**Отчет по лабораторной работе № 24** по курсу

«Языки и методы программирования»

Студент группы М8О-112Б-21 Орешкин Максим Алексеевич, № по списку \_\_\_13\_\_

Контакты www, e-mail: maks-oreh03@mail.ru

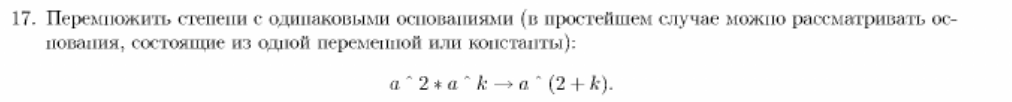
Работа выполнена: « 18 » марта 2021 г.

Преподаватель: доцент каф. 806 \_Никулин С.П\_\_\_\_\_\_\_\_\_\_\_\_\_

Входной контроль знаний с оценкой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отчет сдан « » \_\_\_\_\_\_\_\_\_202\_ г., итоговая оценка \_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Тема:** Динамические структуры данных. Обработка деревьев.
2. **Цель работы:** Научиться создавать и работать с бинарным деревом.
3. **Задание:** **Оборудование** :

ЭВМ , процессор , имя узла сети с ОП ГБ,

НМД ГБ. Терминал адрес . Принтер

Другие устройства

*Оборудование ПЭВМ студента, если использовалось:*

 Процессор intel core i7, с   ОП 8 ГБ, НМД  120832 МБ. Монитор  15,6 /1920\*1820

Другие устройства

1. **Программное обеспечение:**

Операционная система семейства , наименование версия интерпретатор команд версия

Система программирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ версия \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Редактор текстов версия

Утилиты операционной системы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы

Местонахождение и имена файлов программ и данных

*Программное обеспечение ЭВМ студента, если использовалось:*

Операционная система семейства Unix , наименование Ubuntu версия 20.04

интерпретатор команд bash версия 4.4.18

Система программирования версия

Редактор текстов vim версия 8.1

Утилиты операционной системы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Прикладные системы и программы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Местонахождение и имена файлов программ и данных на домашнем компьютере \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Идея, метод, алгоритм**  решения задачи (в формах: словесной, псевдокода, графической [блок-схема, диаграмма, рисунок, таблица] или формальные спецификации с пред- и постусловиями)
2. Ищем операцию с самым низким приоритетом, это операция будет являться корнем бинарного дерева. Эту операцию ищем для того, чтобы разбить уравнение на две части.
3. Записываем в бинарное дерево все выражение.
4. Чтобы преобразовать выражения из вида 2^5\*2^6 = > 2^(5+6), то мы сравниваем левые узлы (внуки) и если они одинаковые, то изменяем ссылку, чтобы левый внук стал отцом, корень стал степенью, правый отец стал операцией сложения, а его дети стали степенью числа.

1. **Сценарий выполнения работы** [план работы, первоначальный текст программы в черновике (можно на отдельном листе) и тесты либо соображения по тестированию].

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

int kol=-1;

struct StrNode {

char data [50];

struct StrNode \*left;

struct StrNode \*right;

};

typedef struct StrNode \*node;

int define\_priority (char a) {

switch (a){

case '-':

case '+':

return 1;

case '\*':

case '/':

return 2;

case '^':

return 4;

default:

return 100;

}

}

node MakeTree (char expr[], int first, int last) {

int prior, MinPrior = 100, k, depth = 0;

node Tree = (node) malloc (sizeof(struct StrNode));

for (int i = first; i <= last; ++i) {

if(expr[i] == '^'){

kol+=1;

}

if (expr[i] == '(') {

depth++;

continue;

}

if (expr[i] == ')') {

depth--;

continue;

}

if (depth>0){

continue;

}

prior = define\_priority(expr[i]);

if (prior <= MinPrior) {

MinPrior = prior;

k = i;

}

}

if (depth !=0) {

printf("Wrong expression!\n");

exit (1);

}

int l;

if (MinPrior == 100) {

if (expr[first] == '(' && expr[last] == ')') {

free(Tree);

return MakeTree(expr, first +1, last - 1);

}

else {

l = last - first + 1;

for (int i = 0; i < l; i++ ) {

Tree->data[i] = expr[first+i];

}

Tree->data[l] = '\n';

Tree->left = NULL;

Tree->right = NULL;

return Tree;

}

}

Tree->data[0] = expr[k];

Tree->data[1] = '\n';

Tree->left = MakeTree(expr, first, k-1);

Tree->right = MakeTree(expr, k+1, last);

return Tree;

}

void print\_tree(node tr, int l) {

if (tr->right != NULL) print\_tree(tr->right, l+1);

for(int i = 0; i < l; ++i) {

printf(" ");

}

printf("%5s", tr->data);

if (tr->left != NULL) print\_tree(tr->left, l+1);

}

void print\_expression(node t) {

if (t==NULL) {

return;

}

if (define\_priority(t->data[0])!=100 && define\_priority(t->left->data[0])!=100 && define\_priority(t->data[0])

