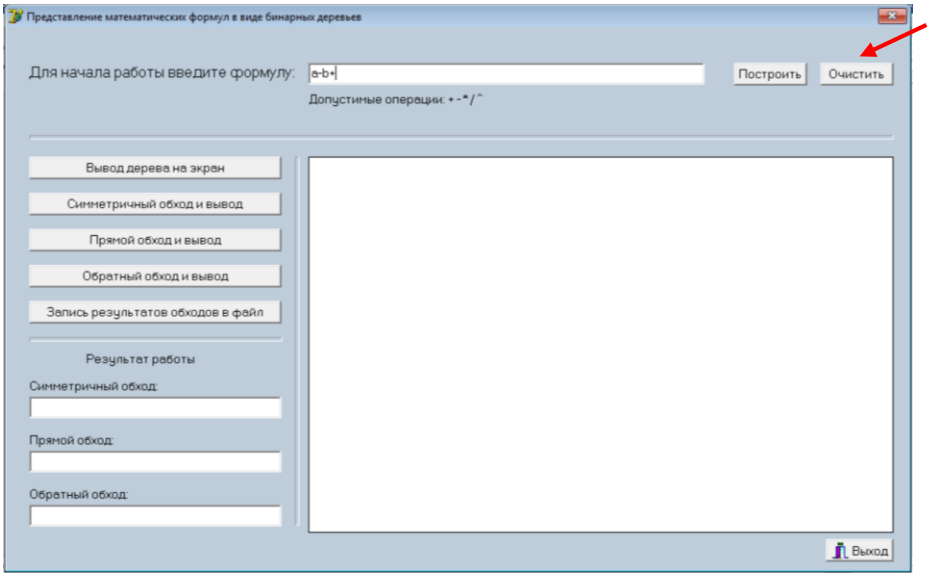


Рисунок 3.5

Введите корректное выражение и нажмите на кнопку Построить. Выберите в меню действие «Вывод дерева на экран» (рисунок 3.6).

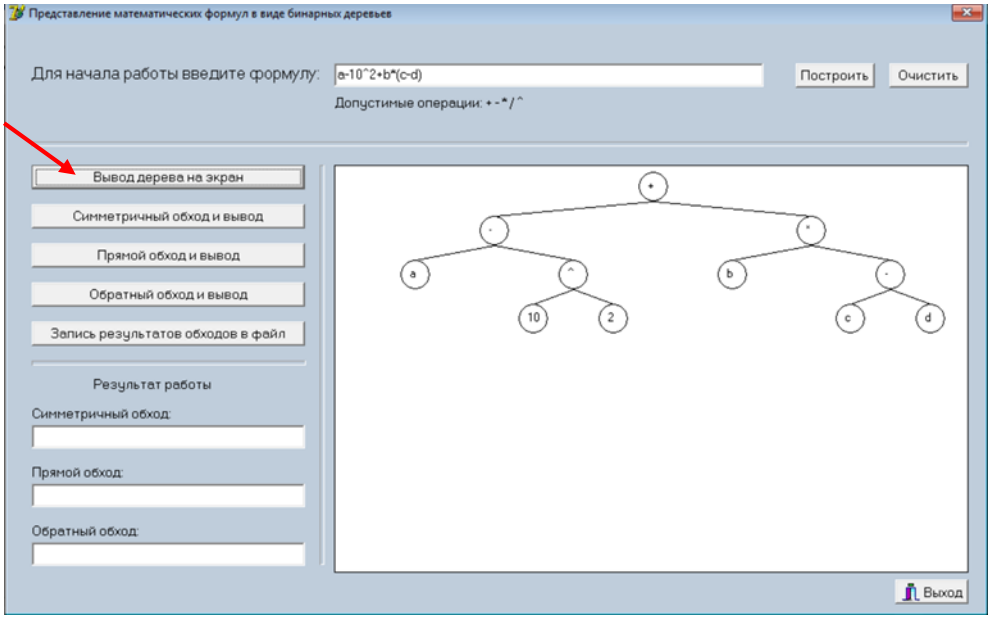


Рисунок 3.6

Выберите в меню действия для осуществления обходов (рисунок 3.7).

