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: Деревья

Студент гр. 9382 _____ Кузьмин Д. И.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург
2020

Цель работы.

Познакомиться с такой нелинейной структурой данных, как бинарное дерево, способами её представления и реализации, получить навыки решения задач обработки бинарных деревьев.

Основные теоретические положения.

Деревом называется конечное множество T, состоящее из одного или более узлов, таких, что а) имеется один специально обозначенный узел, называемый корнем данного дерева; б) остальные узлы (исключая корень) содержатся в m ≥ 0 попарно не пересекающихся множествах T1, T2, .., Tm, каждое из которых, в свою очередь,является деревом (поддеревом дерева T).

Бинарное дерево это конечное множество узлов, которое либо пусто, либо состоит из корня и двух непересекающихся бинарных деревьев, называемых правым поддеревом и левым поддеревом.

Задание.

11–17. 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 *Elem=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. 3.12.

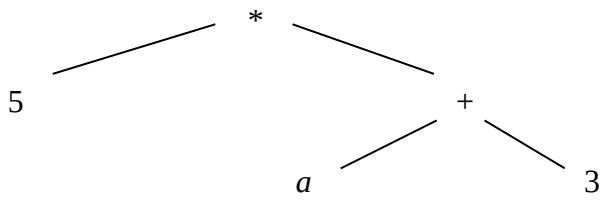

Рис. 3.12. Дерево-формула

Требуется:

- Для всех вариантов (11–17):

- для заданной формулы *f* построить дерево-формулу *t*;

- для заданного дерева-формулы *t* напечатать соответствующую формулу *f*;

Вариант 12в:

- построить дерево-формулу *t* из строки, задающей формулу в префиксной форме (перечисление узлов *t* в порядке КЛП);

- преобразовать дерево-формулу *t*, заменяя в нем все поддеревья, соответствующие формуле (*f + f)*, на поддеревья, соответствующие формуле (*2 * f)*.

Описание функций и структур данных

1) В работе используется бинарное дерево, реализованное на базе массива и описанное в классе binaryTree

2) Данный класс имеет поля: char* elementsArray – массив элементов дерева, int size – размер массива.

3) Элементы хранятся в массиве по следующему принципу: элемент с номером I имеет потомков с индексами 2*i+1 для левого и 2*i+2 для правого. Например: элемент 0 (корень) имеет потомков с номерами 1 и 2. При этом некоторые элементы массива могут быть пустыми. Для них выделен специальный символ “_”.

4) Класс имеет некоторые функции. Среди них:

bool isLeaf(int index) — где index – номер элемента массива. Функция проверяет, является ли элемент «листом».

int getRightElemIndex(int index), где index – индекса элемента массива. Функция возвращает индекс правого потомка выбранного элемента.

int getLeftElemIndex(int index) - аналогично для левого.

bool areEqualElements(int index1, int index2), где index1 и index2- номера проверяемых элементов. Функция проверяет, являются ли элементы идентичными(совпадают не только значения корней, но и дальнейшие потомки).

Void transform() - функция, осуществляющая замену всех поддеревьев вида (f + f) на (2 * f).

Описание алгоритма(построение формулы-дерева)

1) Построение формулы-дерева осуществляется аналогично обходу вида ЛКП.

2) Начиная с первого элемента(перед ним нужно поставить скобку). Будут выведены сначала левый потомок, затем корневое значение, затем правый потомок и закрывающая скобка.

3) В случае, если какой-то потомок является поддеревом, то для него проделываются те же действия.

Описание алгоритма(преобразование поддеревьев)

1) Алгоритм проверят каждый элемент дерева. Если он является поддеревом(исходное дерево здесь тоже считается поддеревом), то проверятся во-первых значение корня(должно быть +), а также равенство соответственно левого и правого потомка.

2) Если такое равенство имеет место, то значение корня заменяется на * , а зачение левого потомка на 2 и он становится «листом».

