МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Бинарные Деревья

Студент гр. 9304 _____ Мохаммед А.А.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург

2020

ЦЕЛЬ РАБОТЫ

Цель работы: Познакомиться со структурой данных – деревом, освоить на практике использование деревьев для решения задач.

Вариант 15.

Формулировка задачи:
Формулу вида

< формула > ::= < терминал > | (< формула > < знак > < формула >)

< знак > ::= + | - | *

< терминал > ::= 0 | 1 | ... | 9 | *a* | *b* | ... | *z*

можно представить в виде бинарного дерева («*дерева-формулы*») с элементами типа *char* согласно следующим правилам:

- формула из одного терминала представляется деревом из одной вершины с этим терминалом;

- формула вида (f_1 *s* f_2) представляется деревом, в котором корень это знак *s*, а левое и правое поддеревья соответствующие представления формул f_1 и f_2. Например, формула (5 * (*a* + 3)) представляется деревом-формулой, показанной на рис. 1.

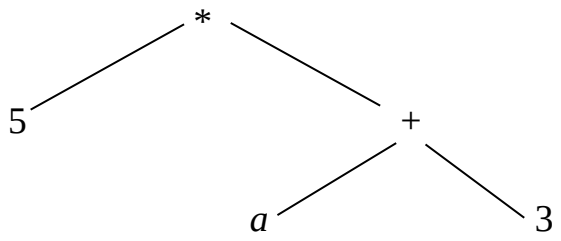

Рис. 1.

Требуется:
а) для заданной формулы *f* построить дерево-формулу *t*;
б) для заданного дерева-формулы *t* напечатать соответствующую формулу *f*;
в) с помощью построения дерева-формулы *t* преобразовать заданную формулу *f* из инфиксной формы в префиксную (перечисление узлов *t* в порядке КЛП) или в

постфиксную (перечисление в порядке ЛПК);

з) преобразовать дерево-формулу t, заменяя в нем все поддеревья, соответствующие формулам $((f_1 * f_2) + (f_1 * f_3))$ и $((f_1 * f_3) + (f_2 * f_3))$, на поддеревья, соответствующие формулам $(f_1 * (f_2 + f_3))$ и $((f_1 + f_2) * f_3)$;

РЕАЛИЗАЦИЯ ЗАДАЧИ
ИСПОЛЬЗУЕМЫЕ СТРУКТУРЫ ДАННЫХ

node – класс, который редставляет из себя узел бинарного дерева в ссылочной реализации. Поля структуры *node:*

- *info* – значение узла, имеет тип *base* (определен как *char*).
- *lt* – указатель на левого сына.
- *rt* – указатель на правого сына.

Методы класса *node:*

- Конструктор *node* – обнуляет указатели на детей.

ИСПОЛЬЗУЕМЫЕ ФУНКЦИИ

Create – возвращает пустое бинарное дерево.

isNull – принимает узел бинарного дерева, возвращает *true,* если узел пустой*, u false,* если нет.

RootBT – принимает узел дерева, возвращает значение этого узла.

Left – принимает узел дерева, возвращает левого сына.

Right – принимает узел дерева, возвращает правого сына.

ConsBT – принимает значение типа *base* и два узла дерева. Создает новый узел с полученным значением, для которого полученные узлы являются детьми, и возвращает этот узел.

destroy – принимает дерево, очищает память, занятую им. Не возвращает ничего.

isEqual – принимает два узла, возвращает *true*, если деревья, корнями которых являются принятые узлы, равны, и *false*, если нет.

isTerminal – принимает символ и возвращает *true*, если он является терминалом, или *false*, если нет.

isSign – принимает символ и возвращает *true*, если он является знаком, или *false*, если нет.

MakeNode – принимает символ, создает бездетный узел дерева, значение которого равно этому символу, и возвращает созданный узел.

PrintBt – принимает бинарное дерево и выводит его скобочную запись.

SkipSpaces – принимает поток ввода и считывает оттуда все пробелы, идущие подряд.

