Отчёт по лабораторной работе № 24 по курсу 1

студента группы М8О-108-19 Горохов Михаил Антонович, № по списку 7

Адреса www, e-mail, jabber, skype ma.gorokhoov@gmail.com

Работа выполнена: "21" апреля 2020г.

Преподаватель: Поповкин Александр Викторович каф.806

Входной контроль знаний с оценкой _____

Отчёт сдан "21" апреля 2020г., итоговая оценка _____

Подпись преподавателя _____

1. **Тема**: Алгоритмы и структуры данных. Практическое применение деревьев.

2. **Цель работы**: Составить программу выполнения заданных преобразований арифметических выражений с применением деревьев.

3. **Задание** (*вариант № 7*): Редуцировать выражения, заменив операцию умножения переменной на целое число n на сумму n слогаемых.

4. **Оборудование** (*лабораторное*):
   ЭВМ ---, процессор Intel(R) Pentium(R) CPU G2140 @ 3.30GHz, имя узла сети alice5 с ОП 8192 МБ НМД 350 ГБ. Терминал --- адрес 192.168.2.105. Принтер _____
   Другие устройства ---

   Оборудование ПЭВМ студента, если использовалось:
   Процессор Ryzen 3 2200U 2.5 GHz, ОП 4096 МБ, НМД 128 ГБ. Монитор ---
   Другие устройства ---

5. **Программное обеспечение** (*лабораторное*):
   Операционная система семейства Linux, наименование Ubuntu версия 18.04.3 LTS
   Интерпретатор команд Bash версия 4.4.19(1)-release
   Система программирования GCC версия ---
   Редактор текстов GNU Emacs версия 25.2.2
   Утилиты операционной системы _____
   Прикладные системы и программы _____
   Местонахождения и имена файлов программ и данных _____

   Программное обеспечение ЭВМ студента, если использовалось:
   Операционная система семейства Linux, наименование Xubuntu версия 18.04 bionic
   Интерпретатор команд Bash версия 5.0.9(1)-release
   Система программирования GCC версия ---
   Редактор текстов vim версия 8.1
   Утилиты операционной системы _____
   Прикладные системы и программы _____
   Местонахождения и имена файлов программ и данных _____

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

Удобно будет ввести три структуры данных: Дек, Дерево, Узел.
Дек удобно использовать, так как нам нужно будут и особенности очереди и особенности стека. Дек своего рода швецарский нож, элементы которого являются математические узлы (Node).
Очень удобно, чтобы и дек и дерево состояло из одних и тех же элементов, из узлов.
Удобно будет запомнить строку, введенную пользователем.

Мной был составлен ToDo list, помогающий мне ориентироваться в прогрессе лабораторной работы.
В этом ToDo list описан кратко алгоритм, полеченный из разбиения лабораторной работы на подзадачи:
[+] 1. Реализовать структуры данных дек, дерево и узел и функции для них.
[+] 2. Считать строку из stdin.
[+] 3. Выделить из строки узлы: числа, литеры, операторы. Положить в стек input.
[+] 4. Реализовать сортировочную станцию. Получить стек output с узлами в префиксной форме.
[+] 5. Получить из стека output дерево префиксной формы.
[+] 6. Провести замену узлов "*" в дереве на n узлов "+". Реализовать через рекурсивную функцию.
[+] 7. Реализовать вывод дерева в инфиксной форме.
[ ] 8. Написать отчет по лабораторной работе 24.

Использовался алгоритм сортировочной станции Дейкстры.
Более подробный алгоритм предоставлен в сценарии работы.

