Отчет по лабораторной работе 24.

Студент группы М80-108БВ-24 № по списку 12

Контакты e-mail klimkucin51@gmail.com

Работа выполнена: «26» мая 2025 г. Преподаватель: каф. 806 Бучкин Тимур Артемович Входной контроль знаний с оценкой \_ \_ \_

Отчет сдан «26» мая 2025 г., итоговая оценка \_ \_ \_

Подпись преподавателя \_ \_ \_

1. Тема: Динамические структуры данных. Обработка бинарных деревьев

2. Цель работы: Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев. Преобразование выражения в дерево рекомендуется осуществлять одним из известных методов (Рутисхаузера, Дейкстры и др.). Операнды в обрабатываемых выражениях могут быть целого или веще ственного типа (по усмотрению преподавателя). Задания могут быть переформулированы и для булевско- го типа. Преобразование выражения реализовать в виде набора подпрограмм. Программа должна вводить и печатать выражения в исходном (текстовом) виде, преобразовывать их в деревья, выполнять заданные преобразования путем обращения к подпрограммам и печатать результаты в виде дерева и в текстовом представлении.3.   
Задание: Вывести общие сомножители из разности

4. Оборудование asus vivobook

5. Программное обеспечение ubuntu

Система программирования: нет. Редактор текстов: VSCode.

6. Идея, метод, алгоритм решения задачи (в формах: словесной, псевдокода, графической [блок- схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями):

Идея: Сбор переменных, замена, изменения дерева для добавления умножения

Оценка сложности алгоритма: O(n^2) где количество размер таблицы

7. протокол:

Итоговый алгоритм, уже использующий функции написанные ранее

#include "../include/ExprTree.h"

2

#include <stdlib.h>

3

4

typedef struct Node\_ {

5

Token token;

6

struct Node\_ \*left;

7

struct Node\_ \*right;

8

9

struct Node\_ \*parent;

10

} Node;

11

12

struct ExprTree\_ {

13

Node \* root;

14

};

15

16

Node \* Node\_create(Token token, Node \* parent) {

17

Node \* node = (Node \*)malloc(sizeof(Node));

18

node->token = token;

19

node->left = NULL;

20

node->right = NULL;

21

node->parent = parent;

22

23

return node;

24

}

25

26

Node \* prefixForm2Tree(Tokens \* prefixForm, Node \* parent) {

27

if(Tokens\_empty(prefixForm)) {

28

return NULL;

29

}

30

Node \* node = Node\_create(\*Tokens\_top(prefixForm), parent);

31

Tokens\_pop(prefixForm);

32

if(node->token.type != TOKEN\_OPERATION)

33

return node;

34

35

node->right = prefixForm2Tree(prefixForm, node);

36

if(node->token.data.op != OP\_UNARY\_MINUS) {

37

node->left = prefixForm2Tree(prefixForm, node);

38

}

39

40

return node;

41

}

42

43

void destroyNode(Node \*miidle) {

44

if (!miidle->left || !miidle->right)

45

{

46

miidle->left = NULL;

47

miidle->parent = NULL;

48

miidle->right = NULL;

49

50

return;

51

}

52

53

destroyNode(miidle->left);

54

destroyNode(miidle->right);

55

free(miidle);

56

}

57

58

ExprTree \* TreeCrete(Tokens \* prefixForm) {

59

ExprTree \* tree = (ExprTree \*)malloc(sizeof(ExprTree));

60

tree->root = prefixForm2Tree(prefixForm, NULL);

61

62

return tree;

63

}

64

65

void TreePrint\_impl(Node \* node, int depth) {

66

if(!node)

67

return;

68

69

TreePrint\_impl(node->right, depth + 1);

70

for(size\_t i = 0; i < depth; ++i)

71

printf("->");

72

printToken(node->token);

73

printf("\n");

74

TreePrint\_impl(node->left, depth + 1);

75

}

76

77

void TreePrintExpr\_impl(Node \* node) {

78

if(!node) {

79

return;

80

}

81

if(node->left) {

82

if(

83

node->left->token.type == TOKEN\_OPERATION &&

84

getOpPriority(node->token.data.op) > getOpPriority(node->left->token.data.op)

85

) {

86

printf("(");

87

TreePrintExpr\_impl(node->left);

88

printf(")");

89

} else {

90

TreePrintExpr\_impl(node->left);

91

}

92

}

93

printToken(node->token);

94

if(node->right) {

95

if(

96

node->right->token.type == TOKEN\_OPERATION &&

97

getOpPriority(node->token.data.op) > getOpPriority(node->right->token.data.op)

98

) {

99

printf("(");

100

TreePrintExpr\_impl(node->right);

101

printf(")");

102

} else {

103

TreePrintExpr\_impl(node->right);

104

}

105

}

106

}

107

108

void GetRoot(ExprTree \* tree) {

109

SearchPow(tree->root->left, tree);

110

}

111

112

void ParrentAndBaby(Node \*destroyBaby)

113

{

114

115

Token newNodeToken;

116

Node \*parent = destroyBaby->parent;

117

Node \*grandParent = parent->parent;

118

119

if (parent->right == destroyBaby)

120

{

121

newNodeToken = parent->left->token;

122

}

123

else if (parent->left == destroyBaby)

124

{

125

newNodeToken = parent->right->token;

126

}

127

destroyNode(parent);

128

Node\_create(newNodeToken, grandParent);

129

}

130

131

132

133

void SearchPow(Node \* node, ExprTree \* tree){

134

if (node->token.type == TOKEN\_VARIABLE || node->token.type == TOKEN\_NUMBER){

135

PowToMult(node, tree->root);

136

137

}

138

else{

139

if (node->left)

140

SearchPow(node->left, tree);

141

if (node->right)

142

SearchPow(node->right, tree);

143

}

144

return;

145

}

146

147

Node \* searchCopy(Node \* node, Node \* copy){

148

if (node->token.data.var == copy->token.data.var || node->token.data.num == copy->token.data.num ){

149

return copy;

150

}

151

else{

152

if (copy->left)

153

return searchCopy(node, copy->left);

154

if (copy->right)

155

return searchCopy(node, copy->left);

156

}

157

}

158

159

void PowToMult(Node \* node, Node \* root){

160

Node \* copy = searchCopy(node, root->right);

161

if (copy){

162

Token Mult = {

163

.type = TOKEN\_OPERATION,

164

.data = {

165

.op = OP\_MULT,

166

}

167

};

168

root->parent = Node\_create(Mult, NULL);

169

Token NewToken = node->token;

170

ParrentAndBaby(node);

171

ParrentAndBaby(copy);

172

Node\_create(NewToken, root->parent);

173

174

}

175

}

176

177

178

void TreePrint(ExprTree \* tree) {

179

TreePrint\_impl(tree->root, 0);

180

}

181

182

void TreePrintExpr(ExprTree \* tree) {

183

TreePrintExpr\_impl(tree->root);

184

}

8. Вывод: В результате работы я улучшил своё понимание бинарных деревьев

Подпись студента: \_ \_ \_