TreeToForm – принимает дерево-формулу, создает строку с соответствующей этому дереву формулой, и возвращает созданную строку.

FormToTree – принимает поток ввода, считывает оттуда формулу, строит по ней дерево-формулу и возвращает построенное дерево.

Transform – принимает узел дерева и выполняет над ним преобразования, описанные в пункте «з».

ProcTree – принимает дерево-формулу и выполняет над всеми его узлами, если это возможно, преобразования, описанные в пункте «з».

Translate – принимает дерево-формулу, создает строку, содержащую префиксную запись формулы, соответствующей полученному дереву, и возвращает созданную строку (префиксную запись формулы).

print – получает дерево, делает его отрисовку.

ТЕСТИРОВАНИЕ

РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Тестовые случаи представлены в приложении Б. Ошибки выявить не удалось. По результатам тестирования было показано, что задача выполнена.

ВЫВОД

В ходе работы была написана программа на языке C++, которая строит дерево-формулу, соответствующую формуле, выполняет на нем преобразования, преобразует формулу в инфиксной форме в формулу в префиксной с помощью дерева и отрисовывает дерево. Был получен опыт в использовании бинарных деревьев в ссылочной реализации.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл MakeFile:

```
all: main.o bt_implementation.o functions.o
	g++ main.o bt_implementation.o functions.o -o a.out

main.o: main.cpp Btree.h functions.h g++
	-c main.cpp
functions.o: functions.cpp functions.h Btree.h
	g++ -c functions.cpp

bt_implementation.o: bt_implementation.cpp
	Btree.h g++ -c bt_implementation.cpp
clean:
	rm *.o bt_implementation
```

Файл btree.h:

```
#pragma once

namespace binTree_modul
{

//-------------------------------------
  typedef char base;

  struct node {
    base info;
    node *lt;
    node *rt;
    node ()
    {
      lt = NULL; rt = NULL;
    }
  };

  typedef node *binTree;

  binTree Create(void);
  bool isNull(binTree);
  base RootBT (binTree);
  binTree Left (binTree);
  binTree Right (binTree);
  binTree ConsBT(const base x, binTree lst,  binTree rst);
  void destroy (binTree&);
```

```
  bool isEqual(binTree a,  binTree b);

} // end of namespace bTree_modul
```

Файл functions.h:

```
#pragma once

#include <iostream>
#include "btree.h"

using namespace std;
using namespace binTree_modul;

namespace lab3_namespace
{
  bool isTerminal( const char c );
  bool isSign( const char c );
  binTree MakeNode( base c );
  void PrintBt(binTree b);
  void SkipSpaces(istream &in);
  string TreeToForm(binTree b);
  binTree FormToTree(istream &in);
  void Transform(binTree b);
  void ProcTree(binTree b);
  string Translate(binTree b);
  void print(binTree f, int l);
}
```

Файл bt_implementation.cpp:

```
#include <iostream>
#include <cstdlib>
#include "btree.h"
using namespace std;

namespace binTree_modul
{
//-------------------------------------
  binTree Create()
  { return NULL;
  }
//-------------------------------------
  bool isNull(binTree b)
  { return (b == NULL);
  }
//-------------------------------------
```

```
  base RootBT (binTree b)
  { if (b == NULL) { cerr << "Error: RootBT(null) \n"; exit(1); }
    else return b->info;
  }
//-------------------------------------
  binTree Left (binTree b)
  { if (b == NULL) { cerr << "Error: Left(null) \n"; exit(1); }
    else return b ->lt;
  }
//-------------------------------------
  binTree Right (binTree b)
  { if (b == NULL) { cerr << "Error: Right(null) \n"; exit(1); }
    else return b->rt;
  }
//-------------------------------------
  binTree ConsBT(const base x, binTree lst,  binTree rst)
  { binTree p;
    p = new node;
    if ( p != NULL) {
      p ->info = x;
      p ->lt = lst;
      p ->rt = rst;
      return p;
    }
    else {cerr << "Memory not enough\n"; exit(1);}
  }

  bool isEqual(binTree a,  binTree b)
  {
    if (a == NULL && b == NULL)
      return true;
    if (a == NULL || b == NULL)
      return false;
    return isEqual(a->lt, b->lt) && isEqual(a->rt, b->rt) && a->info == b->info;
  }

//-------------------------------------
  void destroy (binTree &b)
  { if (b != NULL)  {
      destroy (b->lt);
      destroy (b->rt);
      delete b;
      b = NULL;
    }
  }

} // end of namespace
```