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

Код библиотеки lib24.c

```
#ifndef __lib24.h__
#define __lib24.h__

#include <stdio.h>
#include <stdlib.h>

#define NUM -3
#define LIT -2

typedef int T;
typedef struct Node {
   // priority: -3 - number; -2 - litter; 0 - +-; 1 - */; 2 - ^; 3 - ();
   short priority;
   T value;
   struct Node* left;
   struct Node* right;
   struct Node* parrent;
} Node;
typedef struct {
   Node* root;
} Tree;
typedef struct {
   int size;
   Node* first;
   Node* last;
} Deque;

Node* makeNode(T value, short priority) {
   Node* node = malloc(sizeof(Node));
   node->value = value;
   node->priority = priority;
   node->left = NULL;
   node->right = NULL;
   node->parrent = NULL;

   return node;
}
void destroyNode(Node* node) {
   if(node->left  != NULL) destroyNode(node->left);
   if(node->right != NULL) destroyNode(node->right);
   free(node);
   node = NULL;
   return;
}
void createDeque(Deque* dq) {
   dq->size = 0;
```

```
    dq->first = NULL;
    dq->last = NULL;
}
void push_front(Deque* dq, Node* node) {
    if(dq->size == 0) {
        dq->first = node;
        dq->last = node;
        dq->size++;
        return;
    }
    else {
        node->right = dq->first;
        dq->first->left = node;
        dq->first = node;
        dq->size++;
        return;
    }
}
void push_back(Deque* dq, Node* node) {
    if(dq->size == 0) {
        dq->last = node;
        dq->first = node;
        dq->size++;
        return;
    }
    else {
        node->left = dq->last;
        dq->last->right = node;
        dq->last = node;
        dq->size++;
        return;
    }
}
Node* pop_front(Deque* dq) {
    if(dq->size == 0) {
        printf("Error! There is no Nodes. Get this kek\n");
        return NULL;
    }
    else if (dq->size == 1) {
        Node* node;
        node = dq->first;
        dq->first = NULL;
        dq->last = NULL;
        dq->size--;

        return node;
    }
    else {
        Node* node;
        node = dq->first;
        dq->first = node->right;
        dq->size--;

        return node;
    }
}
Node* pop_back(Deque* dq) {
    if(dq->size == 0) {
        printf("Error! There is no Nodes. Get this kek\n");
        return NULL;
    }
    else if (dq->size == 1) {
        Node* node;
        node = dq->last;
```

```
        dq->first = NULL;
        dq->last = NULL;
        dq->size--;

        return node;
    }
    else {
        Node* node;
        node = dq->last;
        dq->last = node->left;
        dq->size--;

        return node;
    }
}
void printDeque(Deque* dq) {
    Node* node = dq->first;
    for(int i=0; i < dq->size; i++) {
        if (node->priority == NUM) {
            printf("%d|", node->value);
        }
        else {printf("%c|", node->value);}
        node = node->right;
    }
    printf("\n");
    return;
}

void sortingStation(Deque* input, Deque* output) {
    Deque* stack = malloc(sizeof(Deque));
    createDeque(stack);
    Node* node = NULL;
    while(input->size != 0){
        node = pop_front(input);
        if(node->priority == NUM || node->priority == LIT) {
            push_back(output, node);
            continue;
        }
        // That is operator
        if(node->value == '(') {
            push_back(stack, node);
            continue;
        }
        if(node->value == ')') {
            while(stack->last->value != '(') {
                push_back(output, pop_back(stack));
            }
            // Pop '('
            pop_back(stack);
            continue;
        }
        while(stack->size != 0 && stack->last->priority >= node->priority) {
            push_back(output, pop_back(stack));
        }
        if(stack->size == 0 || stack->last->priority < node->priority) {
            push_back(stack, node);
            continue;
        }
    }
    while(stack->size != 0) {
        push_back(output, pop_back(stack));
    }

    free(stack);
```

```
    return;
}
void buildPartTree(Node* node, Deque* dq) {
    if(node->right == NULL) {
        Node* dqnode = pop_back(dq);

        dqnode->right = NULL;
        dqnode->left = NULL;
        dqnode->parrent = node;

        node->right = dqnode;
        if(0 <= dqnode->priority && dqnode->priority <= 2) {
            buildPartTree(node->right, dq);
        }
    }
    if(node->left == NULL) {
        Node* dqnode = pop_back(dq);

        dqnode->right = NULL;
        dqnode->left = NULL;
        dqnode->parrent = node;

        node->left = dqnode;
        if(0 <= dqnode->priority && dqnode->priority <= 2) {
            buildPartTree(node->left, dq);
        }
    }
}
void buildTree(Tree* tree, Deque* dq) {
    Node* node = pop_back(dq);
    node->left = NULL;
    node->right = NULL;
    tree->root = node;
    buildPartTree(node, dq);
}
Node* getCopyNode(Node* node) {
    Node* new_node = malloc(sizeof(Node));
    if(node->right != NULL) new_node->right = getCopyNode(node->right);
    if(node->left  != NULL) new_node->left  = getCopyNode(node->left);
    new_node->parrent = NULL;
    new_node->value = node->value;
    new_node->priority = node->priority;

    return new_node;
}
void replaceNodesTree(Node* node) {
    if(node->left  != NULL) replaceNodesTree(node->left);
    else return;
    if(node->right != NULL) replaceNodesTree(node->right);
    else return;

    if(node->left->priority == NUM && node->right->priority == NUM) return;

    if(node->value == '*' && node->left->priority == NUM && node->right->priority != NUM) {
        Node* nodetmp = node->left;
        node->left = node->right;
        node->right = nodetmp;
    }
    // Здесь явно что-то плохо написано
    if(node->priority == 1 && node->value == '*') {
        if (node->right->priority == NUM) {
            if(node->right->value > 1) {
                Node* node_operation = makeNode('*', 1);
                Node* node_left = getCopyNode(node->left);
```

```
            Node* node_right = makeNode(node->right->value-1, NUM);
            node_left->parrent = node_operation;
            node_right->parrent = node_operation;

            node_operation->left = node_left;
            node_operation->right = node_right;

            destroyNode(node->right);
            node->right = node_operation;
            node_operation->parrent = node;

            node->value = '+';
            node->priority = 0;
            replaceNodesTree(node->right);
         }
         else if(node->right->value == 1) {
            Node* nodetmp = node->left;

            destroyNode(node->right);

            node->right = nodetmp->right;
            node->left = nodetmp->left;
            node->priority = nodetmp->priority;
            node->value = nodetmp->value;
            free(nodetmp);
            /*
            node->left = NULL;
            nodetmp->parrent = node->parrent;
            // Сын говорит отцу, что Сын - сын Отца
            if(nodetmp->parrent->right == node) nodetmp->parrent->right == nodetmp;
            else if(nodetmp->parrent->left == node) nodetmp->parrent->left == nodetmp;
            destroyNode(node);
            node = nodetmp;
            */
         }
         else if(node->right->value == 0) {
            destroyNode(node->right);
            destroyNode(node->left);
            node->right = NULL;
            node->left = NULL;

            node->priority = NUM;
            node->value = 0;
         }
      }
   }
}
void replaceNodes(Tree* tree) {
   replaceNodesTree(tree->root);
   return;
}
void printNode(Node* node, int tabs) {
   for(int i=0; i < tabs; i++) {
      printf("\t");
   }
   if ((0 <= node->priority && node->priority <= 2) || node->priority == LIT) {
      printf("%c\n", node->value);
   }
   else {
      printf("%d\n", node->value);
   }

   if(node->right != NULL) printNode(node->right, tabs+1);
   if(node->left  != NULL) printNode(node->left, tabs+1);
```

```
    return;
}
void printTree(Tree* tree) {
    printNode(tree->root, 0);
    return;
}
void printInfixNode(Node* node) {
    if(node->left != NULL && node->right != NULL) {
        printf("(");
        printInfixNode(node->left);
        printf("%c", node->value);
        printInfixNode(node->right);
        printf(")");
    }
    else {
        if (node->priority == NUM) printf("%d", node->value);
        if (node->priority == LIT) printf("%c", node->value);
    }
    return;
}
void printInfixTree(Tree* tree) {
    printInfixNode(tree->root);
    printf("\n");
}
long char_to_number(char c) {
    return c - '0';
}
long str_to_number(char* str, int size) {
    long result = 0;
    for(int i=0; i < size; i++) {
        result *= 10;
        result += char_to_number(str[i]);
    }

    return result;
}
void DequeFromString(Deque* in, char* infix, int size) {
    char* numbers = malloc(sizeof(char)*size);
    int j=0;
    Node* node;
    //printf("Захожу в цикл i<size\n");
    for(int i=0; i < size; i++) {
        //printf("Обработка символа %c\n", infix[i]);
        if(infix[i] == ' ') continue;

        if('0' <= infix[i] && infix[i] <= '9') {
            //printf("О, цифра\n");
            numbers[j] = infix[i];
            j++;
            continue;
        }
        else if('a' <= infix[i] && infix[i] <= 'z') {
            //printf("a-z\n");
            node = makeNode((T)infix[i], LIT);
            push_back(in, node);
        }
        else if((infix[i] == '+') || infix[i] == '-' ||
                infix[i] == '*' || infix[i] == '/' ||
                infix[i] == '^' || infix[i] == '%' || infix[i] == '(' || infix[i] == ')') {
            // Also, if last char is number
            if(j > 0) {
                long value = str_to_number(numbers, j);
                j = 0;
                node = makeNode(value, NUM);
```

```
            push_back(in, node);
        }
        short priority = 4;
        switch(infix[i]) {
            case '+': priority = 0; break;
            case '-': priority = 0; break;
            case '*': priority = 1; break;
            case '/': priority = 1; break;
            case '^': priority = 2; break;
            // Почему при priority >=3 скобки в sortingStation обваливаются, а при <0 ничего плохого не
происходит, и все работает?
            case '(': priority = -4; break;
            case ')': priority = -4; break;
            default: printf("Bad char in switch!!!\n");
        }
        node = makeNode((T)infix[i], priority);
        push_back(in, node);
    }
  }
  // Last char maybe is number
  if(j > 0) {
    long value = str_to_number(numbers, j);
    j = 0;
    node = makeNode(value, NUM);
    push_back(in, node);
  }
  free(numbers);
  free(infix);
}

#endif
```

