Лабораторная работа №4
Работа с деревьями

Цель работы: изучить библиотечные классы языка C#, предназначенные для работы с деревьями и научится их использовать в программе.

Справочная информация

Дерево, как модель иерархических отношений

Мы определили, дерево, как связный граф без циклов. Все вершины дерева равноправны. Но для моделирования иерархических отношений, которые часто встречаются как в практической жизни, так и в математике, и в программировании, используется дополненное определение дерева. В этом случае вершины называют, как правило, узлами, а ребра задают отношения, которые называют родительскими. Один узел выделен как корень.
Если рассматривать дерево в старом определении, то для того что бы получить дерево в новом определении, надо одну вершину выделить, назвав ее корнем, а смежные ей вершины назвать сыновьями. Для этих узлов корень будет родителем. Корень не имеет родителя. Т.е. вершины, получив название узлы, упорядочиваются некоторым образом. Если дерево представить веревочной моделью, в которой вершины узелки, а ребра нитки, то в этом случае, взяв за узелок, названный корнем и встряхнув за нее все дерево, мы получим дерево в новом определении. Итак, корень может иметь сыновей. Каждый из его сыновей может иметь своих сыновей, т.е. быть родителем для этих узлов и т.д. Каждый узел, кроме корня, имеет одного родителя. Узлы, не имеющие сыновей, называются листьями.
Итак, новое определение дерева отличается от старого, тем, что узлы упорядочиваются. Какое определение дерева используется зависит от контекста.
Сыновья узла обычно упорядочиваются слева направо. Если порядок игнорируется, то дерево называется неупорядоченным, в противном случае дерево называется упорядоченным.
Примеры деревьев.

1. Структура вложенности каталогов и файлов в современных операционных системах.
2. Дерево синтаксического разбора.
3. Иерархическая структура вложенных элементов данных.
4. Иерархия классов.

Библиотечные классы, предназначенные для работы с деревьями

Для работы с деревьями в библиотеке языка C# введен класс *TreeNode*. Объекты, которого имеют двойственную природу, их можно рассматривать

или как поддерево или как узел, который совпадает с корнем этого поддерева. Свойство *Text* определяет имя корня. Свойство *Nodes* – является списком всех сыновей корня этого поддерева. Каждый сын – это в свою очередь объект типа *TreeNode* Удаление узла приводит к удалению всех его потомков, т.е. всего поддерева, корнем которого является удаляемый узел. Реализация такой структуры возможна, поскольку в списке *Nodes* хранятся ссылки на узлы являющиеся сыновьями.
Рассмотрим пример создания дерева.
TreeNode t=new TreeNode("Корень");
Данная команда создает узел – объект типа *TreeNode* , который будет корнем нашего дерева.
TreeNode t1=new TreeNode("C1");
t.Nodes.Add(t1);
Данная команда добавляет узел *t1* к дереву *t.* Теперь к узлу *t1* можно обращаться и по ссылке *t1* и по номеру *0* в списке *t.Nodes*, кроме того, к этому узлу можно обратиться через свойство *t.FirstNode* поскольку данный узел стоит в списке первым. Свойство *t.LastNode* возвращает ссылку на последний элемент в списке *t.Nodes*. Свойство *t.Parent* возвращает ссылку на родительский узел, если *t* корень, она будет равна *null*.
Свойство *t1.Index* возвращает номер узла *t1*, в списке *Nodes.*

