Министерство Образования и Науки Российской Федерации  
Новосибирский Государственный Технический Университет  
Кафедра Прикладной Математики

**Лабораторная работа № 3**

**по дисциплине «Структуры данных и алгоритмы»**

Факультет: ФПМИ

Группа: ПМ-53

Студент: Тябин Егор Алексеевич

Преподаватель: Еланцева Ирина Леонидовна

### Новосибирск 2016

# Условие задачи

Формула вида:  
<Формула>::=<терминал>| (<формула><знак><формула>)  
<знак>::= +| -| \*  
<терминал>::= 0| 1| 2| 3| 4| 5| 6| 7| 8| 9  
можно представить в виде двоичного дерева (‘дерева-формулы’):  
- формула из одного терминала (цифры) представляется деревом из одной вершины(корнем) с этим терминалом;  
-формула вида (f1 s f2) – деревом в котором корень – это знак s, а левое и правое поддеревья – это соответствующие представления f1 и f2:  
Проверить является ли заданное двоичное дерево(с элементами типа char) деревом формулой и напечатать его элементы в порядке 1) прямого 2) обратного 3)концевого обхода;

# 2. Анализ задачи

Дано:

Формула представленная в виде двоичного дерева:  
<Формула>::=(<элемент>,<формула>,<формула>) | (<элемент>, ?, ?)  
<элемент>::= (| )| +| -| \*| ?| 0| 1| 2| 3| 4| 5| 6| 7| 8| 9|   
Для представления пустого дерева нужно ввести “?”.

Результат:

Последовательность символов.

## Метод решения

Для работы с деревом введем подзадачи работы со стеком и работы с деревом.

## Выделение основных подзадач

1. Очистка очереди
2. Проверка очереди на пустоту
3. Добавление элемента в очередь
4. Взятие элемента из очереди
5. Построение дерева
6. Прямой обход
7. Обратный обход
8. Концевой обход

1) Очистка очереди. Так как очередь задана линейным массивом с указателем на конец, то достаточно положить конец на начало очереди(массива), т.е. на первый элемент.

2) Проверка очереди на пустоту. Так как мы имеем указатель на конец очереди, нужно проверить, стоит ли указатель на первом элементе очереди(массива). Если да, то очередь пуста, иначе не пуста.  
3) Добавление элемента в очередь. В очередь элементы добавляются в конец. Так как мы имеем указатель на конец, то элемент добавляется на место указателя, а сам указатель передвигается на следующее место.

4) Взятие элемента из очереди. В очереди элементы берутся из начала. Берется первый элемент в очереди, потом каждый элемент передвигается на место предыдущего, как и указатель. После чего взятый элемент распечатывается.  
5) Построение дерева. Считывается символ. Если ‘(‘, то считывается следующий символ и заносится в дерево. Строим левое поддерево рекурсивно, потом строим правое поддерево рекурсивно. Считываем символ.  
Если ‘?’, то возвращается значение NULL в поддерево.  
Если ‘,’, то строим поддерево рекурсивно.  
6)Прямой обход. Рекурсивно выводим дерево прямым обходом. Корень, левое поддерево в прямом порядке, правое поддерево в прямом порядке.  
7)Обратный обход. Выводим дерево обратным обходом, параллельно определяя, является ли дерево деревом формулой. Левое поддерево в обратном порядке, корень, правое поддерево в обратном порядке. В обратном обходе дерево-формула должно выглядеть так: <число><знак><число><знак>…<число>. И мы проверяем похожа ли полученная последовательность символов на эту.  
9)Концевой обход. Рекурсивно выводим дерево концевым обходом. Левое поддерево в концевом порядке, правое поддерево в концевом порядке, корень.

Следовательно, для решения задачи мы сначала строим дерево, потом обходим дерево прямым порядков, далее обходим дерево обратным порядком параллельно определив, является ли дерево деревом-формулой, и наконец обходим дерево концевым порядком.

# 3. Структуры данных, используемые для представления исходных данных и результатов задачи

## Внешнее представление входных данных

Смотреть дано.

## Внешнее представление выходных данных

Последовательность символов, выводимых на экране в ходе работы программы.

# Внутреннее представление входных данных

Struct tree {char data; tree \*left, \*right} – структура tree, состоящий из 3 полей: элемент типа char, указатель на левое и правое дерево.