Код основной части программы main.c:

```
#include <stdio.h>
#include <stdlib.h>

#include "lib24.cpp"

int main() {
  int size;
  char* infix;
  Deque in;
  Deque out;

  createDeque(&in);
  createDeque(&out);
  // Запоминаю вводимую строку
  char buffer;
  size = 0;
  unsigned long memSize = 1;
  infix = (char *) malloc(sizeof(char));
  while((buffer = getchar()) != EOF) {
    if (size >= memSize) {
      memSize *= 2;
      infix = (char*)realloc(infix, memSize * sizeof(char));
    }
    infix[size] = buffer;
    size++;
  };
  infix = (char *) realloc(infix, size * sizeof(char));
  // Запомнил
  //printf("%s", infix);
  // Ок!
```

```
    // Выделяю из строки узлы.
    DequeFromString(&in, infix, size);
    printf("Создан стек!\n");
    // Проверяю содержимое
    //printDeque(&in);
    // Сортировочная станция
    sortingStation(&in, &out);
    printf("Отсортировал\n");
    //printDeque(&out);

    // Строим дерево
    Tree tree;
    buildTree(&tree, &out);
    printf("Дерево построилось без ошибок\n");
    //printTree(&tree);

    // Заменяю узлы
    replaceNodes(&tree);
    //printTree(&tree);

    // Вывожу результат
    printInfixTree(&tree);
    return 0;
}
```