Для просмотра дерева и работы с ним создан компонент *TreeView*, который является контейнером для объектов *TreeNode* Котпонент *TreeView* имеет свойство *Nodes*, которое является списком объектов *TreeNode*. Котпонент *TreeView* визуальный - это прямоугольник, в котором нарисовано дерево, похожее на дерево папок и файлов. Каждый узел может быть свернутым и развернутым, может иметь имя и значок. Свойства для чтения *IsExpanded, IsVisible; IsSelected;* имеют булев тип и сообщают, является ли узел развернутым, видимым и выделенным, соответственно.
Чаще всего работа происходит с выделенным узлом, поэтому класс *TreeView* имеет свойство *SelectedNode*, которое указывает на выделенный узел, это свойство предназначено и для чтения и для записи.
Команда *treeView1.SelectedNode.Remove();* удалит выделенный узел, а *treeView1.SelectedNode.Text="op";* переименует выделенный узел.
У класса *TreeView* есть события , связанные с действиями над деревьями : *AfterSelect, BeforeSelect, AfterCollapse BeforeCollapse, AfterExpand* BeforeExpand, Первые два события происходят при выделении узла. Событие *BeforeSelect* до выделения, *AfterSelect* - после выделения. Следующие два события связаны со сворачивание узла, а последние два с разворачиванием узла.
Все обработчики этих событий имеют формальный параметр e, который, через свойство *Node* передает ссылку на узел, с которым происходит соответствующее событие.
Например,

```
private void treeView1_BeforeCollapse(object sender,
System.Windows.Forms.TreeViewCancelEventArgs e)
        {
        label2.Text=e.Node.Text;
        }
```

Данный фрагмент программы в момент сворачивания узла будет выводить его имя на экран.

Задание

Общее задание. На форму помещен пустой компонент treeView1 и 3 кнопки. При нажатии на первую кнопку в компоненте treeView1 появляется дерево, определенное вариантом.

Задание состоит из 4 частей. В первой части указывается обработчик, какого события должен быть создан. Вторая часть задания определяет работу второй кнопки, третья часть задания – работу третьей кнопки. Четвертая часть задания определяет дерево, которое выводится при нажатии первой кнопки.

Первая часть задания. Работа с событиями свертывания, развертывания и выделения узлов.

1. В момент свертывания узла, его имя выводится на форму.
2. В момент развертывания узла, его имя выводится на форму.
3. В момент выделения узла, его имя выводится на форму.

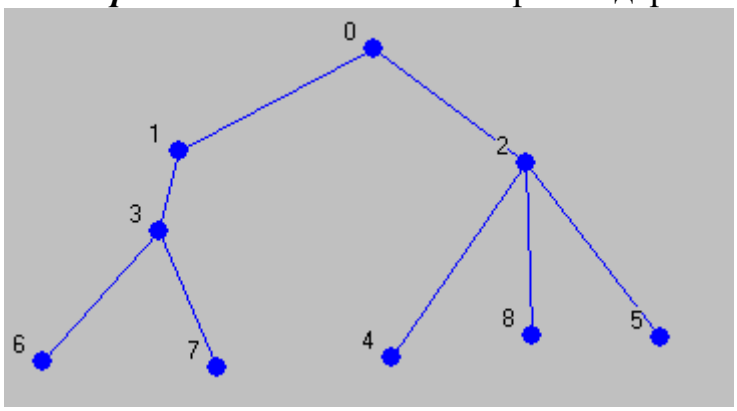
Вторая часть задания. Это задание определяет работу первой кнопки

1. выводит на форму имя выделенного узла.
2. удаляет выделенный узел.
3. добавляет сына к выделенному узлу. Имя сына задается пользователем.

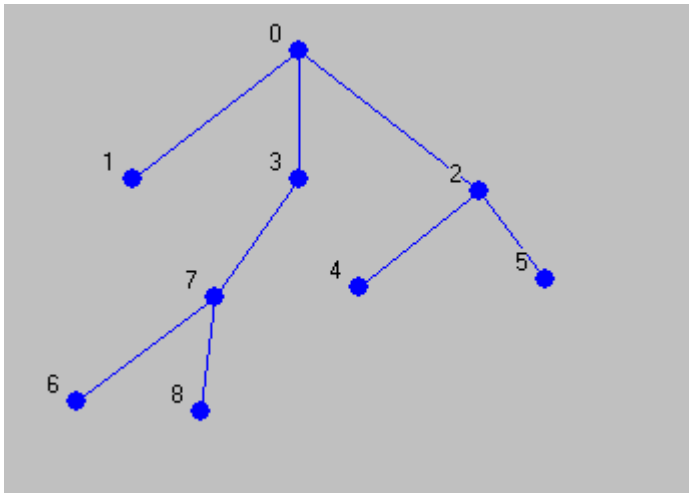
Третья часть задания. Кнопка, которая выполняет, (если возможно, если не возможно, то выводит сообщение об этом) следующее действие:

1. На форму выводит список имен сыновей выделенного узла.
2. На форму выводит имя родителя выделенного узла.
3. На форму выводит имя первого сына выделенного узла.
4. На форму выводит имя последнего сына выделенного узла.
5. На форму выводит имя левого брата выделенного узла.
6. На форму выводит имя правого брата выделенного узла.
7. Сообщает, является ли выделенный узел листом.
8. Выясняет, имеет ли выделенный узел братьев.
9. Удаляет первого сына выделенного узла.
10. Удаляет последнего сына выделенного узла.
11. Удаляет всех сыновей выделенного узла, сам узел оставляет на месте.
12. Удаляет левого брата выделенного узла.
13. Удаляет правого брата выделенного узла.

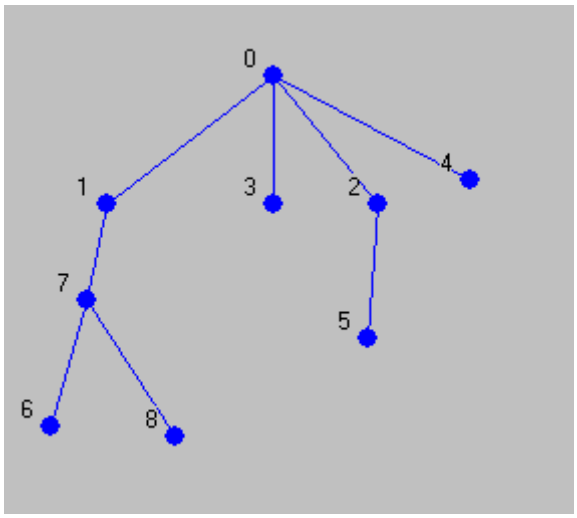
Четвертая часть задания. Вариант дерева.