> define\_priority(t->left->data[0]) || t->data[0] == '^' && t->left->data[0] == '^' ){

printf("(");

print\_expression(t->left);

printf(")");

} else print\_expression(t->left);

for (int i = 0; i < 50; ++i) {

if (t->data[i] == '\n') {

break;

}

printf("%c", t->data[i]);

}

if (define\_priority(t->data[0])!=100 && define\_priority(t->right->data[0])!=100 && define\_priority(t->data[0]

) > define\_priority(t->right->data[0]) || t->data[0] == '^' && t->right->data[0] == '^' ){

printf("(");

print\_expression(t->right);

printf(")");

} else print\_expression(t->right);

}

node copy (node t) {

if (t == NULL) {

return NULL;

}

node new\_t = (node) malloc (sizeof(struct StrNode));

for (int i = 0; i<50; ++i) {

new\_t->data[i] = t->data[i];

}

new\_t ->left = copy(t->left);

new\_t->right = copy(t->right);

return new\_t;

}

node transformation (node \* t) {

if (\*t == NULL) {

return NULL;

}

if ((\*t)->data[0] == '\*' && (\*t)->right->data[0] == '^' && (\*t)->left->data[0]=='^' && (\*t)->left->left->data[0] == (\*t)->right->left->data[0]) {

(\*t)->data[0] = '^';

node base1\_left = copy((\*t)->left->left);

node base1\_right = copy((\*t)->left->right);

node base2\_l = copy(base1\_left);

node base2\_r = copy(base1\_right);

node degree = copy((\*t)->right);

\*(\*t)->right->data = '+';

(\*t)->right->left = base2\_r;

(\*t)->left = base2\_l;

}

(\*t)->left = transformation(&((\*t)->left));

(\*t)->right = transformation(&((\*t)->right));

return \*t;

}

int main(void) {

node t = NULL;

int opt = -1;

while (opt!=5) {

printf("Choose an action:\n 1. Enter an expression and create tree.\n 2. Transforn expression.\n 3. Print tree.\n 4. Print expression.\n 5. Exit\n");

scanf("%d", &opt);

switch (opt) {

case 1: {

printf("Please, enter an expression: ");

char expression[1000];

scanf("%s", expression);

int n = 0;

while (expression[n] != '\0') {

n++;

}

t = MakeTree(expression, 0, n-1);

break;

}

case 2: {

for(int i=0;i<kol;i++)

t = transformation(&t);

break;

}

case 3: {

printf("\n");

print\_tree(t, 0);

break;

}

case 4: {

printf("\n");

print\_expression(t);

printf("\n");

break;

}

}

}

return 0;

}

*Пункты 1-7 отчета составляются строго до начала лабораторной работы.*

*Допущен к выполнению работы.*  **Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Распечатка протокола**  (подклеить листинг окончательного варианта программы с тестовыми примерами, подписанный преподавателем).

maxim@vb:~$ gcc 24.c

maxim@vb:~$ ./a.out

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

1

Please, enter an expression: 2^5\*3\*5

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

5

\*

3

\*

5

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

5

\*

3

\*

5

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

4

2^5\*3\*5

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

1

Please, enter an expression: 2^3\*2^4

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

4

^

2

\*

3

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

4

+

3

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

4

2^(3+4)

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

1

Please, enter an expression: 2^3\*2^a\*2\*4

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

4

\*

2

\*

a

+

3

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

4

2^(3+a)\*2\*4

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

1

Please, enter an expression: (2^5\*2^6)\*(3^5\*3^7)

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

7

^

3

\*

5

^

3

\*

6

^

2

\*

5

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

7

+

5

^

3

\*

6

+

5

^

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

4

2^(5+6)\*3^(5+7)

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

1

Please, enter an expression: (5^12\*5^13)\*(4^a\*4^b)\*(7^a\*7^13)

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

13

^

7

\*

a

^

7

\*

b

^

4

\*

a

^

4

\*

13

^

5

\*

12

^

5

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

2

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

3

13

+

a

^

7

\*

b

+

a

^

4

\*

13

+

12

^

5

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

4

5^(12+13)\*4^(a+b)\*7^(a+13)

Choose an action:

1. Enter an expression and create tree.

2. Transforn expression.

3. Print tree.

4. Print expression.

5. Exit

5

**Дневник отладки** должен содержать дату и время сеансов отладки и основные события (ошибки в сценарии и программе, нестандартные ситуации) и краткие комментарии к ним. В дневнике отладки приводятся сведения об использовании других ЭВМ, существенном участии преподавателя и других лиц в написании и отладке программы.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № | Лаб. или дом. | Дата | Время | Событие | Действие по исправлению | Примечание |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |

1. **Замечания автора** по существу работы \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 11. **Выводы**

Во время выполнения лабораторной работы, я научился работать с бинарными деревьями иразбирать арифметические выражения с помощью бинарных деревьев.

Подпись студента\_\_\_\_\_\_\_\_\_\_\_\_