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

Допущен к выполнению работы. Подпись преподавателя _____

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

Первая часть — с полным описанием выполнения программы, вторая — лаконичная форма вывода этапа и результата работы.

```
[revammark@Raft laba24]$ ./main
1+2*3-4
1+2*3-4
Захожу в цикл i<size
Обработка символа 1
O, цифра
Обработка символа +
Обработка символа 2
O, цифра
Обработка символа *
Обработка символа 3
O, цифра
Обработка символа -
Обработка символа 4
O, цифра
Обработка символа

Создан стек!
1|+|2|*|3|-|4|
Отсортировал
1|2|3|*|+|4|-|
Дерево построилось без ошибок
-
      4
      +
            *
                  3
                  2
            1
-
      4
```

```
    +
        *
            3
            2
        1
((1+(2*3))-4)
[revammark@Raft laba24]$ ./main
2+a*4-9*b
2+a*4-9*b
Захожу в цикл i<size
Обработка символа 2
O, цифра
Обработка символа +
Обработка символа a
a-z
Обработка символа *
Обработка символа 4
O, цифра
Обработка символа -
Обработка символа 9
O, цифра
Обработка символа *
Обработка символа b
a-z
Обработка символа

Создан стек!
2|+|a|*|4|-|9|*|b|
Отсортировал
2|a|4|*|+|9|b|*|-|
Дерево построилось без ошибок
-
    *
        b
        9
    +
        *
            4
            a
        2
-
    +
        +
            +
                +
                    +
                        +
                            +
                                +
                                    +
                                        b
                                        b
                                    b
                                b
                            b
                        b
                    b
                b
            b
        b
    +
        +
            +
                +
                    a
                    a
```

```
                    a
                a
            2
((2+(a+(a+(a+a))))-(b+(b+(b+(b+(b+(b+(b+b)))))))))
[revammark@Raft laba24]$ ./main
1-2-3-4-4*2342^7878^(a*b*c*d*(89^6))+a*0
1-2-3-4-4*2342^7878^(a*b*c*d*(89^6))+a*0
    Захожу в цикл i<size
Обработка символа 1
O, цифра
Обработка символа -
Обработка символа 2
O, цифра
Обработка символа -
Обработка символа 3
O, цифра
Обработка символа -
Обработка символа 4
O, цифра
Обработка символа -
Обработка символа 4
O, цифра
Обработка символа *
Обработка символа 2
O, цифра
Обработка символа 3
O, цифра
Обработка символа 4
O, цифра
Обработка символа 2
O, цифра
Обработка символа ^
Обработка символа 7
O, цифра
Обработка символа 8
O, цифра
Обработка символа 7
O, цифра
Обработка символа 8
O, цифра
Обработка символа ^
Обработка символа (
Обработка символа a
a-z
Обработка символа *
Обработка символа b
a-z
Обработка символа *
Обработка символа c
a-z
Обработка символа *
Обработка символа d
a-z
Обработка символа *
Обработка символа (
Обработка символа 8
O, цифра
Обработка символа 9
O, цифра
Обработка символа ^
Обработка символа 6
O, цифра
Обработка символа )
Обработка символа )
```