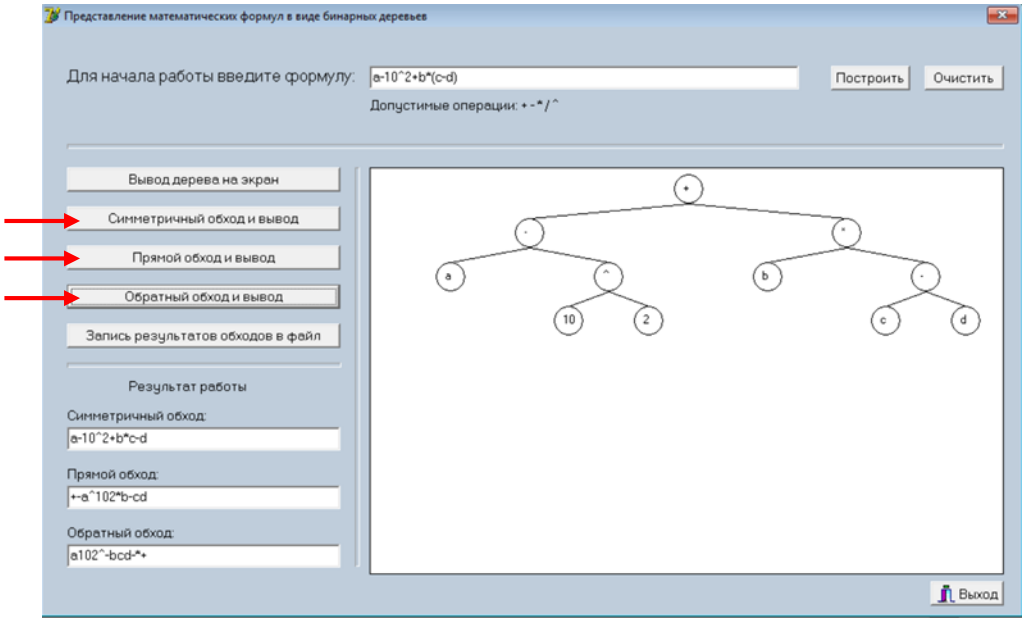


Рисунок 3.7

Выберите действие «Запись результатов обходов в файл» (рисунок 3.8).