## Внутреннее представление данных

Struct stack{tree \*elem; stack \*next} – Структура stack, состоящий из 3 полей: указатель на элемент типа tree и указатель на следующий элемент в структуре stack.  
Stack \*&top – указатель на элемент типа stack, который используется в вызываемых функциях для работы со стеком. После окончания функции элемент запоминается.  
Char input – элемент типа char, используется в функции построения дерева для запоминания вводимого символа.  
tree \*d – указатель на дерево.  
tree \*T – указатель на элемент типа tree, используется в подзадачах обходов, для указывания на дерево d.  
int flag – целочисленный элемент, который используется чтобы выйти из цикла.  
int i – целочисленный элемент, который используется определения четности места на котором стоит элемент дерева.  
int flaq – целочисленный элемент, который используется для определения дерева формулы. Становится равным 0, если обнаруживается признак дерева не формулы.  
FILE \*fpp – указатель на поток чтения файла input.txt  
stack \*st – указатель на стек.

# 4. Укрупненный алгоритм решения задачи

Алг buildtree(fp) {  
Ввод input;  
Если input=’(‘ : T=new tree;  
 Ввод input;  
 T.[data]=input;  
 T[left]=buildtree(fp);  
 T[right]=buildtree(fp);  
 Ввод input;  
 Возврат T;  
Если input=’?’: Возврат NULL;  
Если input=’,’ : d=buildtree; Возврат d; }

Алг infix(T, S) {  
Вывод «Обратный обход»  
flag=1; flaq=2;  
Повторять:  
 Повторять: input(T, S);  
 T=T[left];  
 Пока T≠NULL;  
 Если Pusta(S) = 0, то: T=output(S);  
 Вывод T[data];  
 Если (i%2=0 и (T[data]>’9’ или T[data]<’0’)), то: flaq=0;  
 Если (i%2≠0 и (T[data] ≠ ’\*’ и T[data] ≠ ’-‘ и T[data] ≠ ‘+’)), то flaq=0;   
 i=i+1;  
 T=T[right];  
 Иначе: flag=0;  
Пока flag=1;  
Если (i%2=0), то flaq=0;  
Если (flaq=0), то Вывод: Не является деревом-формулой;  
Иначе: Вывод: Является деревом-формулой;

# Структура программы

## Взаимосвязь функций

Void main()

void infix(tree \*T, stack \*S);

void input(tree \*x, stack \*&top);  
tree \*output(stack \*&top)

tree \*buildtree(FILE \*&fp);

Tree \*buildtree(FILE \*&fp);

void prefix(tree \*T);

void prefix(tree \*T);

void postfix(tree \*T)

void postfix(tree \*T)

## Составные части программы

Наименование функции:   
Pusta  
Прототип функции:  
int Pusta(stack \*top);  
Данная подпрограмма проверяет очередь на пустату, и возвращает 1 или 0, если список пуст и не пуст соответственно.

Наименование функции:  
Ochistka  
Прототип функции:  
void Ochistka(stack \*top);  
Данная подпрограмма очищает список, положив указатель на начало массива.

Наименование функции:  
input  
Прототип функции:  
void input(tree \*x, stack \*&top);   
Данная подпрограмма добавляет указанное число в конец очереди. И возвращает 1 или 0, если добавить удалось или нет соответственно.

Наименование функции:  
output  
Прототип функции:  
tree \*output(stack \*&top)  
Данная подпрограмма берет число из начала очереди(num), сдвигая другие элементы. Возвращает дерево.

Наименование функции:  
buildtree  
Прототип функции:  
tree \*buildtree(FILE \*&fp)  
Данная подпрограмма строит дерево.

Наименование функции:  
prefix  
Прототип функции:   
void prefix(tree \*T)  
Данная подпрограмма выводит последовательность символов, которые являются элементами дерева, в прямом порядке.

Наименование функции:  
infix  
Прототип функции:  
void infix(tree \*T, stack \*S)  
Данная подпрограмма выводит последовательность символов, которые являются элементами дерева, в обратном порядке. Параллельно проверяет, является ли дерево деревом формулой.

Наименование функции:  
postfix  
Прототип функции:  
void postfix(tree \*T)  
Данная подпрограмма выводит последовательность символов, которые являются элементами дерева, в концевом порядке.