Файл functions.cpp:

```cpp
#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "btree.h"

using namespace std;
using namespace binTree_modul;

// display the tree as a tree
// more informative errors

namespace lab3_namespace
{
  bool isTerminal( const char c )
  {
    return (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9');
  }

  bool isSign( const char c )
  {
    return c == '-' || c == '+' || c == '*';
  }

  binTree MakeLeaf( base c )
  {
    return ConsBT(c, nullptr, nullptr);
  }

  void PrintBt(binTree b)
  {
    if (b == NULL)
      return;

    if (b->lt == NULL && b->rt == NULL){
      cout << ' ' << b->info << ' ';
      return;
    }

    cout << "( ";
    PrintBt(b->lt);
    cout << ' ' << b->info << ' ';
    PrintBt(b->rt);
    cout << " )";
  }

  void SkipSpaces(istream &in)
  {
```

```
  base c;
  do
  {
    c = in.peek();
    if (c == ' ')
      c = in.get();
  } while (c == ' ');
}

string TreeToForm(binTree b)
{
  string str = "";
  if (b == NULL)
    return str;

  if (b->lt == NULL && b->rt == NULL)
  {
    str += ' ';
    str += b->info;
    str += ' ';
    return str;
  }

  str += "( ";
  str += TreeToForm(b->lt);
  str += ' ';
  str += b->info;
  str += ' ';
  str += TreeToForm(b->rt);
  str += " )";
  return str;
}

binTree FormToTree(istream &in)
{
  base c, sign;
  binTree left, right;

  SkipSpaces(in);
  c = in.get();

  if (isTerminal(c))
    return MakeLeaf(c);

  if (c == '('){
    SkipSpaces(in);
    left = FormToTree(in);

    SkipSpaces(in);
    c = in.get();
```

```
    if (isSign(c))
      sign = c;
    else { cerr << "error: sign expected" << endl; return NULL; }

    SkipSpaces(in);
    right = FormToTree(in);

    SkipSpaces(in);
    c = in.get();
    if (c != ')') { cerr << "error: ')' expected" << endl; return NULL; }


    left = ConsBT(sign, left, right);
    return left;
  }
  else {
    if (isSign(c)) cerr << "error: extra sign in formula" << endl;
    else cerr << "error: external symbol if formula" << endl;
    return FormToTree(in);
    //return NULL;
   }
}


 void Transform(binTree b)
 {
// ((f1 * f2) + (f1 * f3)) -> (f1 * (f2 + f3))
  if (isEqual(b->lt->lt, b->rt->lt)){
    binTree f1, f2, f3;

    f1 = b->lt->lt;
    f2 = b->lt->rt;
    f3 = b->rt->rt;

    b->info = '*';
    b->lt = f1;
    b->rt->info = '+';
    b->rt->lt = f2;
    b->rt->rt = f3;

    return;
  }
  // ((f1 * f2) + (f3 * f1)) -> (f1 * (f2 + f3))
  if (isEqual(b->lt->lt, b->rt->rt)){
    binTree f1, f2, f3;

    f1 = b->lt->lt;
    f2 = b->lt->rt;
    f3 = b->rt->lt;

    b->info = '*';
```

```
    b->lt = f1;
    b->rt->info = '+';
    b->rt->lt = f2;
    b->rt->rt = f3;

    return;
  }
  // ((f2 * f1) + (f1 * f3)) -> (f1 * (f2 + f3))
  if (isEqual(b->lt->rt, b->rt->lt)){
    binTree f1, f2, f3;

    f1 = b->lt->rt;
    f2 = b->lt->lt;
    f3 = b->rt->rt;

    b->info = '*';
    b->lt = f1;
    b->rt->info = '+';
    b->rt->lt = f2;
    b->rt->rt = f3;

    return;
  }
  // ((f2 * f1) + (f3 * f1)) -> (f1 * (f2 + f3))
  if (isEqual(b->lt->rt, b->rt->rt)){
    binTree f1, f2, f3;

    f1 = b->lt->rt;
    f2 = b->lt->lt;
    f3 = b->rt->lt;

    b->info = '*';
    b->lt = f1;
    b->rt->info = '+';
    b->rt->lt = f2;
    b->rt->rt = f3;

    return;
  }

}


void ProcTree(binTree b)
{
  if (b == NULL)
    return;
  ProcTree(b->lt);
  ProcTree(b->rt);

  if (b == NULL ||
```

```
    b->lt == NULL || b->lt->lt == NULL || b->lt->rt == NULL ||
    b->rt == NULL || b->rt->lt == NULL || b->rt->rt == NULL)
    return;

  if (b->lt->info != '*' || b->rt->info != '*' || b->info != '+')
    return;


  Transform(b);

}

string Translate(binTree b)
{
  string str = "";

  if (b == NULL)
    return str;
  str += ' ';
  str += b->info;
  str += ' ';
  if (b->lt != NULL){
    str += ' ';
    str += Translate(b->lt);
    str += ' ';
  }
  if (b->rt != NULL){
    str += ' ';
    str += Translate(b->rt);
    str += ' ';
  }
  return str;
}

void print(binTree f, int l)
{
  if(f == nullptr){
    for(int i = 0; i < l; i++)
      cout << "\t";
    cout << '#' << endl;
    return;
  }

  print(f->rt, l+1);

  for(int i = 0; i < l; i++)
    cout << "\t";

  cout << f->info << endl;
```

```
    print(f->lt, l+1);
    }
}
```