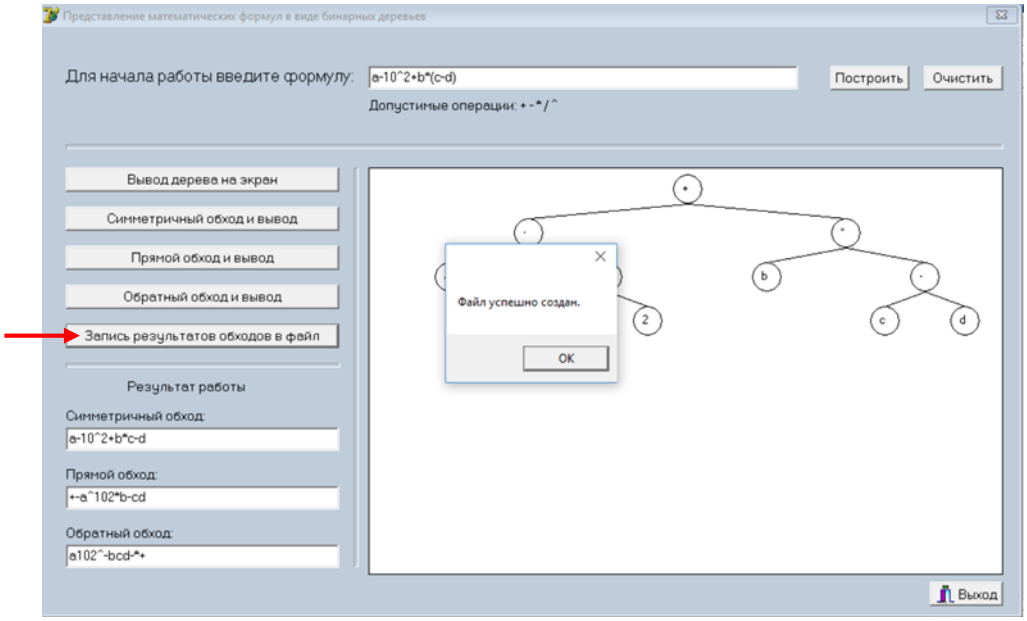


Рисунок 3.8

Для ввода новой формулы нажмите кнопку «Очистить». Для выхода из программы нажмите кнопку «Выход».

ЗАКЛЮЧЕНИЕ

В результате работы была разработана программа, которая создает бинарное дерево по арифметической формуле и осуществляет симметричный, прямой и обратный обходы построенного дерева. Программа имеет ясный и понятный интерфейс, обеспечивающий удобство в работе.

В ходе курсового проектирования были закреплены теоретические и практические навыки по дисциплине «Основы алгоритмизации и программирования» и усвоены приемы построения и способы обхода бинарного дерева.

СПИСОК ЛИТЕРАТУРЫ

[1] Серебряная, Л. В. Структуры и алгоритмы обработки данных : учеб.-метод. пособие / Л. В. Серебряная, И. М. Марина. – Минск : БГУИР, 2013. – 51с.

[2] Иванова, Г. С. Основы программирования : Учебник для вузов / Г. С. Иванова. – М. : Изд-во МГТУ им. Н. Э. Баумана, 2002. – 416 с.

[3] Бесплатная интернет библиотека [Электронный ресурс]. – Режим доступа: http://book.lib-i.ru/.

[4] Глухова, Л. А. Электронный учебно-методический комплекс «Основы алгоритмизации и программирования»: Учеб. пособие. Часть 2 / Л. А. Глухова. – Мн.: БГУИР, 2006.