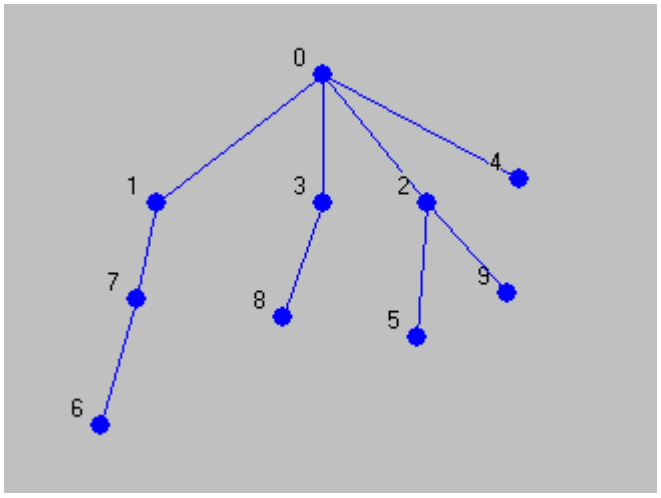
Вариант 1



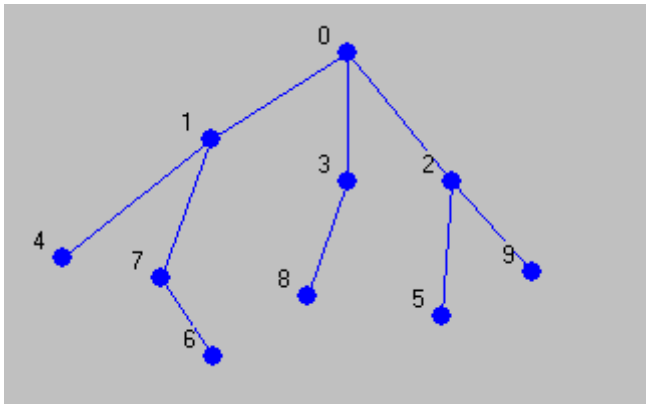
Вариант 2



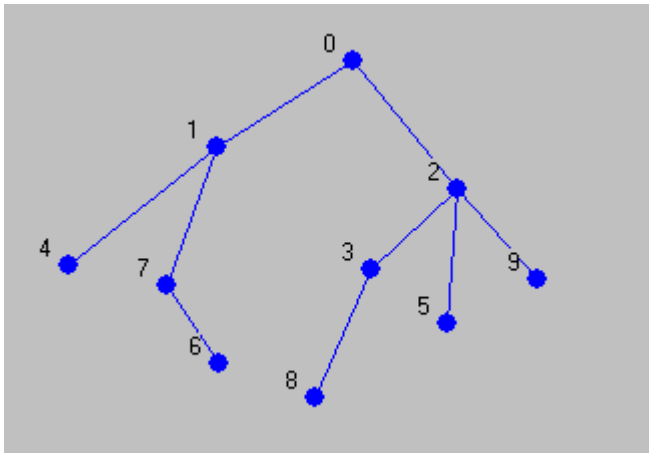
Вариант 3



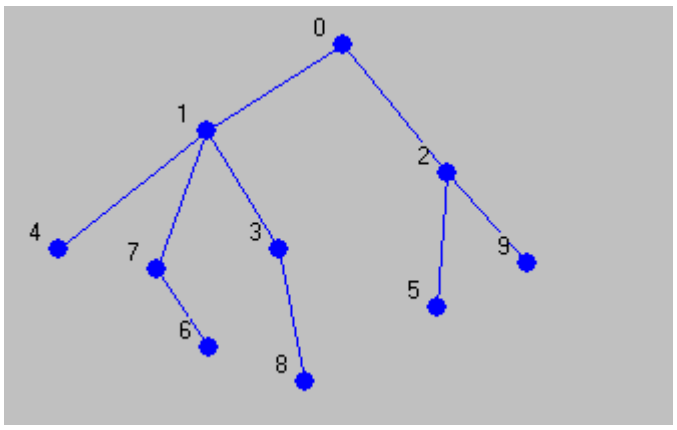
Вариант 4



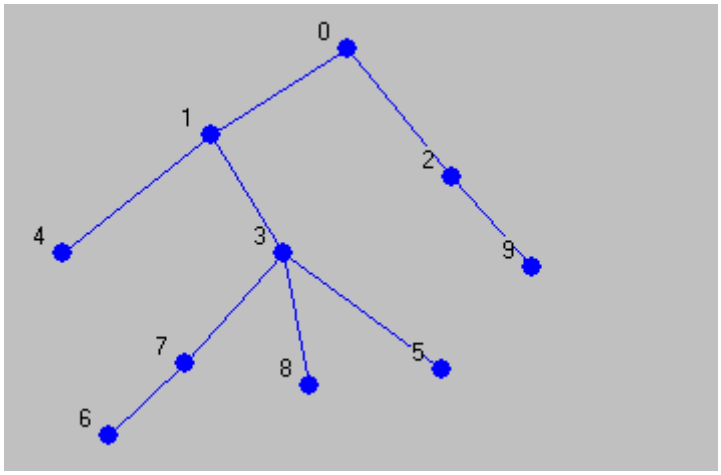
Вариант 5



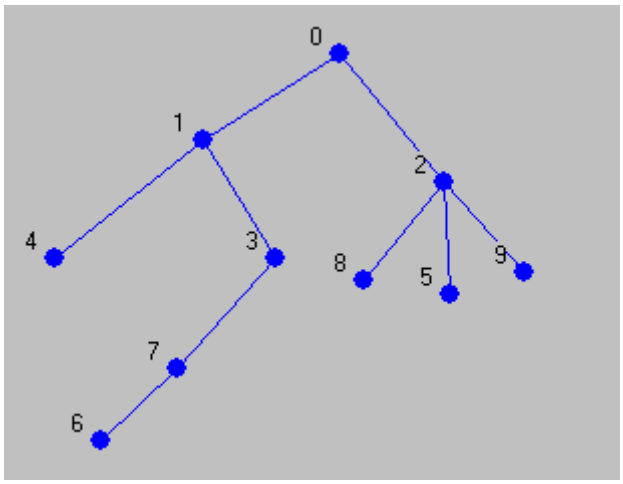
Вариант 6



Вариант 7



Вариант 8



Вариант 9

Индивидуальные задания.