Обработка символа +
Обработка символа a
a-z
Обработка символа *
Обработка символа 0
O, цифра
Обработка символа

Создан стек!
1|-|2|-|3|-|4|-|4|*|2342|^|7878|^|(|a|*|b|*|c|*|d|*|(|89|^|6|)|)|+|a|*|0|
Отсортировал
1|2|-|3|-|4|-|4|2342|7878|^|a|b|*|c|*|d|*|89|6|^|*|^|*|-|a|0|*|+|
Дерево построилось без ошибок

```
+
    *
        0
        a
    -
        *
            ^
                *
                    ^
                        6
                        89
                    *
                        d
                        *
                            c
                            *
                                b
                                a
                ^
                    7878
                    2342
            4
        -
            4
            -
                3
                -
                    2
                    1
+
    0
    -
        +
            +
                +
                    ^
                        *
                            ^
                                6
                                89
                            *
                                d
                                *
                                    c
                                    *
                                        b
                                        a
                        ^
                            7878
                            2342
                    ^
```

```
                        *
                            ^
                                6
                                89
                            *
                                d
                                *
                                    c
                                    *
                                        b
                                        a
                        ^
                            7878
                            2342
                    ^
                        *
                            ^
                                6
                                89
                            *
                                d
                                *
                                    c
                                    *
                                        b
                                        a
                        ^
                            7878
                            2342
                ^
                    *
                        ^
                            6
                            89
                        *
                            d
                            *
                                c
                                *
                                    b
                                    a
                    ^
                        7878
                        2342
            -
                4
                -
                    3
                    -
                        2
                        1
((((((1-2)-3)-4)-(((2342^7878)^((((a*b)*c)*d)*(89^6)))+(((2342^7878)^((((a*b)*c)*d)*(89^6)))+
      (((2342^7878)^((((a*b)*c)*d)*(89^6)))+((2342^7878)^((((a*b)*c)*d)*(89^6)))))))+0)
[revammark@Raft laba24]$ ./main
1+2
Создан стек!
Отсортировал
Дерево построилось без ошибок
(1+2)
[revammark@Raft laba24]$ ./main
1+2*3
Создан стек!
Отсортировал
Дерево построилось без ошибок
```

(1+(2*3))
[revammark@Raft laba24]$ ./main
1*a
Создан стек!
Отсортировал
Дерево построилось без ошибок
a
[revammark@Raft laba24]$ ./main
2*a
Создан стек!
Отсортировал
Дерево построилось без ошибок
(a+a)
[revammark@Raft laba24]$ ./main
12-6*a
Создан стек!
Отсортировал
Дерево построилось без ошибок
(12-(a+(a+(a+(a+(a+a))))))
[revammark@Raft laba24]$ 12-6*a^2
bash: 12-6*a^2: command not found
[revammark@Raft laba24]$ ./main
12-6*a^2
Создан стек!
Отсортировал
Дерево построилось без ошибок
(12-((a^2)+((a^2)+((a^2)+((a^2)+((a^2)+(a^2)))))))
[revammark@Raft laba24]$ date
Вт 21 апр 2020 18:17:14 MSK
[revammark@Raft laba24]$

9. **Дневник отладки** должен содержать дату и время сеансов отладки, и основные ошибки (ошибки в сценарии и программе, не стандартные операции) 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

10. Замечание автора по существу работы Работы очень большая, похожа на итоговый курсовой проект Бакалавриата некоторых вузов нашей страны. Работа заняла порядка 20 часов.

11. Выводы В ходе выполнения этой работы было получено множество знаний и навыков программирования на Си, отточены навыки управления памятью и организации структур данных. В частности, я отточил навыки разработки структур данных дек, дерево и узлов и навыки реализации функций для них, я научился составлять дерево арифметических и логических выражений из строки, проводить операцию над деревом (в частности, редуцировать выражения, заменив операцию умножения переменной на целое число n на сумму n слогаемых). Теперь я гораздо увереннее чувствую себя в разработке программ на Си и не боюсь кода на 450 строчек.

Недочеты, допущенные при выполнении задания, могут быть устранены следующим образом

Подпись студента _____