ПРИЛОЖЕНИЕ А

Листинг программы

```
unit Menu;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,
  Dialogs, Buttons, StdCtrls, ExtCtrls, Check, Tree;

type
  TForm1 = class(TForm)
    lb_vvod: TLabel;
    Edit1: TEdit;
    Bevel1: TBevel;
    bt_Clear: TButton;
    bt_Close: TBitBtn;
    bt_build: TButton;
    lb_check: TLabel;
    bt_draw_tree: TButton;
    bt_inorder: TButton;
    bt_preorder: TButton;
    bt_in_file: TButton;
    Bevel2: TBevel;
    Bevel3: TBevel;
    edit_inorder: TEdit;
    edit_preorder: TEdit;
    bt_postoreder: TButton;
    edit_postorder: TEdit;
    lb_inorder: TLabel;
    lb_preorder: TLabel;
    lb_postorder: TLabel;
    lb_result: TLabel;
    Shape1: TShape;
    field: TImage;
    Label1: TLabel;
    procedure bt_ClearClick(Sender: TObject);
    procedure bt_buildClick(Sender: TObject);
    procedure bt_draw_treeClick(Sender: TObject);
    procedure bt_inorderClick(Sender: TObject);
    procedure bt_preorderClick(Sender: TObject);
```

```
    procedure bt_postorederClick(Sender: TObject);
    procedure bt_in_fileClick(Sender: TObject);
   end;

var
  Form1: TForm1;
  s,q: string;
  root: PNode;
  f: textFile;

implementation

{$R *.dfm}

procedure TForm1.bt_ClearClick(Sender: TObject);
begin
  Edit1.Text:='';
  s:='';
  root:=nil;
  Form1.field.Picture:=nil;
  edit_inorder.Text:='';
  edit_preorder.Text:='';
  edit_postorder.Text:='';
end;

procedure TForm1.bt_buildClick(Sender: TObject);
begin
  s:= Edit1.Text;
  if (length(s)=0) then
     MessageBox(0,'Введите выражение.','Ошибка',MB_OK)
  else
  if checks(s)=false then
     MessageBox(0,'Выражение некорректно. Введите еще
раз.','Ошибка',MB_OK)
  else
  begin
    new(root);
    MakeTree(root,s,1,length(s));
    MessageBox(0,'Дерево построено','',MB_OK);
  end
end;

procedure TForm1.bt_draw_treeClick(Sender: TObject);
begin
```

```
  if root<>nil then
    DrawTree(root,0,325,650,20)
  else
    MessageBox(0,'Сначала постройте дерево','Ошибка',MB_OK);
end;

procedure TForm1.bt_inorderClick(Sender: TObject);
begin
  if root<> nil then
  begin
    q:='';
    Inorder(root,q);
    edit_inorder.Text:=q;
  end
  else
    MessageBox(0,'Сначала постройте дерево','Ошибка',MB_OK);
end;

procedure TForm1.bt_preorderClick(Sender: TObject);
begin
  if root<> nil then
  begin
    q:='';
    Preorder(root,q);
    edit_preorder.Text:=q;
  end
  else
    MessageBox(0,'Сначала постройте дерево','Ошибка',MB_OK);
end;

procedure TForm1.bt_postorederClick(Sender: TObject);
begin
  if root<> nil then
  begin
    q:='';
    postorder(root,q);
    edit_postorder.Text:=q;
  end
  else
    MessageBox(0,'Сначала постройте дерево','Ошибка',MB_OK);
end;

procedure TForm1.bt_in_fileClick(Sender: TObject);
begin
```

```
          if root=nil then
            MessageBox(0,'Сначала постройте дерево.','Ошибка',MB_OK)
          else
          if (edit_inorder.Text='') or (edit_preorder.Text='') or
             (edit_postorder.Text='') then
             MessageBox(0,'Для записи результатов в файл необходимо
выполнить все обходы.',
             'Ошибка',MB_OK)
          else
          begin
            assignFile(f,'C:\Users\User\Desktop\Del\results.txt');
            rewrite(f);
            writeln(f,'Симметричный обход:');
            writeln(f,edit_inorder.Text);
            writeln(f,'Прямой обход:');
            writeln(f,edit_preorder.Text);
            writeln(f,'Обратный обход:');
            writeln(f,edit_postorder.Text);
            closeFile(f);
            MessageBox(0,'Файл успешно создан в папке с файлом
программы.','',MB_OK);
          end
        end;

        end.
```

unit Check;

```
        Interface

        function Checks(var s:string):boolean;

        Implementation

        const
          digits: set of char = ['0'..'9'];
          letters: set of char = ['A'..'Z', 'a'..'z' , 'А'..'Я','а'..'я'];
          op: set of char = ['+', '-', '*', '/','^'];

        procedure probel(var s:string);
        begin
          while pos(' ',s)>0 do
          delete(s,pos(' ',s),1);
        end;
```

```
procedure Number (s:string; var i:integer);
begin
  while (i < length(s)) and (s[i + 1] in digits) do
    i := i + 1;
end;

function Checks(var s:string):boolean;
var state:0..3;
    i,k:integer;
begin
  probel(s);
  i:=0;
  k:=0;
  state:=0;
  while state <> 3 do
  case state of
    0:
    if i < length(s) then
    begin
      i := i + 1;
      if s[i]='(' then k:=k+1
      else
        if not (s[i] in (letters+digits))then
          begin
            result:=false;
            exit;
          end
        else
          begin
            if s[i] in letters then
              if (i<>length(s)) and (s[i+1]  in letters) then
                begin
                  result:=false;
                  exit;
                end;
            if s[i] in digits then Number(s,i);
            state:=1;
          end
    end
    else
      state:=2;
    1:
    if i<length(s) then
```

```
    begin
      i:=i + 1;
      if s[i]=')' then
      begin
        k:=k - 1;
        if k < 0 then begin result:=false; exit; end;
      end
      else
        if s[i] in op then state := 0
      else begin result:=false; exit; end;
    end
    else state := 2;
    2:
    if (s[i] in (letters + digits+ [')'])) and (k = 0) then
    begin
      result:=true;
      state:= 3
    end
    else
    begin
      result:=false;
      exit;
    end;
  end;
end;

end.
```

unit Tree;

```
Interface
type
  PNode=^Node;
  node=record
      data:string[5];
      left,right:PNode;
    end;

procedure MakeTree(r:PNode; expr:string; first,last: integer);
procedure DrawTree(p:PNode;l,c,r,y:integer);
procedure Inorder(p:PNode; var q:string);
procedure Preorder(p:PNode; var q:string);
procedure Postorder(p:PNode; var q:string);
```