Вариант полного задания состоит из вариантов четырех частей.

Вариант 1. 1.1, 2.1, 3.1, 4.1.
Вариант 2. 1.2, 2.1, 3.2, 4.2.
Вариант 3. 1.3, 2.1, 3.3, 4.3.
Вариант 4. 1.1, 2.2, 3.4, 4.4.
Вариант 5. 1.2, 2.2, 3.5, 4.5.
Вариант 6. 1.3, 2.2, 3.6, 4.6.
Вариант 7. 1.1, 2.3, 3.7, 4.7.
Вариант 8. 1.2., 2.3, 3.8, 4.8.
Вариант 9. 1.3, 2.3, 3.9, 4.9.
Вариант 10. 1.1, 2.3, 3.10, 4.1.
Вариант 11. 1.1, 2.2, 3.11, 4.2.
Вариант 12. 1.1, 2.2, 3.12, 4.3.
Вариант 13. 1.1, 2.1, 3.13, 4.4.
Вариант 14. 1.2, 2.1, 3.1, 4.5.
Вариант 15. 1.2, 2.2, 3.2, 4.6.
Вариант 16. 1.2, 2.3, 3.3, 4.7.
Вариант 17. 1.3, 2.1, 3.4, 4.8.
Вариант 18. 1.3, 2.3, 3.5, 4.9.

Контрольные вопросы

1. Как определяется дерево?
2. Какой узел называется корнем?
3. Какой узел называется листом?
4. У какого узла нет родителя?
5. У какого узла нет сыновей?
6. Как можно рассматривать объекты класса `TreeNode`?
7. Какое свойство класса `TreeNode` позволяет перейти от родителя к сыновьям?
8. Как проверить, имеет ли узел сыновей?
9. Как проверить имеет ли узел родителя?
10. Как обратится к первому сыну узла t?
11. Какой компонент предназначен для просмотра дерева?
12. Как поместить дерево в компонент *TreeView?*
13. Какие события компонента *TreeView* позволяют работать с включенным в него деревом?
14. Какое свойство компонента позволяет *TreeView* обратиться к выделенному узлу?

План отчета

1. Титульный лист.
2. Цель работы.
3. Задание. При формулировке задания, рисунок дерева обязателен.

4. Теоретический материал, используемый при выполнении задания.
5. Распечатка программы.

Этапы выполнения лабораторной работы

1. Изучить теоретический материал.
2. Ответить на контрольные вопросы.
3. Написать программу, решающую поставленную задачу.
4. Отладить программу
5. Написать отчет.

Лабораторная работа №5
Обходы дерева

Цель: Изучить алгоритмы самых распространенных обходов дерева и на основе одного из алгоритмов написать программу.

Справочная информация

Обходы дерева

Обход дерева – это список, в который заносятся узлы по мере их прохождения.

Наиболее распространены три следующих обхода :

1) прямой;
2) обратный;
3) симметричный.

Дадим определения этих обходов.

Если дерево T есть нулевое дерево, то в список обхода заносится пустая запись.

Если дерево состоит из одного узла, то в список записывается этот узел.

Рассмотрим более сложное дерево T, n-корень дерева T, T_1,…, $T_к$, - поддеревья дерева T, корнями которых, являются сыновья узла n.

1.При прохождении в прямом порядке узлов дерева T сначала посещается корень n, затем узлы поддерева T_1,…, T_k, так же в прямом порядке.

2. При прохождении в обратном порядке узлов дерева T, сначала посещается в обратном порядке все узлы поддерева T_1, затем T_2, затем T_3, …, $T_к$, так же в обратном порядке и затем корень n.
3. При прохождении в симметричном порядке узлов дерева T, сначала посещаются в симметричном порядке все узлы поддерева T_1, далее корень n, затем последовательно в симметричном порядке все узлы поддеревьев T_2,…$T_к$.

Помеченные деревья

Часто каждому узлу дерева ставят в соответствие метку-значение. Дерево, у которого узлами в соответствие поставлены метки называются помеченными деревом.