3) Алгоритм завершается, после просмотра последнего элемента.

Исходный код см. в приложении А.

Тестирование.

Результаты тестирование представлены в табл. 1.

Таблица 1 — результаты тестирования.

№ п-п	Входные данные	Выходные данные	Комментарий
1	+aa	1)(a + a) 2) (2*a)	Формула дерева-имеет вид (a + a)
2	-c+88	1)(c – (8 + 8)) 2) (c-(2*8))	В формуле-дереве присутствует поддерево (8 + 8)
3	-++aa+aa	1)((a + a)) - (a + a)) 2)((2* a) - (2* a))	Будут заменены два поддерева (a + a)
4	a	1)a 2)a	Дерево состоит из единственного элемента
5	-3*21	1)(3 — (2 * 1)) 2)(3 — (2* 1))	Преобразований не требуется
6	-4+*bb*bb	1)(4-((b*b) +(b*b)) 2)(4-(2*(b*b))	Поддерево ((b*b) + (b*b)) преобразовано
7	-3*a-a1	1)(3-(a*(a-1)) 2)(3-(a*(a-1))	Отсутствуют поддеревья вида (f+ f)
8	-+aa-bb	1)((a+a) – (b – b)) 2)((2*a) – (b - b))	(a + a) заменено
9	--aa-aa	1)((a-a)-(a-a)) 2)((a-a)-(a-a))	Замены не требуется
10	*00	1)(0*0) 2(0*0)	Замены не требуется

Выводы.

Был изучен принцип бинарных деревьев. Были получены навыки разработки алгоритмов, работающих с бинарными деревьями.

ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД ПРОГРАММЫ

```
#include <iostream>
#include <cstring>

class BinTree {
public:
  int size = 100;
  char* elementsArray = new char[size]; //массив, в котором хранятся
элементы деерва
  BinTree() {
    for (int i = 0; i < size; i++)
      elementsArray[i] = '_'; //инициализируется "пустым" символом
  }
  friend void printLKP(BinTree* b, int index);
  bool isLeaf(int index) {
    return (!strchr("+-*", elementsArray[index])); //функция проверяет,
является ли элемент "листом"
  }
  int getRightElemIndex(int index) {
    return index * 2 + 2; //возвращает индекс правого потомка
  }
  int getLeftElemIndex(int index) {
    return index * 2 + 1; //возвращает индекс левого потомка
  }

  bool areEqualElements(int index1, int index2) { //проверка идентичности
двух элементов


    if (isLeaf(index1) && isLeaf(index2)) {

      if
        (elementsArray[index1] == elementsArray[index2])
      {
        std::cout << "Элементы-листья " << elementsArray[index1] << " и "
<< elementsArray[index2] << " равны\n";
        return true;
      }
      else {
        std::cout << "Элементы-листья " << elementsArray[index1] << " и "
<< elementsArray[index2] << " не равны\n";
        return false;
      }
    }

    else if (isLeaf(index1) == isLeaf(index2)) {
```

```
      std::cout << "Проверка ";//если оба элемента поддеревья, то далее
проверяются их потомки
      printLKP(this, index1);
      std::cout << " и ";
      printLKP(this, index2);
      return areEqualElements(getLeftElemIndex(index1),
getLeftElemIndex(index2))
        && areEqualElements(getRightElemIndex(index1),
getRightElemIndex(index2));
    }
    else return false;
  }

  char sign(int i) {

      std::cout << "Знак  " << "\"" <<elementsArray[i] << "\"\n";
      return elementsArray[i];

  }

  void transform() { //замена поддеревьев вида (f + f) на (2 * f)
    int i = 0;
    for (int i = 0; i < size; i++)
      if (!isLeaf(i)) {
        std::cout << "Проверка поддерева: ";
        printLKP(this, i);
        std::cout << "\n";
        std::cout << "Проверка равенства элементов ";
        printLKP(this, getLeftElemIndex(i));
        std::cout << " и ";
        printLKP(this, getRightElemIndex(i));
        std::cout << "\n";
        if (areEqualElements(getLeftElemIndex(i), getRightElemIndex(i))
&& sign(i) == '+') {

          std::cout << "Поддерево ";
          printLKP(this, i);
          std::cout << " соответствует виду (f+f). Производится замена на
";
          elementsArray[i * 2 + 1] = '2';
          elementsArray[getRightElemIndex(i * 2 + 1)] = '_';
          elementsArray[getLeftElemIndex(i * 2 + 1)] = '_';
          elementsArray[i] = '*';
          printLKP(this, i);
          std::cout << "\n";
        }
        else {
          std::cout << "Поддерево ";
          printLKP(this, i);
          std::cout << " не соответствуют формуле (f + f)\n";
```

```
        }
      }
  }
};
void readBinTree(BinTree* b, int index, FILE* f);


void printLKP(BinTree* b, int index = 0) {//вывод дерева-формула

  if (index < b->size && b->elementsArray[index] != '_') {

    if (!b->isLeaf(index))
      std::cout << "(";
    printLKP(b, index * 2 + 1);
    std::cout << b->elementsArray[index];
    printLKP(b, index * 2 + 2);
    if (!b->isLeaf(index))
      std::cout << ")";
  }
}

//функции для считывания дерева
void readLeftElem(BinTree* b, int index, FILE* f) {


  b->elementsArray[index * 2 + 1] = fgetc(f);
  std::cout << "Левый потомок = " << b->elementsArray[index * 2 + 1] <<
"\n";
  readBinTree(b, index * 2 + 1, f);
}
void readRightElem(BinTree* b, int index, FILE* f) {
  b->elementsArray[index * 2 + 2] = fgetc(f);
  std::cout << "Правый потомок = " << b->elementsArray[index * 2 +
2]<<"\n";
  readBinTree(b, index * 2 + 2, f);

}
void readBinTree(BinTree* b, int index, FILE* f) {

    if (index == 0) {

      b->elementsArray[index] = fgetc(f);
      std::cout << "\nПостроение дерева-формулы\n";
    }

  if (strchr("+-*", b->elementsArray[index])) {
    std::cout << "Считывание левого потомка для элемента " << b-
>elementsArray[index] << "\n";
    readLeftElem(b, index, f);
```

```
    std::cout << "Считывание правого потомка для элемента " << b-
>elementsArray[index] << "\n";
    readRightElem(b, index, f);
    std::cout<<"Построено поддерево ";
    printLKP(b,index);
    std::cout << "\n";
  }

}

int main()
{


  BinTree bt;
  char* c = new char[300];
  char* a = new char[20];
  setlocale(LC_ALL, "Russian");
  while (strcmp(a, "1\n") && strcmp(a, "2\n")) {
    std::cout << "Обработка бинарного дерева. Введите 1 для ввода дерева
с консоли или 2 для ввода с файла\n";
    fgets(a, 20, stdin);
  }
  if (strcmp(a, "1\n") == 0) {
    std::cout << "Введите дерево в префиксной форме: ";
    readBinTree(&bt, 0, stdin);
  }
  else {
    FILE* f1 = fopen("test.txt", "r+");
    std::cout << "Содержимое файла: ";
    while (fgets(c, 300, f1)) std::cout << c;
    f1 = fopen("test.txt", "r+");
    std::cout << "\n";
    readBinTree(&bt, 0, f1);
    fclose(f1);
  }

  std::cout << "Дерево-формула: ";
  printLKP(&bt);

  std::cout << "\n\nВывод преобразованного дерева-формулы с заменой
поддеревьев вида (f + f) на (2 * f):\n";
  bt.transform();
  std::cout << "Результат: ";
  printLKP(&bt, 0);
  std::cout << "\n";

}
```