Implementation

```
uses Menu;

var
  minPrt:integer;

function Priority(c:char): integer;
begin
  case (c) of
  '+','-': Priority:=1;
  '*','/': Priority:=2;
  '^': Priority:=3;
  else
    Priority:=10;
  end;
end;

function LastOperation(expr:string; first,last:integer):integer;
var prt,i,ch:integer;
begin
  minPrt:=100;
  ch:=0;
  for i:=first to last do
  begin
    if expr[i] ='(' then
    begin
      ch:=ch+1;
      continue;
    end;
    if expr[i] =')' then
    begin
      ch:=ch-1;
      continue;
    end;
    if ch>0 then
      continue
    if ch=0 then
    begin
        prt:=Priority(expr[i]);
        if prt<=minPrt then
        begin
          minPrt:=prt;
          result:=i;
```

```
    end;
  end
end;

procedure MakeTree(r: Pnode; expr:string; first,last: integer);
var k:integer;
begin
  if (first=last) then
  begin
    r^.data:=expr[first];
    r^.left:=nil;
    r^.right:=nil;
    exit;
  end;
  if (first<>last) then
  begin
    k:=LastOperation(expr,first, last);
    if minPrt=100 then
    begin
      first:=first+1;
      last:=last-1;
      k:=LastOperation(expr,first, last);
    end;

    if minPrt<>100 then
    begin
      if minPrt<>10 then
      begin
        r^.data:=expr[k];
        new (r^.left);
        MakeTree(r^.left,expr, first, k-1);
        new (r^.right);
        MakeTree(r^.right, expr, k+1, last);
      end
      else
      begin
        r^.data:=copy(expr,first,last-first+1);
        r^.left:=nil;
        r^.right:=nil;
      end;
    end;
  end;
end;
```

```
procedure DrawTree(p:PNode;l,c,r,y:integer);
const rad=15;
      function min(a, b: Integer): Integer;
 begin
       min:= a;
            if b < a then min:=b
 end;

 function Center(a,b: Integer): Integer;
 begin
  Center:= min(a,b)+abs(a-b) div 2;
 end;

begin
 if p<> nil then
 begin
  form1.field.Canvas.Ellipse(c-rad,y-rad, c+rad,y+rad);
  form1.field.Canvas.TextOut(c-6,y-8,p^.data);
 end;
 if p^.left <>nil then
 begin
  DrawTree(p^.left,l,Center(c,l),c,y+45);
  form1.field.Canvas.MoveTo(c,y+rad);
  form1.field.Canvas.LineTo(Center(c,l),y+30);
 end;
 if p^.right <>nil then
 begin
  DrawTree(p^.right,c,Center(c,r),r,y+45);
  form1.field.Canvas.MoveTo(c,y+rad);
  form1.field.Canvas.LineTo(Center(c,r),y+30);
 end;
end;

procedure Inorder(p:PNode;var q:string);
begin
 if p<>nil then
 begin
  Inorder(p^.left,q);
  q:=q+p^.data;
  Inorder(p^.right,q);
 end;
end;

procedure Preorder(p:PNode; var q:string);
```

```
begin
  if p<>nil then
  begin
    q:=q+p^.data;
    Preorder(p^.left,q);
    Preorder(P^.right,q);
  end;
end;

procedure Postorder(p:PNode; var q:string);
begin
  if p<>nil then
  begin
    Postorder(p^.left,q);
    Postorder(p^.right,q);
    q:=q+p^.data;
  end;
end;

end.
```