Метка узла – это не имя узла, а значение, которое храниться в узле. Рассмотрим дерево синтаксического разбора арифметического выражения: (a+b)*(a+c). Оно представлено на Рисунке 3

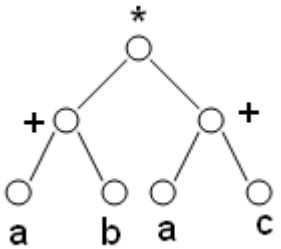


Рисунок 1

Часто при обходе дерева составляется список не имён узлов, а их меток.
Прямой обход : *,+,a,b,+,a,c
Обратный обход : a,b,+,a,c,+,*
Симметричный обход : a,+,b,*,a,+,c

В случае прямого обхода дерева мы получаем префиксную форму выражения, где оператор предшествует левому и правому операндам. Такая форма выражения называется прямой польской записью.
Дадим точное описание префиксной формы выражений.

Считаем, что префиксным выражением, одиночного операнда является сам операнд; далее префиксная форма для выражения (E1)q(E2), где q – бинарный оператор имеет вид q,P1,P2, где P1,P2-префиксные формы для E1,E2, а префиксная форма выражения q(E1), где q – унарный оператор имеет вид q,P1, где P1 -префиксная форма для E1.

Отметим, что в префиксных формах нет необходимости отделять или выделять отдельные выражения скобками, т.к. всегда можно просмотреть префиксное выражение qP1P2 и определить единственным образом P1.

Обратное упорядочивание меток даёт постфиксное представление выражения, в этом представлении оператор следует сразу за операндами. Такая форма выражения называется обратной польской записью.

Дадим точное описание постфиксной формы или обратной польской записи выражений.

Считаем, что постфиксным выражением, одиночного операнда является сам операнд; далее постфиксная форма для выражения (E1)q(E2), где q – бинарный оператор имеет вид P1,P2, q, где P1,P2-постфиксные формы для E1,E2, а префиксная форма выражения q(E1), где q – унарный оператор имеет вид P1, q, где P1 -префиксная форма для E1.

Обратная польская запись так же не требует скобок.

Отметим, что в префиксных и постфиксных формах нет необходимости отделять или выделять отдельные выражения скобками.

При симметричном обходе мы получим стандартную запись выражения, хотя и без скобок.

Алгоритмы прямого и обратного обходов дерева, использующие рекурсивные функции

Procedure PR (n-Узел)
Begin
(1)занести в список обхода узел n
(2)for для каждого сына с узла n в порядке с лева на право do вызвать процедуру PR(do_RR)
End;

В этом схематически показанном наброске процедуры PR(прямое упорядочивание)

Составляющей список узлов дерева при обходе его в прямом порядке достаточно поменять местами строки 1 и 2 чтобы получить процедуру выполняющую обход дерева обратном порядке.

Алгоритм симметричного обхода дерева, использующий рекурсивную функцию

Procedure IN(n-узел)
Begin
If n-листок, then занести в список обхода узел n
else begin
(вызываем процедуру для самого левого)
In(самый левый сын узла n)
Заносим в список обхода n;
For для каждого сына e узла n, исключая самый левый, в порядке слева на право выполняется do In(c);
End;

End;

Алгоритм прямого обхода дерева, не использующий рекурсивную функцию

В этом алгоритме используется стек.

Прямой обход
Шаг 0:заносим в стек корень дерева
Шаг 1:Если на вершине стека находится узел а, удаляем узел а из стека, помещаем его в список обхода и если у а есть сыновья, то заносим их в стек.
Шаг 2:если стек не пуст, то переходим к Шагу 1, если пуст - завершаем работу.

Алгоритм прямого обхода дерева, не использующий рекурсивную функцию
В этом алгоритме так же используется стек, кроме того узлы, попавшие в стек метятся.
Шаг 0:заносим в стек корень дерева и метим его
Шаг 1: Пусть на вершине стека находится узел а. Если узел а имеет непомеченных сыновей, то их помещаем в стек и метим. Если узел а не имеет непомеченных сыновей, то удаляем узел а из стека и помещаем его в список обхода.
Шаг 2:Если стек не пуст, то переходим к Шагу 1,если пуст, то завершаем работу.