Файл main.cpp:

```cpp
#include <iostream>
#include <fstream>
#include <fstream>
#include <cstdlib>
#include <cstring>
#include "btree.h"
#include "functions.h"


using namespace std ;
using namespace binTree_modul;
using namespace lab3_namespace;

int main ()
{
  string formula;
  binTree b;

  cout << "Please write formula :\n";
  b = FormToTree(cin);
  cout << "The formula tree is built:" << endl;
  formula = TreeToForm(b);
  cout << formula << endl;

  cout << "Transforming a formula tree..." << endl;
  ProcTree(b);
  formula = TreeToForm(b);
  cout << formula << endl;

  cout << "Prefix Formula Tree..." << endl;
  formula = Translate(b);
  cout << formula << endl;

  cout << "Formula tree visualization: " << endl;
  print(b, 0);

  destroy (b);
  cout << endl;
  return (0);
}
```

ПРИЛОЖЕНИЕ Б
ТЕСТОВЫЕ СЛУЧАИ

Входное выражение	**Вывод программы**	**Корректность выполнения**
(a + b)	Please write formula : (a+b) The formula tree is built: (a + b) Transforming a formula tree... (a + b) Prefix Formula Tree... + a b Formula tree visualization: # b # + # a #	да
(a ++ b)	Please write formula : (a++b) error: extra sign in formula The formula tree is built: (a + b) Transforming a formula tree... (a + b) Prefix Formula Tree... + a b Formula tree visualization: # b # + # a #	да

(a + b	error: ')' expected The formula tree is built: Transforming a formula tree... Prefix Formula Tree... Formula tree visualization:	да
(((a+b)*c)+(3*(a+b)))	Please write formula : (((a+b)*c)+(3*(a+b))) The formula tree is built: (((a + b) * c) + (3 * (a + b))) Transforming a formula tree... ((a + b) * (c + 3)) Prefix Formula Tree... * + a b + c 3 Formula tree visualization: # 3 # + # c # * # b # + # a # 	да

a+b	Please write formula : a+b The formula tree is built: a Transforming a formula tree... a Prefix Formula Tree... a Formula tree visualization: # a #	да