# Текст программы

**/\* Файл global.h \*/**

#ifndef GLOBAL\_H

#define GLOBAL\_H

struct tree { char data; tree \*left, \*right; };

struct stack { tree \*elem; stack \*next; };

#endif

**/\* Файл stack.h \*/**  
#include "global.h"

#ifndef OPERATIONS\_H

#define OPERATIONS\_H

int Pusta(stack \*top);

void Ochistka(stack \*&top);

void input(tree \*x, stack \*&top);

tree \*output(stack \*&top);

// void Read(char te, FILE \*f, stack \*&top);

// void Vivod(FILE \*f, stack \*&top);

#endif

**/\* Файл tree.h \*/**  
#include "global.h"

#ifndef TREE\_H

#define TREE\_H

tree \*buildtree(FILE \*&fp);

void prefix(tree \*T);

void infix(tree \*T, stack \*S);

void postfix(tree \*T);

#endif

**/\* Файл stack.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include "stack.h"

int Pusta(stack \*top) {

if (top == NULL) return 1;

else return 0;

}

void Ochistka(stack \*&top){

top = NULL;

}

void input(tree \*x, stack \*&top){

stack \*w;

w = new stack;

w->elem = x; w->next = top; top = w;

}

tree \*output(stack \*&top){

tree \*num;

stack \*w;

num = top->elem;

w = top;

top = top->next;

delete w;

return num;

}

**/\* Файл tree.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include "stack.h"

#include "tree.h"

tree \*buildtree(FILE \*&fp)

{

char input;

tree \*T;

fscanf(fp, "%c", &input);

switch (input){

case '(': { T = new tree;

fscanf(fp, "%c", &input); T->data = input;

T->left = buildtree(fp);

T->right = buildtree(fp); fscanf(fp, "%c", &input);

return T;

}

case '?': return NULL;

case ',': d = buildtree(fp); return d;

}

}  
void prefix(tree \*T)

{

if (T != NULL) {

printf\_s("%c", T->data); prefix(T->left);

prefix(T->right);

}

}

void infix(tree \*T, stack \*S)

{

printf\_s("\n\n Обратный обход: ");

int flag = 1, i=2, flaq=1;

while (flag)

{

while (T != NULL)

{

input(T, S);

T = T->left;

}

if (!Pusta(S))

{

T = output(S);

printf\_s("%c", T->data);

if (i % 2 == 0 && (T->data<'0' || T->data>'9')) { flaq = 0; }

if (i % 2 != 0 && (T->data != '\*' && T->data != '-' && T->data != '+')) { flaq = 0; }

i++;

T = T->right;

}

else flag = 0;

}

if (i % 2 == 0) flaq = 0;

if (flaq == 0) printf("\n Не является деревом-формулой");

else printf("\n Является деревом-формулой");

}

void postfix(tree \*T)

{

if (T != NULL){

postfix(T->left); postfix(T->right);

printf("%c", T->data);

}

}

**/\* Файл main.cpp \*/**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <locale.h>

#include <stdio.h>

#include <conio.h>

#include "stack.h"

#include "global.h"

#include "tree.h"

void main(){

setlocale(LC\_CTYPE, "russian");

FILE \*fpp;

tree \*d;

stack \*st;

d = new tree;

Ochistka(st);

fpp=fopen("input.txt", "r");

d = buildtree(fpp);  
 if(d==NULL) printf(“Дерево пустое”);

else {

printf\_s("\n\n Прямой обход: ");

prefix(d);

infix(d, st);

printf("\n\n Концевой обход: ");

postfix(d); }

\_getch();  
}

# 7.Тесты

Тест 1.  
Назначение: все элементы являются вопросиками.  
Входные данные:   
(?,?,?)  
Результат:  
Прямой обход: ?  
Обратный обход: ?  
Не является деревом-формулой  
Концевой обход: ?

Тест 2.  
Назначение: проверка работоспособности на правильно заданном дереве-формуле.  
Входные данные:  
(\*,(-,(6,?,?),(5,?,?)),(+,(1,?,?),(0,?,?)))  
Результат:  
Прямой обход: \*-65+10  
Обратный обход: 6-5\*1+0  
Является деревом формулой  
Концевой обход: 65-10+\*

Тест 3.  
Назначение: проверка на отсутствие поддерева.  
Входные данные:  
(+,(5,?,?),?)  
Результат:  
Прямой обход: +5  
Обратный обход: 5+  
Не является деревом формулой  
Концевой обход: 5+

Тест 4.  
Назначение: супер дерево.  
Входные данные:  
(\*,(-,(-,(\*,(1,?,?),(2,?,?)),(5,?,?)),(+,(3,?,?),(8,?,?))),(\*,(-,(0,?,?),(4,?,?)),(+,(9,?,?),(7,?,?)))  
Результат:  
Прямой обход: \*--\*125+38\*-04+97  
Обратный обход: 1\*2-5-3+8\*0-4\*9+7  
Является деревом-формулой  
Концевой обход: 12\*5-38+-04-97+\*\*

Тест 5.  
Назначение: если дерево пустое.  
Входные данные:  
?  
Результат:  
Дерево пустое

# 8. Результат работы программы

-Программа выдала верное решение на всех тестах и, следовательно, правильно работает.