Алгоритм симметричного обхода дерева, не использующий рекурсивную функцию

Шаг 0. Помещаем в стек корень дерева и метим его.

Шаг 1. Пусть узел *N* находится на вершине стека. Если *N* – лист, то удаляем его из стека и помещаем в список обхода. Если *N* – не лист, то рассматриваем его левого сына *‘c’*. Если *‘c’* не помечен, то помещаем его в стек и метим, если *‘c’* помечен, то узел *N* удаляем из стека и помещаем в список обхода, а в стек помещаем всех сыновей узла *N* кроме левого и метим их.

Шаг 2. Проверяем стек. Если он не пуст переходим к шагу 1, если пуст – завершаем программу.

Программа, построенная по алгоритму симметричного обхода дерева, не использующего рекурсивную функцию:

string f(TreeNode t) // t- это ссылка на корень дерева

```
{
  string List=" "// список обхода будем формировать, как список меток его
узлов
ArrayList  St=new ArrayList(); // список St мы будем использовать, как стек
TreeNode p=t;
St.Add(p); p.Tag=1;// помещаем корень дерева в стек и метим его
while (p!=null)
{
if (p.Nodes.Count = = 0) {St.Remove(p); List=List+p.Text+" ";}
else { TreeNode c=p.Nodes[0];
if (c.Tag = = null) {St.Add(c); c.Tag=1;}
else {St.Remove(p); List=List+p.Text+" ";

for(int i=1; i<p.Nodes.Count; i++) // помещает в стек всех сыновей кроме
левого
{ St.Add(p.Nodes[i]) p.Nodes[i]. Tag=1;}
if(St.Count = = 0) return List; // если стек пуст, то мы завершаем работу,
возвращая в качестве ответа, список меток всех узлов
p=(TreeNode) St.LastNode();
}
return List;
}
```

Задание

1. Написать арифметическое выражение, содержащее 15 символов. Это выражение должно обязательно содержать скобки.
2. Построить дерево синтаксического разбора придуманного выражения.
3. На этапе проектирования программы ввести построенное дерево синтаксического разбора в компонент TreeView.
4. Написать и отладить функцию, выполняющую обход дерева по алгоритму, определенному вариантом задания.
5. Поместить на форму кнопку, выполняющую обход введенного в компонент TreeView дерева. Результат обхода должен выводиться на форму.

Варианты алгоритмов обхода дерева:

1. Алгоритм прямого обхода, не использующий рекурсивные функции.
2. Алгоритм обратного обхода, не использующий рекурсивные функции.
3. Алгоритм прямого обхода, использующий рекурсивную функцию.
4. Алгоритм обратного обхода, использующий рекурсивную функцию.
5. Алгоритм симметричного обхода, использующий рекурсивную функцию.

Контрольные вопросы

1. Что такое обход дерева?
2. Какой обход называется прямым?
3. Какой обход называется обратным?
4. Какой обход называется симметричным?
5. Выполнить обход заданного дерева в прямом порядке.
6. Выполнить обход заданного дерева в обратном порядке.
7. Выполнить обход заданного дерева в симметричном порядке.
8. При каком обходе первым проходится корень?
9. При каком обходе корень проходится в последнюю очередь?
10. Дайте определение польской записи арифметического выражения.
11. Нужны ли скобки при постфиксной записи выражения?
12. Как строится дерево синтаксического разбора арифметического выражения.
13. Какое свойство компонента `TreeView` позволяет обратиться к дереву?
14. Продемонстрировать, как строится дерево на этапе проектирования программы.

План отчета

1. Титульный лист.
2. Цель работы.
3. Придуманное арифметическое выражение.
4. Дерево синтаксического разбора арифметического выражения.
5. Теоретический материал, используемый при выполнении задания.
6. Распечатка программы.

Этапы выполнения лабораторной работы

1. Изучить теоретический материал.
2. Ответить на контрольные вопросы.
3. Придумать арифметическое выражение.
4. Построить его дерево синтаксического разбора.
5. На этапе проектирования программы ввести полученное дерево в компонент *TreeView*.
6. Написать программу, решающую поставленную задачу.
7. Отладить программу
8. Написать отчет.